Basic Library for WPF/Silverlight

2018.04.10 更新

グレープシティ株式会社

目次

製品の概要

ComponentOne for WPF/Silverlight Basic Library はMaskedTextBox、ComboBox、TabControl など、WPF アプリケーションに役立つ入力系コンポーネントのセットです。

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。

> **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。

ComboBox

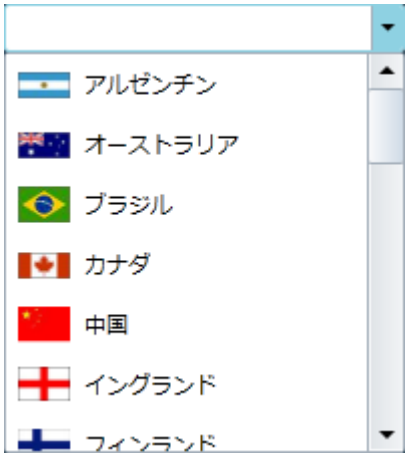
ComboBox for WPF/Silverlight を使用すると、機能豊富でカスタマイズされたアプリケーションを作成できます。次のような主要機能を利用して、**ComboBox for WPF/Silverlight** を最大限に活用してください。

- **自動検索ドロップダウンリスト**

  最初の数文字を入力すると、項目が即座に見つかります。ComboBox が自動的にリストを検索し、ユーザーが入力した項目が選択されます。

- **ドロップダウンリストへのデータテンプレートの設定**

  ComboBox は、データテンプレートを完全にサポートします。これにより、リスト項目に任意のビジュアル要素を簡単に追加できます。このビジュアル要素には、テキスト、画像、および他の任意のコントロールがあります。このコントロールは要素の仮想化を使用するため、数百の項目が挿入される場合でも、常にすばやくロードされます。



- **実績ある使い慣れたオブジェクトモデル**

  ComboBox には、WPF/Silverlight ComboBox コントロールに基づく機能豊富なオブジェクトモデルを持ちます。簡単に、エンドユーザーがドロップダウンリストにない項目を入力できるかどうかを指定したり、選択中の項目のインデックスを取得または設定したり、ドロップダウンリストの高さを指定することができます。

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

クイックスタート

このクイックスタートは、**ComboBox 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

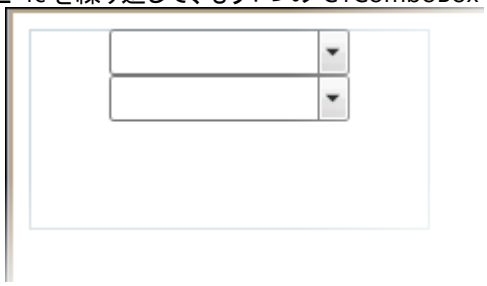
イックスタートでは、最初に Visual Studio で2つの **C1ComboBox** コントロールを含む新しいプロジェクトを作成します。最初のコントロールのリストには3つの項目を挿入します。これらの項目をクリックすることで、2つ目のコンボボックスに表示するリストが決定されます。

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 **ComboBox 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。

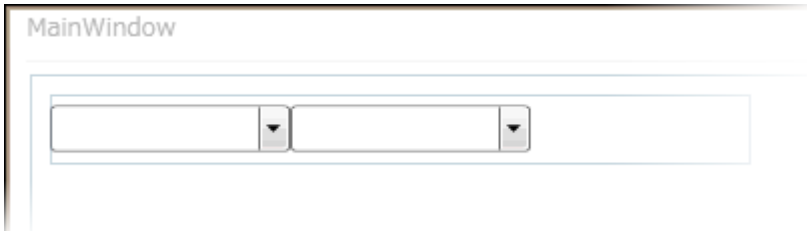
1. Visual Studio で、**[ファイル]**→**[新しいプロジェクト]**を選択します。
2. **[新しいプロジェクト]**ダイアログボックスの左ペインから言語を選択し、テンプレートリストから**[WPF アプリケーション]**または**[Silverlight アプリケーション]**を選択します。
3. プロジェクトの名前を入力し、**[OK]**をクリックします。
4. 参照の追加ダイアログボックスで、以下のアセンブリを見つけて選択し、[OK]をクリックしてプロジェクトに参照を追加します。
   - **WPF**: C1.WPF.4.dll
   - **Silverlight**: C1.Silverlight.5.dll
5. 次の手順に従って、プロジェクトに2つの C1ComboBox コントロールを追加します。
   a. ツールボックスで、StackPanel アイコンをダブルクリックして、このコントロールをプロジェクトに追加します。
   b. StackPanel コントロールを選択します。
   c. [C1ComboBox]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。
   d. 手順 4b と 4c を繰り返して、もう1つの C1ComboBox を StackPanel に追加します。プロジェクトは次のようになります。



6. **ツールボックス**で、**StackPanel** アイコンをダブルクリックして、このコントロールをプロジェクトに追加します。**StackPanel** コントロールのプロパティを次のように設定します。
   - **Width** プロパティを「**300**」に設定します。
   - **Height** プロパティを「**35**」に設定します。
   - **Orientation** プロパティを **Horizontal** に設定します。
7. **c1ComboBox**1 のプロパティを次のように設定します。
   - **Width** プロパティを「**150**」に設定します。
   - **Height** プロパティを「**35**」に設定します。
   - **Name** プロパティを「**Category**」に設定します。
8. **c1ComboBox2** のプロパティを次のように設定します。
   - **Width** プロパティを「**150**」に設定します。

- **Height** プロパティを「**35**」に設定します。
- **Name** プロパティを「**Fields**」に設定します。

プロジェクトは次のようになります。



WPF プロジェクトを作成し、それに2つの **C1ComboBox** コントロールを追加して、クイックスタートの最初の手順を完了しました。次の手順では、最初の **C1ComboBox** コントロールに項目を追加します。

手順 2:コントロールへの項目の追加

前の手順では、プロジェクトを作成し、2つの **C1ComboBox** コントロールを追加しました。この手順では、最初のコンボボックスに3つの項目を追加します。

次の手順に従います。

1. 最初の **C1ComboBox** の **Category** を選択します。
2. **[プロパティ]**ウィンドウで、**[Items]**の省略符ボタンをクリックして**[コレクションエディター:Items]**ダイアログボックスを開きます。
3. **[追加]**を3回クリックして、3つの **C1ComboBoxItem** をコントロールに追加します。**c1ComboBoxItem1**、**c1ComboBoxItem2**、**c1ComboBoxItem3** という名前の3つの **C1ComboBoxItem** がコントロールに追加されます。
4. **c1ComboBoxItem1** のプロパティを次のように設定します。
   - **Content** プロパティを「文学」に設定します。
   - **Height** プロパティを「25」に設定します。
5. **c1ComboBoxItem2** のプロパティを次のように設定します。
   - **Content** プロパティを「ノンフィクション」に設定します。
   - **Height** プロパティを「25」に設定します。
6. **C1ComboBoxItem3** のプロパティを次のように設定します。
   - **Content** プロパティを「ビジネス」に設定します。
   - **Height** プロパティを「25」に設定します。
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。

この手順では、最初のコンボボックスに項目を追加しました。次の手順では、最初のコンボボックスでユーザーがある項目を選択したときに、2つ目のコンボボックスに項目を設定するためのコードをプロジェクトに追加します。

手順 3:アプリケーションへのコードの追加

最後の手順では、最初のコンボボックスに項目を追加しました。この手順では、最初のコンボボックスでユーザーが選択したオプションに基づいて、2つ目のコンボボックスにデータを設定するためのコードをプロジェクトに追加します。

1. 最初の**C1ComboBox**("Category")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** テキストボックス内をダブルクリックして、**C1ComboBox1_SelectedIndexChanged** イベントハンドラを追加します。**MainPage.xaml.cs** ページが開きます。
4. 次の名前空間をプロジェクトにインポートします。

Visual Basic
```
Imports System.Collections.Generic
```

C#
```
using System.Collections.Generic;
```

5. **C1ComboBox1_SelectedIndexChanged** イベントハンドラに次のコードを追加します。

Visual Basic
```
'「文学」選択肢用リストを作成します
Dim dropDownList_Literature As New List(Of String)()
dropDownList_Literature.Add("歴史小説")
dropDownList_Literature.Add("社会小説")
dropDownList_Literature.Add("SF")
'「ノンフィクション」選択肢用リストを作成します
Dim dropDownList_NonFiction As New List(Of String)()
dropDownList_NonFiction.Add("社会")
dropDownList_NonFiction.Add("事件")
dropDownList_NonFiction.Add("エンターテインメント")
'「ビジネス」選択肢用リストを作成します
Dim dropDownList_Business As New List(Of String)()
dropDownList_Business.Add("経済学")
dropDownList_Business.Add("マーケティング")
dropDownList_Business.Add("マネジメント")
'SelectedIndex 値をチェックして、適切なリストを2つ目のコンボボックスに割り当てます
If Category.SelectedIndex = 0 Then
   Fields.ItemsSource = dropDownList_Literature
ElseIf Category.SelectedIndex = 1 Then
   Fields.ItemsSource = dropDownList_NonFiction
ElseIf Category.SelectedIndex = 2 Then
   Fields.ItemsSource = dropDownList_Business
End If
```

C#
```
//「文学」選択肢用リストを作成します
List<string> dropDownList_Literature = new List<string>();
dropDownList_Literature.Add("歴史小説");
dropDownList_Literature.Add("社会小説");
dropDownList_Literature.Add("SF");
```

```
//「ノンフィクション」選択肢用リストを作成します
List<string> dropDownList_NonFiction = new List<string>();
dropDownList_NonFiction.Add("社会");
dropDownList_NonFiction.Add("事件");
dropDownList_NonFiction.Add("エンターテインメント");
//「ビジネス」選択肢用リストを作成します
List<string> dropDownList_Business = new List<string>();
dropDownList_Business.Add("経済学");
dropDownList_Business.Add("マーケティング");
dropDownList_Business.Add("マネジメント");
//SelectedIndex 値をチェックして、適切なリストを2つ目のコンボボックスに割り当てます
if (Category.SelectedIndex == 0)
{
     Fields.ItemsSource = dropDownList_Literature;
}
else if (Category.SelectedIndex == 1)
{
     Fields.ItemsSource = dropDownList_NonFiction;
}
else if (Category.SelectedIndex ==2)
{
     Fields.ItemsSource = dropDownList_Business;
}
```

次の手順では、プロジェクトを実行し、このクイックスタートの結果を確認します。

手順 4:アプリケーションの実行

前の3つの手順では、2つのコンボボックスを含む WPF プロジェクトを作成し、最初のコンボボックスに項目を追加し、最初のコンボボックスの項目が選択されたときに、2つ目のコンボボックスに項目を設定するためのコードを作成しました。この手順では、プロジェクトを実行し、このクイックスタートの結果を確認します。

次の手順に従います。

1. [F5]キーを押してプロジェクトを実行します。2つの空のコンボボックスを含むプロジェクトがロードされます。

2. 2つ目のコンボボックスのドロップダウン矢印をクリックし、ドロップダウンリストが空であることを確認します。

3. 最初のコンボボックスのドロップダウン矢印をクリックし、**[文学]**を選択します。

4. 2つ目のコンボボックスのドロップダウン矢印をクリックし、ドロップダウンリストに以下の項目が含まれていることを確認します。

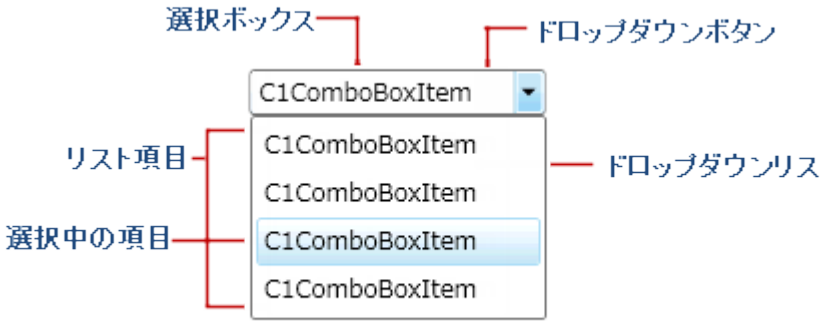
5. 最初のコンボボックスのドロップダウン矢印をクリックし、**[ノンフィクション]**を選択します。
6. 2つ目のコンボボックスのドロップダウン矢印をクリックし、ドロップダウンリストに以下の項目が含まれていることを確認します。

7. 最初のコンボボックスのドロップダウン矢印をクリックし、**[ビジネス]**を選択します。
8. 2つ目のコンボボックスのドロップダウン矢印をクリックし、ドロップダウンリストに以下の項目が含まれていることを確認します。

おめでとうございます。これで、**ComboBox for WPF/Silverlight** クイックスタートは終了です。

C1ComboBox の要素

C1ComboBox コントロールは、ドロップダウンリストにデータを表示するために使用される柔軟なコントロールです。基本的に、これは、選択項目を入力するためのテキストボックスと、リストからオプションを選択するためのリストボックスを組み合わせたコントロールです。次の図に、**C1ComboBox** コントロールを示します。



C1ComboBox の各要素については、以下を参照してください。

- **選択ボックス**

  選択ボックスには、ユーザーが検索対象のリスト項目をテキストボックスに直接入力できるようにする機能と、現在選択されている項目を表示する機能の2つの機能があります。このボックスのコンテンツは、**C1ComboBox** コントロールの選択中のインデックス項目のコンテンツと同じです。

- **ドロップダウンボタン**

  ドロップダウンボタンをクリックすると、ドロップダウンリストが表示されます。

- **ドロップダウンリスト**

ドロップダウンリストは、一連のリスト項目で構成されます（以下を参照）。ここには、必要な数のリスト項目を表示できます。項目の数がドロップダウンリストのサイズを超えた場合は、自動的にスクロールバーが表示されます。

- **リスト項目**

  ドロップダウンリストの各リスト項目は、**C1ComboBoxItem** クラスによって表されます。リスト項目は、テキスト、ピクチャのほか、コントロールにすることもできます。

- **選択中の項目**

  リスト内で選択中の項目は、開発者が固定してしまうことも、実行時にユーザーが選択することもできます。選択中のリスト項目の **IsSelected** プロパティは、**True** になります。

C1ComboBox の機能

コンボボックス項目

コンボボックス項目を追加する

C1ComboBox コントロールは、次に示すいくつかの方法で追加できます。

XAMLの場合

1. **C1ComboBox** コントロールに項目を追加するには、**<c1:C1ComboBox>** タグと **</c1:C1ComboBox>** タグの間に次の XAML マークアップを追加します。

   マークアップ

   ```
   <c1:C1ComboBoxItem Height="25" Content="C1ComboBoxItem"/>
   <c1:C1ComboBoxItem Height="25" Content="C1ComboBoxItem"/>
   <c1:C1ComboBoxItem Height="25" Content="C1ComboBoxItem"/>
   <c1:C1ComboBoxItem Height="25" Content="C1ComboBoxItem"/>
   ```

2. プログラムを実行します。
3. ドロップダウン矢印をクリックし、ドロップダウンリストに4つの項目が表示されることを確認します。結果は次の図のようになります。

コードの場合- WPF

1. **x:Name="C1ComboBoxItem1"** を **<c1:C1ComboBoxItem>** タグに追加します。これで、このオブジェクト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.cs または MainWindow.xaml.cs ページを開きます。
3. 次の名前空間をプロジェクトにインポートします。

   Visual Basic

   ```
   Imports C1.WPF
   ```

   C#

   ```
   Using C1.WPF;
   ```

4. 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

```
C1ComboBox1.Items.Add(New C1ComboBoxItem() With {.Content = "C1ComboBoxItem1"})
C1ComboBox1.Items.Add(New C1ComboBoxItem() With {.Content = "C1ComboBoxItem2"})
C1ComboBox1.Items.Add(New C1ComboBoxItem() With {.Content = "C1ComboBoxItem3"})
C1ComboBox1.Items.Add(New C1ComboBoxItem() With {.Content = "C1ComboBoxItem4"})
```

C#

```
c1ComboBox1.Items.Add(new C1ComboBoxItem() { Content = "C1ComboBoxItem1" });
c1ComboBox1.Items.Add(new C1ComboBoxItem() { Content = "C1ComboBoxItem2" });
c1ComboBox1.Items.Add(new C1ComboBoxItem() { Content = "C1ComboBoxItem3" });
c1ComboBox1.Items.Add(new C1ComboBoxItem() { Content = "C1ComboBoxItem4" });
```

5. プログラムを実行します。

コードの場合- Silverlight

1. **x:Name="C1ComboBoxItem1"** を **<c1:C1ComboBoxItem>** タグに追加します。これで、このオブジェクト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.cs または MainPage.xaml.vb ページを開きます。
3. 次の名前空間をプロジェクトにインポートします。

Visual Basic

```
Imports C1.Silverlight
```

C#

```
Using C1.Silverlight;
```

4. 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

```
C1ComboBox1.Items.Add(New C1ComboBoxItem() With {.Content = "C1ComboBoxItem1"})
C1ComboBox1.Items.Add(New C1ComboBoxItem() With {.Content = "C1ComboBoxItem2"})
C1ComboBox1.Items.Add(New C1ComboBoxItem() With {.Content = "C1ComboBoxItem3"})
C1ComboBox1.Items.Add(New C1ComboBoxItem() With {.Content = "C1ComboBoxItem4"})
```

C#

```
C1ComboBox1.Items.Add(new C1ComboBoxItem() { Content = "C1ComboBoxItem1" });
C1ComboBox1.Items.Add(new C1ComboBoxItem() { Content = "C1ComboBoxItem2" });
C1ComboBox1.Items.Add(new C1ComboBoxItem() { Content = "C1ComboBoxItem3" });
C1ComboBox1.Items.Add(new C1ComboBoxItem() { Content = "C1ComboBoxItem4" });
```

5. プログラムを実行します。

コレクションの場合

1. x:Name="C1ComboBoxItem1" を <c1:C1ComboBoxItem> タグに追加します。これで、このオブジェクト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.cs** または **MainPage.xaml.cs** ページを開きます。
3. 次の名前空間をプロジェクトにインポートします。

Visual Basic
```
Imports System.Collections.Generic
```

C#
```
using System.Collections.Generic;
```

4. **InitializeComponent()** メソッドの下に次のコードを追加して、リストを作成します。

Visual Basic
```
Dim dropDownList As New List(Of String)()
dropDownList.Add("C1ComboBoxItem1")
dropDownList.Add("C1ComboBoxItem2")
dropDownList.Add("C1ComboBoxItem3")
dropDownList.Add("C1ComboBoxItem4")
```

C#
```
List<string> dropDownList = new List<string>();
dropDownList.Add("C1ComboBoxItem1");
dropDownList.Add("C1ComboBoxItem2");
dropDownList.Add("C1ComboBoxItem3");
dropDownList.Add("C1ComboBoxItem4");
```

5. **ItemsSource** プロパティを設定して、コンボボックスにリストを追加します。

Visual Basic
```
C1ComboBox1.ItemsSource = dropDownList
```

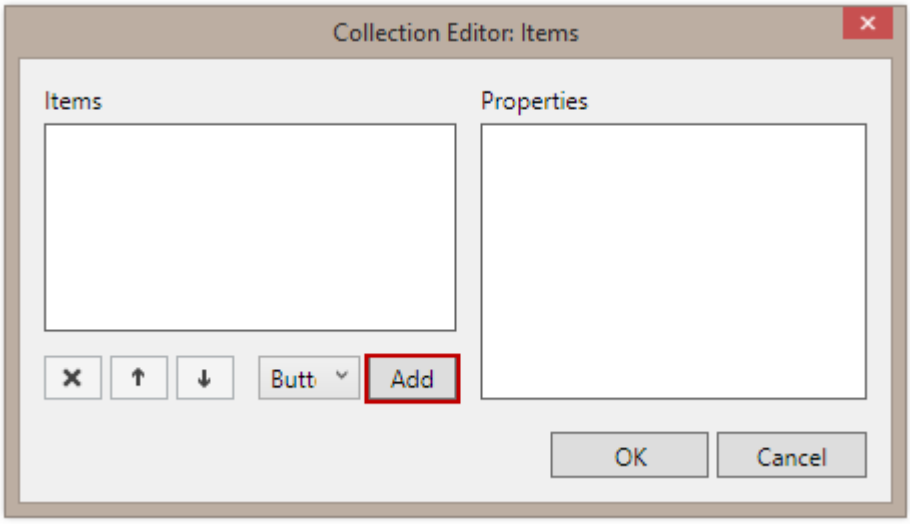
C#
```
c1ComboBox1.ItemsSource = dropDownList;
```

6. プログラムを実行します。

デザイナの場合

次の手順に従います。

1. **[プロパティ]**ウィンドウで、**[Items(コレクション)]**の省略符ボタン()をクリックして**[コレクションエディター: Items]**ダイアログボックスを開きます。

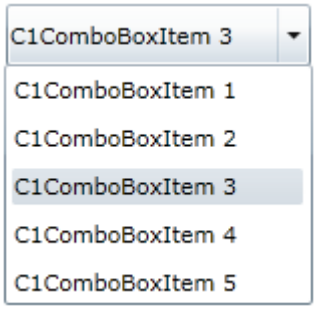


2. **[別のアイテムを追加]**のドロップダウンをクリックして、**C1ComboBoxItem** を **C1ComboBox** コントロールに追加します。

項目の選択

SelectedIndexプロパティは現在選択されているインデックスを取得または設定します。The **SelectedIndex** is based on a zero-based index, meaning that 0 represents the first **C1ComboBoxItem**, 1 represents the second **C1ComboBoxItem**, and so on.

In the image below, the **SelectedIndex** is set to **2**, which selects the third **C1ComboBoxItem**.



XAMLの場合

選択中の項目を設定するには、**SelectedIndex="0"** を **<c1:C1ComboBoxItem>** タグに追加します。マークアップは、次のようになります。

マークアップ
`<c1:C1ComboBoxItem Content="C1ComboBoxItem1" SelectedIndex="1">`

コードの場合

次の手順に従います。

1. **MainPage.xaml.cs** または **MainPage.xaml.cs** ページを開きます。Visual Basicの場合は、**MainWindow.xaml.vb** ページを開きます。
2. **InitializeComponent()** メソッドの下に次のコードを追加します。

Visual Basic
`C1ComboBoxItem1.SelectedIndex = 1`

C#
`c1ComboBoxItem1.SelectedIndex = 1;`

3. プログラムを実行します。

デザイナの場合

次の手順に従います。

1. **C1ComboBox** コントロールを選択します。
2. **[プロパティ]**ウィンドウで、**SelectedIndex** プロパティを「**1**」に設定して、2番目の **C1ComboBoxItem** が選択されるようにします。

ドロップダウンリスト

ドロップダウンリストの方向の変更

デフォルトでは、ユーザーが実行時に **C1ComboBox** 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ドロップダウンリストが表示されます。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** プロパティを以下の4つのオプションの1つに設定して、ドロップダウンリストの表示方向を変更することもできます。

イベント	説明
BelowOrAbove (デフォルト)	ドロップダウンボックス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	ドロップダウンボックス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	ドロップダウンボックスをヘッダーの下に強制的に表示します。
ForceAbove	ドロップダウンボックスをヘッダーの上に強制的に表示します。

ドロップダウンリストの方向は、次のいずれかの方法で変更できます。

XAML の場合

次の手順に従います。

1. **DropDownDirection="ForceAbove"** を **<c1:C1ComboBox>** タグに追加します。マークアップは、次のようになります。<c1:C1ComboBox Width="249" DropDownDirection="ForceAbove">
2. プログラムを実行し、ドロップダウン矢印をクリックします。ドロップダウンリストがコントロールの上に表示されることを確認します。

コードの場合

次の手順に従います。

1. x:Name="C1ComboBox1" を <c1:C1ComboBox> タグに追加します。これで、このオブジェクト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.cs** または **MainPage.xaml.vb** ページを開きます。
3. **InitializeComponent()** メソッドの下に次のコードを追加します。

Visual Basic
```
C1ComboBox1.DropDownDirection = DropDownDirection.ForceAbove
```

C#
```
C1ComboBox1.DropDownDirection = DropDownDirection.ForceAbove;
```

4. プログラムを実行し、ドロップダウン矢印をクリックします。ドロップダウンリストがコントロールの上に表示されることを確認します。

デザイナの場合

次の手順に従います。

1. **C1ComboBox** 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、**DropDownDirection** のドロップダウン矢印をクリックし、オプションを選択します。この例では、[**ForceAbove**]を選択します。
3. プログラムを実行し、ドロップダウン矢印をクリックします。ドロップダウンリストがコントロールの上に表示されることを確認します。

ドロップダウンリストの最大高さと最大幅の設定

デフォルトでは、**DropDownWidth** プロパティと **DropDownHeight** プロパティはどちらも **NaN** に設定されているため、ドロップダウンリストのサイズは、最も広い **C1ComboBoxItem** 項目の幅とすべての **C1ComboBoxItem** 項目の高さの合計によって決定されます。

ドロップダウンリストの最大幅と最大高さを制御するには、**C1ComboBox** コントロールの **MaxDropDownWidth** プロパティと **MaxDropDownHeight** プロパティを設定します。これらのプロパティを設定すると、ドロップダウンリストの領域は、指定された値より大きくなりません。リストの幅または高さが、指定された最大の幅または高さを超えている場合は、ドロップダウンリストにスクロールバーが自動的に追加されます。

XAMLの場合

次の手順に従います。

1. MaxDropDownHeight="150" と MaxDropDownWidth="350" を <c1:C1ComboBox> タグに追加します。マークアップは、次のようになります。 <c1:C1ComboBox HorizontalAlignment="Left" Width="249" MaxDropDownHeight="150" MaxDropDownWidth="350">
2. プログラムを実行し、コンボボックスのドロップダウン矢印をクリックして、設定の結果を確認します。

コードの場合

次の手順に従います。

1. **x:Name="C1ComboBox1"** を **<c1:C1ComboBox>** タグに追加します。これで、このオブジェクト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.cs ページを開きます。
3. 次のコードを **InitializeComponent()** メソッドの下に追加して、**DropDownHeight** プロパティを設定します。

Visual Basic
```
C1ComboBox1.MaxDropDownHeight = 150
```

C#
```
c1ComboBox1.MaxDropDownHeight = 150;
```

4. 次のコードを **InitializeComponent()** メソッドの下に追加して、**DropDownWidth** プロパティを設定します。

Visual Basic
```
C1ComboBox1.MaxDropDownWidth = 350
```

C#
```
c1ComboBox1.MaxDropDownWidth = 350;
```

5. プログラムを実行し、コンボボックスのドロップダウン矢印をクリックして、設定の結果を確認します。

デザイナの場合

次の手順に従います。

1. **C1ComboBox** 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、次の手順に従います。
   - **MaxDropDownHeight** に値(「150」など)を設定します。
   - **MaxDropDownWidth** に値(「350」など)を設定します。
3. プログラムを実行し、コンボボックスのドロップダウン矢印をクリックして、設定の結果を確認します。

ドロップダウンリストを開いたまま起動

ドロップダウンリストを開いたまま **C1ComboBox** を起動するには、**IsDropDownOpen** プロパティを **True** に設定します。

XAMLの場合

次の手順に従います。

1. **IsDropDownOpen="True"** を **<c1:C1ComboBox>** タグに追加します。マークアップは、次のようになります。

マークアップ
```
<c1:C1ComboBox HorizontalAlignment="Left" Width="249" IsDropDownOpen="True">
```

2. プログラムを実行し、ページがロードされたときにドロップダウンリストが開いていることを確認します。

コードの場合

次の手順に従います。

1. x:Name="C1ComboBox1" を <c1:C1ComboBox> タグに追加します。これで、このオブジェクト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.cs** または **MainWindow.xaml.cs** ページを開きます。
3. **InitializeComponent()** メソッドの下に次のコードを追加します。

Visual Basic

```
C1ComboBox1.IsDropDownOpen = True
```

C#

```
c1ComboBox1.IsDropDownOpen = true;
```

4. プログラムを実行し、ページがロードされたときにドロップダウンリストが開いていることを確認します。

デザイナの場合

次の手順に従います。

1. **C1ComboBox** 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、[**IsDropDownOpen**]チェックボックスをオンにします。
3. プログラムを実行し、ページがロードされたときにドロップダウンリストが開いていることを確認します。

マウスオーバー時にドロップダウンリストを開く

デフォルトでは、ユーザーがドロップダウン矢印をクリックしたときにのみ、**C1ComboBox** コントロールのドロップダウンリストが表示されます。このトピックでは、ユーザーがコントロールの上にポインタを置いたときにドロップダウンリストを開くためのコードを作成します。

次の手順に従います。

1. **デザイナ上**に、**x:Name="C1ComboBox1"** を **<c1:C1ComboBox>** タグに追加します。これで、このオブジェクト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. **C1ComboBox** 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、[**イベント**]ボタン()をクリックして、コントロールのイベントリストを表示します。
4. **MouseEnter** テキストボックスの内部をダブルクリックします。これにより、コードビューに **C1ComboBox_MouseEnter** イベントハンドラが追加されます。
5. **C1ComboBox1_MouseEnter** イベントハンドラに次のコードを追加します。

Visual Basic

```
C1ComboBox1.IsDropDownOpen = True
```

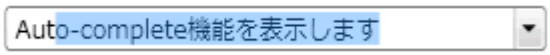
C#

```
c1ComboBox1.IsDropDownOpen = true;
```

6. プログラムを実行した状態で、**C1ComboBox** コントロールの上にカーソルポインタを置きます。コントロールの上にポインタを置いたときにドロップダウンリストが表示されることを確認します。項目を選択するか、コントロールの外側をクリックするまで、ドロップダウンリストは開いたままになります。

オートコンプリート

C1ComboBox コントロールは、ユーザー入力に基づいてリスト項目を選択する自動入力補完機能を備えています。ユーザーが入力を始めると、選択ボックスにリスト項目がロードされます。次の図に例を示します。



ユーザーは、[Enter]キーを押すだけで、**オートコンプリート**機能によって提示されたリスト項目を選択できます。

オートコンプリート機能は、**AutoComplete** プロパティを **False** に設定することによって無効にできます。

XAMLの場合

AutoComplete を無効にするには、AutoComplete="False" を <c1:C1ComboBox> タグに追加します。マークアップは、次のようになります。

マークアップ

```
<c1:C1ComboBox HorizontalAlignment="Left" Width="249" AutoComplete="False">
```

コードの場合

次の手順に従います。

1. x:Name="C1ComboBox1" を <c1:C1ComboBox> タグに追加します。これで、このオブジェクト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.cs** または **MainWindow.xaml.cs** ページを開きます。
3. **InitializeComponent()** メソッドの下に次のコードを追加します。

Visual Basic

```
C1ComboBox1.AutoComplete = False
```

C#

```
c1ComboBox1.AutoComplete = false;
```

4. プログラムを実行します。

デザイナの場合

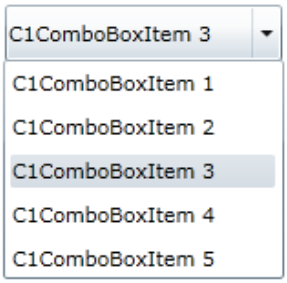
次の手順に従います。

1. **C1ComboBox** 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、[**AutoComplete**]チェックボックスをオフにします。

外観プロパティ

テーマ

ComboBox for WPF/Silverlight 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。**C1ComboBox** 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。



WPFのテーマ

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。

テーマの追加

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。

Visual Basic

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

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
   {
      C1ThemeExpressionDark theme = new C1ThemeExpressionDark();
      //Using ApplyTheme
      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。

Visual Basic

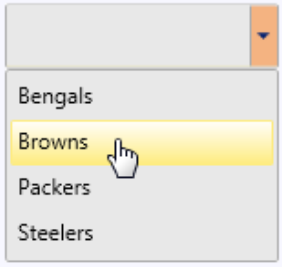
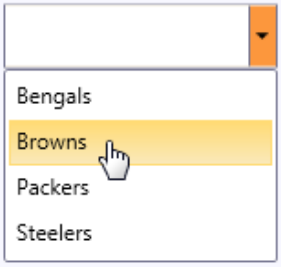
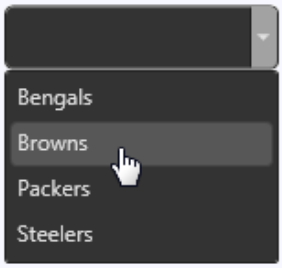
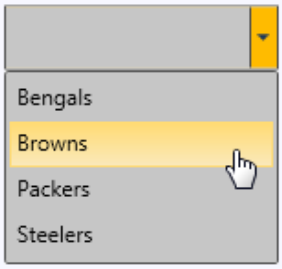
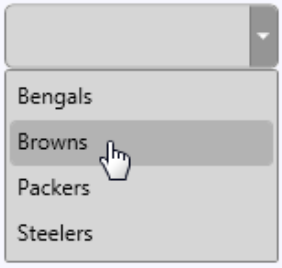
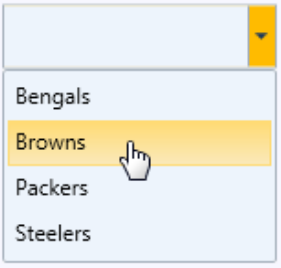
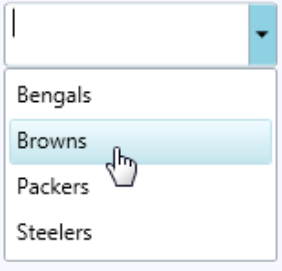
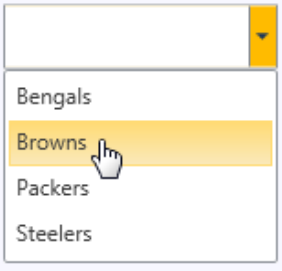
```
Private Sub Window_Loaded(sender As System.Object, e As System.Windows.RoutedEventArgs)
Handles MyBase.Loaded
   Dim theme As New C1ThemeExpressionDark
   'Using Merged Dictionaries
```

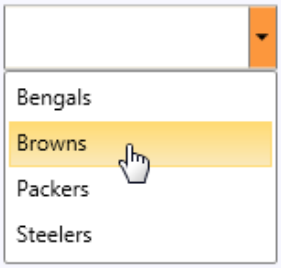
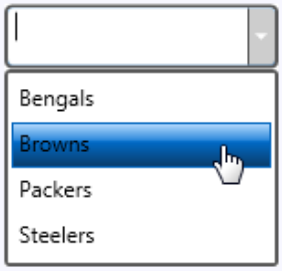
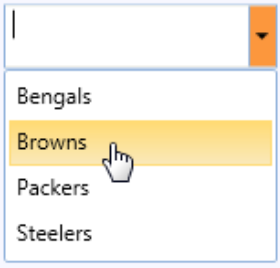
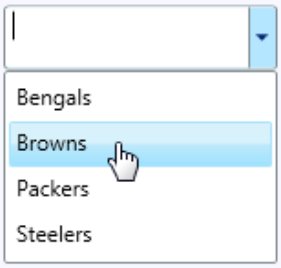
```
Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThemeResources(theme))
End Sub
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
   {
      C1ThemeExpressionDark theme = new C1ThemeExpressionDark();
      //Using Merged Dictionaries

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

組み込みテーマ

テーマ名	テーマのプレビュー	テーマ名	テーマのプレビュー
C1ThemeBureauBlack		C1ThemeOffice2007Silver	
C1ThemeExpressionDark		C1ThemeOffice2010Black	
C1ThemeExpressionLight		C1ThemeOffice2010Blue	
C1Blue		C1ThemeOffice2010Silver	

C1ThemeOffice2007Black		C1ThemeShinyBlue	
C1ThemeOffice2007Blue		C1ThemeWhistlerBlue	

Silverlightのテーマ

C1ComboBox コントロールには、付属している Silverlight のテーマの中から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。

テーマの追加

いずれかのテーマを **C1ComboBox** コントロールに追加するには、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。テーマはVisual Studioのツールボックスにアイコンとして表示されます。

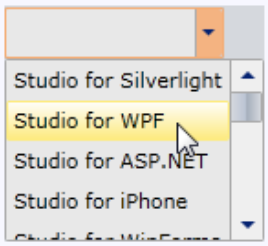
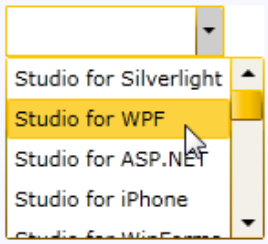
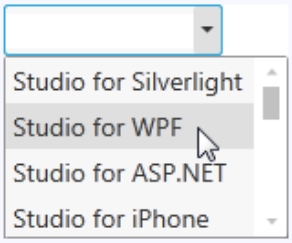
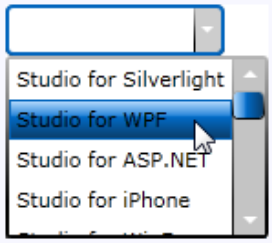
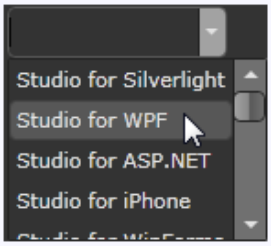
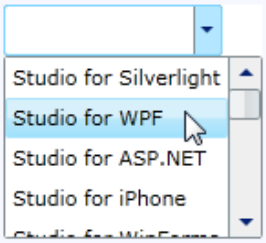
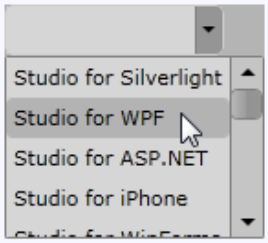
1. Visual Studio で、.xaml ページを開きます。
2. **<Grid></Grid>**タグの間にカーソルを置きます。
3. [**ツール**]パネルで、[**C1ThemeRainierOrange**]アイコンをダブルクリックしてテーマを宣言します。このタグは次のようになります。<my:C1ThemeRainierOrange></my:C1ThemeRainierOrange>

   マークアップ
   ```
   <my:C1ThemeRainierOrange></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、[**C1ComboBox**]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 **<my:C1ThemeRainierOrange>** 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。

   マークアップ
   ```
   <my:C1ThemeRainierOrange>
       <c1:C1ComboBox></c1:C1ComboBox>
   </my:C1ThemeRainierOrange>
   ```
6. プロジェクトを実行します。

組み込みテーマ

テーマ名	テーマのプレビュー	テーマ名	テーマのプレビュー

C1ThemeBureauBlack		C1ThemeRanierOrange	
C1ThemeCosmopolitan		C1ThemeShinyBlue	
C1ThemeExpressionDark		C1ThemeWhistlerBlue	
C1ThemeExpressionLight			

テンプレート

WPF

WPF 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、**ComboBox for WPF** 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。

Microsoft Expression Blend でテンプレートをアクセスするすることができます。

1. **C1ComboBox** コントロールを選択します。
2. メニューから[**テンプレートの編集**]を選択します。
3. [**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。

C1ComboBoxItem テンプレートを編集する場合は

1. **C1ComboBoxItem** を選択します。
2. メニューから[**テンプレートの編集**]を選択します。

3. [**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。

デフォルトテンプレートのほかに、**C1ComboBox** 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。

1. Blend で **C1ComboBox** コントロールを選択します。
2. メニューから[**追加テンプレートの編集**]を選択します。
3. テンプレートを選択し、[**空アイテムの作成**]を選択します。

Silverlight

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、**ComboBox 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。

テンプレートにアクセスするには、Microsoft Expression Blend で、**C1ComboBox** 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。

C1ComboBoxItem テンプレートを編集する場合は、**C1ComboBoxItem** 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。

DragDrop Manager

DragDropManager for WPF/Silverlight を使用すると、どこにでもドラッグアンドドロップ操作を簡単に追加できます。**C1DragDropManager** クラスは、ドラッグ元やドロップ先のカスタマイズを含む、見栄えのよいドラッグアンドドロップ UI を提供します。

DragDropManager for WPF/Silverlight には、次の主要な機能があります。

- **ドラッグアンドドロップ動作の制御**

  C1DragDropManager には、ドラッグアンドドロップ処理全体を制御できるさまざまなメソッドやイベントが用意されています。いくつかの要素をドラッグ元およびドロップ先として登録してから、要素を新しい場所に移動またはコピーするために DragDrop イベントを処理します。

- **ドラッグマーカーのカスタマイズ**

  C1DragDropManager は、ドラッグ元とドロップ先のドラッグマーカーの外観をカスタマイズするためのプロパティを公開します。これにより、操作全体が見栄えよく、ユーザーフレンドリになります。

- **スクロールのサポート**

  スクロール可能なターゲットの端の近くにオブジェクトをドラッグすると、ターゲットが自動的にスクロールされるので、エンドユーザーは、要素を1回の操作で目的の場所にドロップできます。

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

クイックスタート

手順 1:アプリケーションの作成

この手順では、Visual Studio で、**DragDropManager for WPF/Silverlight** を使用してユーザー操作を管理する WPF または Silverlight アプリケーションを作成します。

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**]→[**新規作成**]→[**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(たとえば、「QuickStart」)を入力し、[OK]をクリックします。[新しい WPF 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 WPF/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. ソリューションエクスプローラウィンドウでプロジェクトを右クリックし、[参照の追加]を選択します。
5. 参照の追加ダイアログボックスで、以下のアセンブリを見つけて選択し、[OK]をクリックしてプロジェクトに参照を追加します。
   - **WPF**: C1.WPF.4.dll
   - **Silverlight**: C1.Silverlight.5.dll
6. 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。
7. 次の XAML マークアップを MainPage.xaml ファイルの <Grid> タグと </Grid> タグの間に追加します。

XAML

```
Grid x:Name="ddGrid" Background="Lavender" ShowGridLines="True" Width="400"
Height="300" >
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <TextBlock Text="ドラッグで" FontSize="14" Grid.Row="1" Grid.Column="2" />
    <TextBlock Text="移動できます" FontSize="14" Grid.Row="3" Grid.Column="4" />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0"/>
    <Rectangle Fill="Blue" Grid.Row="0" Grid.Column="4"/>
</Grid>
```

このマークアップは、**TextBlock** コントロールや **Rectangle** コントロールをそれぞれ別の領域に置けるように、いくつかの行定義を含むグリッドを作成します。

ここまでの成果

WPF/Silverlight アプリケーションを正しく作成および設定し、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：アプリケーションへのコードの追加

前の手順では、アプリケーションを設定しましたが、アプリケーションにはまだドラッグアンドドロップ機能を追加していません。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** or **MainWindow.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 文をページの先頭に追加します。

Visual Basic

```
Imports C1.WPF
```

C#

```
using C1.WPF;
```

3. Page コンストラクタで、**MainPage.xaml.cs**（または **.vb**）ファイルまたは MainWindow.xaml.cs (or .vb) ファイルにコードを追加します。次のようになります。

Visual Basic

```
Public Sub New()
    InitializeComponent()
    ' C1DragDropManager を初期化します
    Dim dd As New C1DragDropManager()
    dd.RegisterDropTarget(ddGrid, True)
    For Each e As UIElement In ddGrid.Children
        dd.RegisterDragSource(e, DragDropEffect.Move, ModifierKeys.None)
    Next
    AddHandler dd.DragDrop, AddressOf dd_DragDrop
End Sub
```

C#

```
public MainPage()
{
  InitializeComponent();
  // C1DragDropManager を初期化します
  C1DragDropManager dd = new C1DragDropManager();
  dd.RegisterDropTarget(ddGrid, true);
  foreach (UIElement e in ddGrid.Children)
  {
    dd.RegisterDragSource(e, DragDropEffect.Move, ModifierKeys.None);
  }
  dd.DragDrop += dd_DragDrop;
}
```

このコードは、**C1DragDropManager** の新しいインスタンスを初期化してから、**RegisterDropTarget** メソッドを呼び出して、デフォルトでグリッドがドロップ先として機能するように指定します。次に、**RegisterDragSource** メソッドを呼び出して、ユーザーがグリッド内の要素をドラッグできるように指定します。最後に、**DragDrop** イベントにイベントハンドラをアタッチします。これで、アプリケーションは通知を受け取り、ドラッグされた要素を新しい位置に移動できるようになります。

4. **MainPage.xaml.cs**（または **.vb**）ファイル または MainWindow.xaml.cs (or .vb) fileで、**MainPage** クラスの他のすべてのメソッドより下に、次のイベントハンドラを追加します。

Visual Basic

```
Private Sub dd_DragDrop(source As Object, e As DragDropEventArgs)
    ' マウスの位置を取得します
    Dim pMouse As Point = e.GetPosition(ddGrid)
    ' グリッド行/列座標に変換します
    Dim row As Integer, col As Integer
    Dim pGrid As New Point(0, 0)
    For row = 0 To ddGrid.RowDefinitions.Count - 1
        pGrid.Y += ddGrid.RowDefinitions(row).ActualHeight
        If pGrid.Y > pMouse.Y Then
        Exit For
        End If
    Next
    For col = 0 To ddGrid.ColumnDefinitions.Count - 1
       pGrid.X += ddGrid.ColumnDefinitions(col).ActualWidth
        If pGrid.X > pMouse.X Then
        Exit For
        End If
    Next
    ' 要素を新しい位置に移動します
    e.DragSource.SetValue(Grid.RowProperty, row)
    e.DragSource.SetValue(Grid.ColumnProperty, col)
End Sub
```

C#

```
private void dd_DragDrop(object source, DragDropEventArgs e)
{
  // マウスの位置を取得します
  Point pMouse = e.GetPosition(ddGrid);
  // グリッド行/列座標に変換します
  int row, col;
  Point pGrid = new Point(0, 0);
  for (row = 0; row < ddGrid.RowDefinitions.Count; row++)
    {
      pGrid.Y += ddGrid.RowDefinitions[row].ActualHeight;
      if (pGrid.Y > pMouse.Y)
      break;
    }
  for (col = 0; col < ddGrid.ColumnDefinitions.Count; col++)
  {
    pGrid.X += ddGrid.ColumnDefinitions[col].ActualWidth;
    if (pGrid.X > pMouse.X)
```

```
    break;
  }
 // 要素を新しい位置に移動します
  e.DragSource.SetValue(Grid.RowProperty, row);
  e.DragSource.SetValue(Grid.ColumnProperty, col);
}
```

このイベントハンドラは、最初にマウスの座標を行/列座標に変換します。次に **SetValue** メソッドを使用して、ドラッグされた要素の **Grid.RowProperty** および **Grid.ColumnProperty** 値を更新します。同様のロジックを使用して、他のタイプのパネルやリストタイプのコントロール内の要素をドラッグしたり、1つのパネルから別のパネルに要素をドラッグすることができます。

ここまでの成果

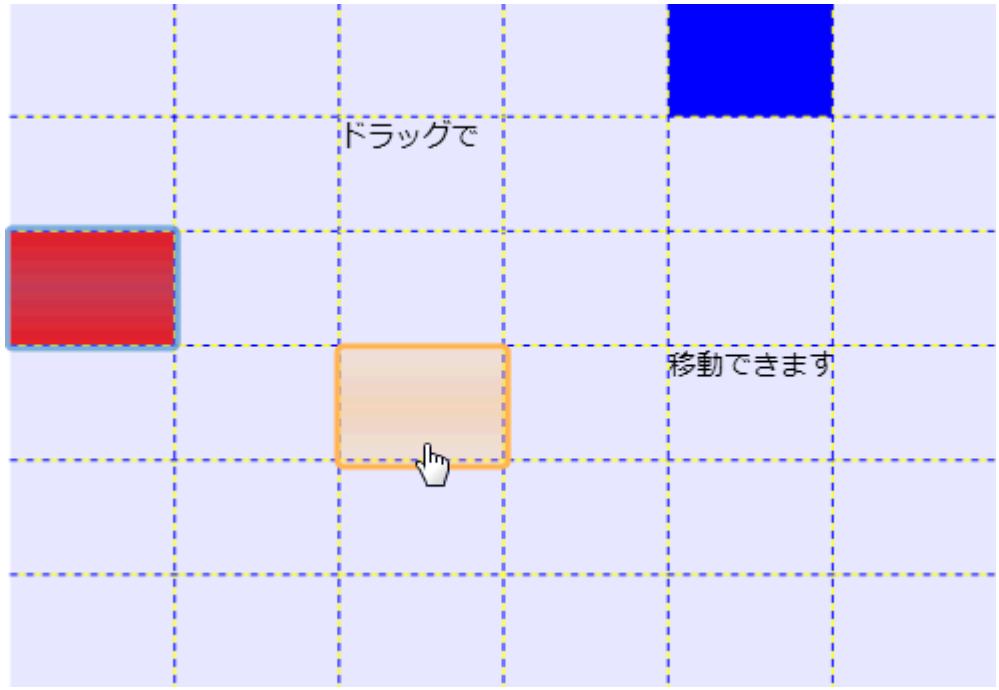
この手順では、機能を追加するコードをアプリケーションに追加しました。次の手順では、アプリケーションを実行し、**DragDropManager for WPF/Silverlight** の実行時の操作をいくつか確認します。

手順 3：アプリケーションの実行

WPF/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。

2. **赤色の四角形**をクリックし、それをグリッド内の別のます目にドラッグします。ドラッグ処理が行われると、ドラッグしている項目を囲む特別な境界線が表示され、透明な四角形はマウスに追随して、要素がドロップされる場所を示します。



3. **TextBlock** などの別の項目をクリックし、新しい場所に移動します。

ここまでの成果

おめでとうございます!

これで **DragDropManager for WPF/Silverlight** のクイックスタートは終了です。**DragDropManager for WPF/Silverlight** を使用してグリッド内の項目を移動する単純なアプリケーションを作成しました。

Drop Down

DropDown for WPF/Silverlight では、セットアップが簡単で、使用する際は強力な単一の項目を選択する機能を実装できます。**DropDown for WPF/Silverlight** は、**ComboBox**、**DateTimePicker** などのシンプルなドロップダウンボックスコントロールを提供します。**C1DropDown** は、従来のドロップダウンボックスよりも柔軟で、コントロールを追加でき、検索ボックスを作成することもできます。

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

- **3つのドロップダウンコントロール**

  DropDown for WPF/Silverlight には、アプリケーションの柔軟性を高める3つのドロップダウンコントロールが含まれます。**C1DropDown** は、従来のドロップダウンコントロールと同様に、一連の項目から選択を行うことができます。**C1DropDownButton** の機能はドロップダウンコントロールと同じですが、外観はボタンです。**C1SplitButton** には、カスタマイズ可能なヘッダーと矢印の2つの部分があります。この矢印をクリックするとドロップダウンリストが開き、ヘッダーボタンを押すと、通常は現在の選択が適用されます。

- **専用のドロップダウンエディタの作成**

  C1DropDownコントロールを使用すると、専用のドロップダウンエディタの作成を全面的に制御できます。スピンボタンに独自のロジックをアタッチしたり、ドロップダウンボタンに独自のドロップダウンフォーム/エディタをアタッチすることができます。たとえば、ドロップダウン部分に **DataGrid** を配置し、それに対する行選択イベントのコードを記述することにより、その行からの値を **C1DropDown** のヘッダー部分に表示することもできます。

- **ヘッダーの上または下への展開**

  DropDownDirection プロパティを使用して、ヘッダー部分の上または下に表示するようにドロップダウンを設定しま

す。フォーム上にスペースがある場合に、コントロールが使用する方向の優先順位を設定します。

- **AutoClose**

  AutoClose プロパティを使用して、ドロップダウンを閉じるかどうか、およびいつ閉じるかをすべて制御します。

- **サイズ変更可能ドロップダウンリスト**

  ドロップダウンボックスの幅と高さは、明示的に設定するか、またはコンテンツに合わせて自動的にサイズ変更できます。

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

クイックスタート

手順 1:アプリケーションの作成

この手順では、**DropDown for WPF/Silverlight** を使って WPF/Silverlight アプリケーションを作成します。**C1DropDown** コントロールをアプリケーションに追加すると、完全に機能するドロップダウンボックス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、C1DropDown コントロールをアプリケーションに追加するには、次の手順に従います。

1. Visual Studio で新しい WPF/Silverlight プロジェクトを作成します。
2. 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以下のアセンブリを選択して[OK]をクリックし、参照をプロジェクトに追加します。
   - **WPF:** C1.WPF.4.dll
   - **Silverlight:** C1.Silverlight.5.dll
3. Visual Studio のツールボックスに移動し、[**C1DropDown**]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 C1DropDown コントロールを再配置します。
5. [**プロパティ**]ウィンドウに移動し、C1DropDown コントロールの **Height** を **30** に、**Width** を **100** に設定します。XAML は次のようになります。

   XAML

   ```
   <c1:C1DropDown Name="C1DropDown1" Height="30" Width="100" />
   ```

   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 **C1DropDown** 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が正しくセットアップされましたが、このアプリケーションを実行すると、**C1DropDown** コントロールにコンテンツがないことがわかります。次の手順では、**C1DropDown** コントロールにコンテンツを追加し、コントロールに対して実行可能ないくつかの操作を確認してみます。

手順 2:コントロールへのコンテンツの追加

前の手順では、WPF/Silverlight アプリケーションを作成し、プロジェクトに **C1DropDown** コントロールを追加しました。この手順では、**C1DropDown** コントロールにコンテンツを追加します。プロジェクトをカスタマイズしてアプリケーションの **C1DropDown** 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. [**XAML**]ビューに切り替えます。次の手順では、XAML マークアップをアプリケーションに追加することにより、ドロップダウンボックスにコンテンツを追加します。
2. **C1DropDown** コントロールにマークアップを追加します。次のようになります。

XAML

```
<c1:C1DropDown Name="C1DropDown1" Height="30" Width="100">
    <c1:C1DropDown.Header>
        <ContentControl VerticalAlignment="Center" HorizontalAlignment="Left"
Margin="5,0,0,0">
            <TextBlock x:Name="selection" Text="« 選択肢 »" FontSize="12"
Foreground="#FF3B76A2" TextAlignment="left" />
        </ContentControl>
    </c1:C1DropDown.Header>
</c1:C1DropDown>
```

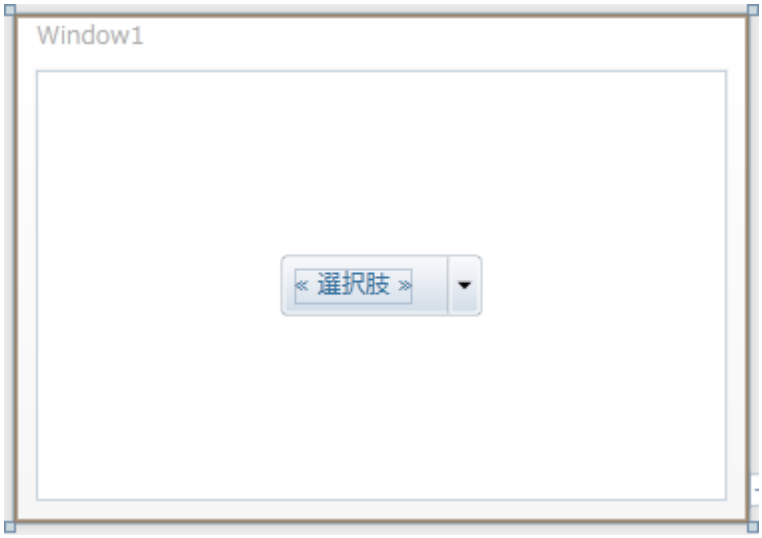
3. 追加した XAML マークアップの **</c1:C1DropDown.Header>** タグの直後に、次のマークアップを追加します。

XAML

```
<c1:C1DropDown.Content>
    <TreeView x:Name="treeSelection" Background="White">
        <TreeViewItem Header="南アメリカ">
            <TreeViewItem Header="ブラジル" />
            <TreeViewItem Header="アルゼンチン" />
            <TreeViewItem Header="ウルグアイ" />
        </TreeViewItem>
```

```
        <TreeViewItem Header="ヨーロッパ">
            <TreeViewItem Header="イタリア" />
            <TreeViewItem Header="フランス" />
            <TreeViewItem Header="イギリス" />
            <TreeViewItem Header="ドイツ" />
        </TreeViewItem>
    </TreeView>
</c1:C1DropDown.Content>
```

これにより、標準の TreeView コントロールが **C1DropDown** コントロールのコンテンツ領域に追加されます。[デザイン]ビューに次の図のようなウィンドウが表示されます。



この手順では、**C1DropDown** 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、コントロールをさらにカスタマイズしてからアプリケーションを実行して、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

WPF/Silverlight 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、**DropDown for WPF/Silverlight** の実行時の動作を確認し、さらにコントロールをカスタマイズするには、次の手順に従います。

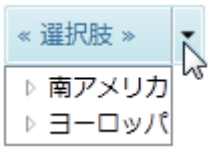
1. [**デバッグ**]メニューから[**デバッグ開始**]を選択し、実行時にアプリケーションがどのように表示されるかを確認します。

   アプリケーションは次の図のように表示されます。

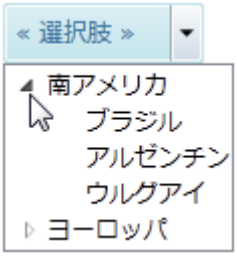
   

   C1DropDown コントロールは、シンプルなドロップダウンボックスとして表示されます。指定したテキストがコントロールのヘッダーに表示されていることを確認します。

2. **C1DropDown** コントロールのドロップダウン矢印をクリックします。TreeView コントロールが表示されることを確認します。

3. ドロップダウンボックスの項目を展開する – TreeView コントロールを通常どおりに操作できることを確認します。



4. もう一度 **C1DropDown**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が閉じることを確認します。

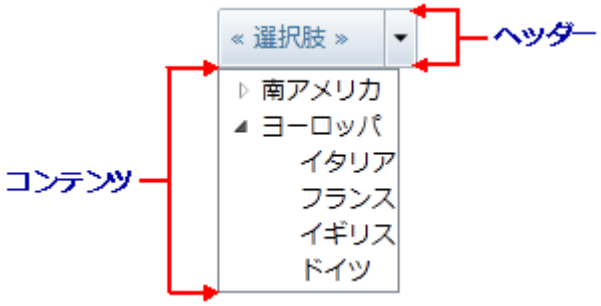
おめでとうございます!

これで **DropDown for WPF/Silverlight** クイックスタートは完了です。簡単な WPF/Silverlight アプリケーションを作成し、**DropDown for WPF/Silverlight** 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。

DropDown の要素

C1DropDown の要素

C1DropDown コントロールは、ヘッダー領域とコンテンツ領域の2つのパーツで構成されます。ヘッダーはドロップダウンボックスが開いていないときに表示され、コンテンツはドロップダウン領域をクリックすると表示されます。次の図に、ヘッダーとコンテンツ領域を示します。



コンテンツは、まったく追加しないこともできますが、ヘッダー領域とコンテンツ領域の一方に追加することも両方に追加することもできます。XAML でコンテンツをヘッダー領域とコンテンツ領域に追加して、**C1DropDown** コントロールをカスタマイズできます。たとえば、次のマークアップでは、上の図のようなドロップダウンコントロールが作成されます。

XAML

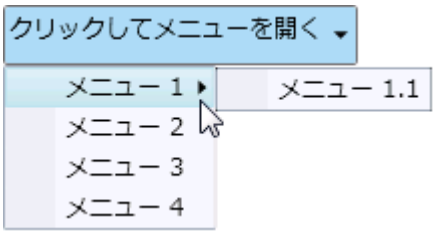
```
<c1:C1DropDown Height="36" HorizontalAlignment="Left" Margin="10,12,0,0"
Name="C1DropDown1" VerticalAlignment="Top" Width="101">
    <c1:C1DropDown.Header>
        <TextBlock Height="21" Name="TextBlock1" Text="« 選択肢 »" Width="120" />
    </c1:C1DropDown.Header>
    <c1:C1DropDown.Content>
        <TreeView HorizontalAlignment="Left" Name="TreeView1" Width="120">
            <TreeViewItem Header="南アメリカ">
                <TreeViewItem Header="ブラジル" />
                <TreeViewItem Header="アルゼンチン" />
```

```
                <TreeViewItem Header="ウルグアイ" />
            </TreeViewItem>
            <TreeViewItem Header="ヨーロッパ">
                <TreeViewItem Header="イタリア" />
                <TreeViewItem Header="フランス" />
                <TreeViewItem Header="イギリス" />
                <TreeViewItem Header="ドイツ" />
            </TreeViewItem>
        </TreeView>
    </c1:C1DropDown.Content>
</c1:C1DropDown>
```

ヘッダーとコンテンツを定義するために **<c1:C1DropDown.Header>** タグと **<c1:C1DropDown.Content>** タグが使用されていることに注意してください。これらのタグの内側にコントロールとコンテンツを追加できます。

C1DropDownButtonの要素

C1DropDownButton コントロールは C1DropDown コントロールに似ており、ヘッダー領域とコンテンツ領域の2つの部分で構成されます。ヘッダーはドロップダウンボックスが開いていないときに表示され、コンテンツはドロップダウン領域をクリックすると表示されます。たとえば、下の図のコンテンツ領域には、構造化メニューが含まれます。



コンテンツは、まったく追加しないこともできますが、ヘッダー領域とコンテンツ領域の一方に追加することも両方に追加することもできます。XAML でコンテンツをヘッダー領域とコンテンツ領域に追加して、**C1DropDown** コントロールをカスタマイズできます。たとえば、次のマークアップでは、上の図のようなドロップダウンコントロールが作成されます。

XAML

```
<c1:C1DropDownButton x:Name="ddl" Header="クリックしてメニューを開く"
HorizontalAlignment="Left">
    <c1:C1MenuList>
        <c1:C1MenuItem Header="メニュー 1">
                <c1:C1MenuItem Header="メニュー 1.1" />
        </c1:C1MenuItem>
        <c1:C1MenuItem Header="メニュー 2" />
        <c1:C1MenuItem Header="メニュー 3" />
        <c1:C1MenuItem Header="メニュー 4" />
    </c1:C1MenuList>
</c1:C1DropDownButton>
```

ヘッダーテキストは **<c1:C1DropDownButton>** タグで定義され,コンテンツは **<c1:C1DropDownButton></c1:C1DropDownButton>** タグ内に置かれます。

C1SplitButton の要素

ボタンとドロップダウンコンテンツ領域を結合した **C1SplitButton** コントロールは、標準の SplitButton に似ています。C1SplitButton は、配置できるコンテンツのタイプがより動的です。下の図で、C1SplitButton のヘッダーにはカラーピッカーアイコンがあり、色が選択されるとこれが変化します。また、C1SplitButton のコンテンツ領域にはカラーピッカーが配置されて

います。

コンテンツは、まったく追加しないこともできますが、ヘッダー領域とコンテンツ領域の一方に追加することも両方に追加することもできます。XAML でコンテンツをヘッダー領域とコンテンツ領域に追加して、**C1SplitButton** コントロールをカスタマイズできます。たとえば、次のマークアップでは、上の図のようなドロップダウンコントロールが作成されます。

XAML

```
<c1:C1SplitButton x:Name="btn" DropDownWidth="100" DropDownHeight="100"
HorizontalAlignment="Left" Click="btn_Click">
    <c1:C1DropDown.Header>
        <StackPanel>
            <TextBlock FontFamily="Times New Roman" FontSize="12" Text="A"
FontWeight="Bold" VerticalAlignment="Top" Margin="3 0 0 -2" />
            <Border x:Name="selectedColorBorder" Background="Red">
                <Border.Style>
                    <Style TargetType="Border">
                        <Setter Property="BorderThickness" Value="1" />
                        <Setter Property="BorderBrush" Value="#FF9E9DA7" />
                        <Setter Property="Width" Value="14" />
                        <Setter Property="Height" Value="5" />
                        <Setter Property="VerticalAlignment" Value="Bottom" />
                        <Setter Property="Margin" Value="1,0,0,1" />
                    </Style>
                </Border.Style>
            </Border>
        </StackPanel>
    </c1:C1DropDown.Header>
    <Border BorderThickness="1" BorderBrush="#FF207A9C" Background="#FFEFF5FF">
        <Border.Resources>
            <c1:ColorConverter x:Key="colorConverter" />
        </Border.Resources>
        <c1:C1SpectrumColorPicker ShowAlphaChannel="False" Margin="4" Color="{Binding
Background, ElementName=selectedColorBorder, Mode=TwoWay, Converter={StaticResource
colorConverter}}" />
    </Border>
</c1:C1SplitButton>
```

<c1:C1DropDown.Header> はヘッダー要素を定義し、コンテンツ要素は **<c1:C1SplitButton></c1:C1SplitButton>** タグ内に置かれます。

DropDown の作成

XAML の場合 - WPF

C1DropDown コントロールを XAML マークアップを使って作成するには、次の手順に従います。

1. Visual Studio のソリューションエクスプローラで、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 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/wpf/Basic" を初期状態の <Window> タグに追加し、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 x:Class="C1WPFPropertyGridCS102809.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="288" Width="418"
xmlns:c1="http://schemas.componentone.com/wpf/Basic">
```

3. <c1:C1DropDown> タグをプロジェクトの <Grid> タグ内に追加して、C1DropDown 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:C1DropDown Height="30" Name="C1DropDown1" Width="100"/>
</Grid>
```

このマークアップは、「**C1DropDown1**」という名前の空の C1DropDown 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。

XAML の場合 - Silverlight

1. Visual Studio のソリューション 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.Silverlight.dll アセンブリを選択し、[OK]をクリックします。
2. xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml" を初期の <UserControl> 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
<UserControl xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:c1="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight" x:Class="C1DropDown.MainPage"
Width="640" Height="480">
```

3. <c1:C1DropDown> タグをプロジェクトの <Grid> タグ内に追加して、C1DropDownを作成します。マークアップは次のようになります。

```
XAML
<Grid x:Name="LayoutRoot" Background="White">
    <c1:C1DropDown Height="30" Name="c1dropdown1" Width="100" />
</Grid>
```

このマークアップは、「c1dropdown1」という名前の空の C1DropDown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 - WPF

1. Visual Studio のソリューションエクスプローラで、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 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。

XAML
```
<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
```

C#
```
using C1.WPF;
```

4. ページのコンストラクタに、C1DropDown 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 c1DropDown1 as New C1DropDown
    c1DropDown1.Height = 30
    c1DropDown1.Width = 100
    LayoutRoot.Children.Add(c1DropDown1)
End Sub
```

C#
```
public Window1()
{
    InitializeComponent();
    C1DropDown c1DropDown1 = new C1DropDown();
    c1DropDown1.Height = 30;
    c1DropDown1.Width = 100;
    LayoutRoot.Children.Add(c1DropDown1);s
}
```

このコードは、「**c1DropDown1**」という名前の空の **C1DropDown** コントロールを作成し、コントロールのサイズを設定し、コントロールをウィンドウに追加します。

コードの場合 - Silverlight

1. x:Name="C1ComboBox1" を <c1:C1ComboBox> タグに追加します。これで、このオブジェクト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.cs または MainWindow.xaml.cs ページを開きます
3. 以下の名前空間をプロジェクトにインポートします。

Visual Basic
```
Imports System.Collections.Generic
```

C#

```
using System.Collections.Generic;
```

4. InitializeComponent() メソッドの下に次のコードを追加してリストを作成します。

Visual Basic

```
Dim dropDownList As New List(Of String)()
dropDownList.Add("C1ComboBoxItem1")
dropDownList.Add("C1ComboBoxItem2")
dropDownList.Add("C1ComboBoxItem3")
dropDownList.Add("C1ComboBoxItem4")
```

C#

```
List<string> dropDownList = new List<string>();
dropDownList.Add("C1ComboBoxItem1");
dropDownList.Add("C1ComboBoxItem2");
dropDownList.Add("C1ComboBoxItem3");
dropDownList.Add("C1ComboBoxItem4");
```

5. **ItemsSource** プロパティを設定して、**ComboBox** にこのリストを追加します。

Visual Basic

```
C1ComboBox1.ItemsSource = dropDownList
```

C#

```
C1ComboBox1.ItemsSource = dropDownList;
```

6. プログラムを実行します。

設計時

1. ウィンドウのデザイン領域をクリックして選択します。
2. ツールボックスの[**C1DropDown**]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、**C1DropDown** 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。

DropDown の機能

ドロップダウンとの対話

XAMLの場合

Button コントロールをドロップダウンボックスに追加するには、**<c1:C1DropDown>>** タグに **<Button Height="23" Name="button1" Width="75">Hello World!<Button>** を追加します。次のようになります。

XAML

```
<c1:C1DropDown Height="30" Name="c1DropDown1" Width="100">
    <Button Height="23" Name="button1" Width="75">Hello World!</Button>
```

```
</c1:C1DropDown>
```

コードの場合

Button コントロールをドロップダウンボックスに追加するには、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 C1Button1 as New Button
    C1Button1.Content = "Hello World!"
    C1DropDown1.Content = c1button1
End Sub
```

C#

```
public Window1()
//or public MainPage() in Silverlight
{
    InitializeComponent();
    Button c1button1 = new Button();
    c1button1.Content = "Hello World!";
    c1DropDown1.Content = c1button1;
}
```

C1DropDown コントロールに複数の項目を追加する場合は、**C1DropDown** コントロールに Grid またはその他のパネルを追加してから、パネルに項目を追加します。

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 **C1DropDown** 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。

XAML の場合

ドロップダウンウィンドウの方向を変更するには、**<c1:C1DropDown>** タグに **DropDownDirection="ForceAbove"** を追加します。次のようになります。

XAML

```
<c1:C1DropDown Height="30" Name="c1DropDown1" Width="100"
DropDownDirection="ForceAbove"/>
```

コードの場合

ドロップダウンボックス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.C1DropDown1.DropDownDirection = DropDownDirection.ForceAbove
```

C#

```
this.c1DropDown1.DropDownDirection = DropDownDirection.ForceAbove;
```

デザイナの場合

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. **C1DropDown** 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。

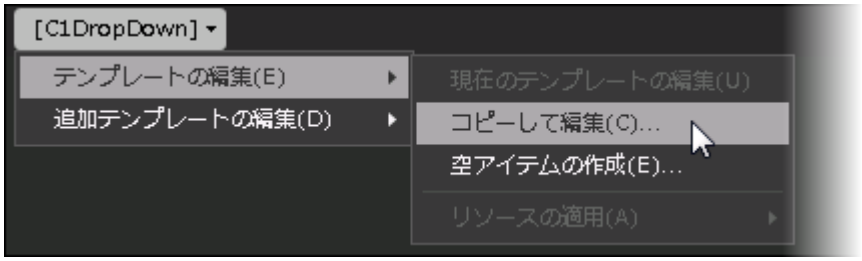
レイアウトおよび外観

ドロップダウンテンプレート

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、**DropDown 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

1. テンプレートにアクセスするには、Microsoft Expression Blend で、**C1DropDown** コントロールを選択し.
2. メニューから[**テンプレートの編集**]を選択します。
3. [**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。
4. [**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。

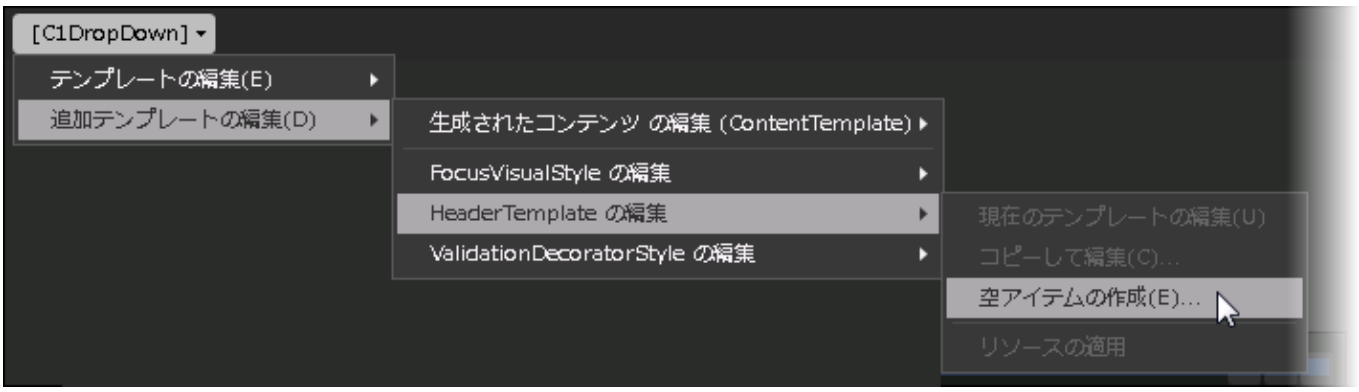


新しく作成されたテンプレートは、[**オブジェクトとタイムライン**]ウィンドウに表示されます。Template プロパティを使ってテンプレートをカスタマイズできます。

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

追加のテンプレート

デフォルトテンプレートのほかに、**C1DropDown** 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 **C1DropDown** 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。



FilePicker

FilePicker for WPF/Silverlight を使用すると、デスクトップアプリケーションでファイル選択が可能になります。**C1FilePicker** コントロールはコンボボックスに似ていますが、ドロップダウンリストではなくファイル選択ダイアログボックスを表示します。

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

- **フィルタファイルの種類**

  ファイル拡張子またはカテゴリに基づいて、検索するファイルの種類を制限します。詳細については、「ファイルのフィルタリング」を参照してください。画像ファイルのフィルタを追加する例については「ファイルフィルタを追加する」トピックを参照してください。

- **複数ファイルの選択**

  エンドユーザーは、一度に1つのファイルを選択することも、複数のファイルを選択して処理時間を短縮することもできます。詳細については、「複数ファイルの選択」を参照してください。C1FilePicker コントロールで複数ファイルを選択する例については、「複数のファイルを選択する」トピックを参照してください。

- **ウォーターマークのサポート**

  ユーザーのファイル選択を助けるウォーターマークを指定できます。詳細については、「ウォーターマーク」を参照してください。ウォーターマークを削除する例については、「ウォーターマークを削除する」トピックを参照してください。

- **ClearStyle を使用して簡単に色を変更する**

  エンドユーザーは、一度に1つのファイルを選択することも、複数のファイルを選択して処理時間を短縮することもできます。詳細については、「ComponentOne ClearStyle 技術」トピックを参照してください。

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。

クイックスタート

手順 1:アプリケーションの作成

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**]→[**新規作成**]→[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 アプリケーション]

または[Silverlight アプリケーション]を選択します。プロジェクトの名前(たとえば、「QuickStart」)を入力し、[OK]をクリックします。プロジェクトが作成され、MainWindow.xaml ファイルが開きます。

3. 参照の追加ダイアログボックスで、以下のアセンブリを見つけて選択し、[OK]をクリックしてプロジェクトに参照を追加します。

   - **WPF**: C1.WPF.4.dll
   - **Silverlight**: C1.Silverlight.5.dll

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. 次の XAML マークアップを MainWindow.xaml ファイルの **<Grid>** タグと **</Grid>** タグの間に追加します。

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

6. ツールボックスに移動し、[**C1FilePicker**]アイコンをダブルクリックして、コントロールをアプリケーションに追加します。

7. **C1FilePicker** タグを次のように編集します。

```
XAML
<c1:C1FilePicker x:Name="C1FilePicker1" Grid.Row="0" Margin="15" Height="30"
SelectedFilesChanged="C1FilePicker_SelectedFilesChanged" />
```

このマークアップは、コントロールの名前を決定し、コントロールが表示されるグリッド行を指定します。また、コントロールの高さとコントロール周囲のマージンを設定し、**SelectedFilesChanged** イベントのイベントハンドラを指定します。イベントハンドラコードは後の手順で追加します。

8. 次の XAML マークアップを MainPage.xaml ファイルの **<c1:C1FilePicker>** タグと **</Grid>** タグの間に追加します。

```
XAML
<ScrollViewer Grid.Row="1" Margin="15,0,15,0">
    <ListBox x:Name="ListBox" />
</ScrollViewer>
<StackPanel Grid.Row="2" Name="stackPanel1" Orientation="Horizontal"
HorizontalAlignment="Center">
    <Button Content="ファイル選択を解除" Height="30" Margin="0,15,15,15"
Name="button1" Width="150" Grid.Row="2" Click="button1_Click" />
    <Button Content="リスト項目を削除" Height="30" Margin="15,15,0,15"
Name="button2" Width="150" Grid.Row="2" Click="button2_Click" />
</StackPanel>
```

このマークアップは、**ListBox** コントロールを追加します。これで、**C1FilePicker** を使用して選択したローカル画像を表示できるようになります。また、このマークアップにより、**C1FilePicker** コントロールのコンテンツをクリアするボタンと **ListBox** のコンテンツをクリアするボタンも追加されます。

手順 2:アプリケーションへのコードの追加

前の手順では、WPF/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、コントロー

ルは動作しません。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** ファイルまたは**MainWindow.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 文をページの先頭に追加します（ページに含まれていない場合）。

Visual Basic

```
Imports System.Windows.Media.Imaging
Imports C1.WPF
または
Imports System.Windows.Media.Imaging
Imports C1.Silverlight
```

C#

```
using System.Windows.Media.Imaging;
using C1.WPF;
または
using System.Windows.Media.Imaging;
using C1.Silverlight;
```

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

Visual Basic

```
Private Sub C1FilePicker_SelectedFilesChanged(sender As Object, e As EventArgs)
    If C1FilePicker1.SelectedFile IsNot Nothing Then
        Dim stream = C1FilePicker1.SelectedFile.OpenRead()
        Dim source = New BitmapImage()
        source.SetSource(stream)
        Dim image = New Image()
        image.Source = source
        image.Stretch = Stretch.Uniform
        image.Height = 100
        ListBox.Items.Add(image)
    End If
End Sub
```

C#

```
private void C1FilePicker_SelectedFilesChanged(object sender, EventArgs e)
{
    if (C1FilePicker1.SelectedFile != null)
    {
        var stream = C1FilePicker1.SelectedFile.OpenRead();
        var source = new BitmapImage();
        source.SetSource(stream);
        var image = new Image();
        image.Source = source;
        image.Stretch = Stretch.Uniform;
        image.Height = 100;
```

```
            ListBox.Items.Add(image);
        }
    }
```

このコードにより、**SelectedFilesChanged** イベントが処理されます。ユーザーが **C1FilePicker** コントロールを使用して画像を選択すると、その画像はカスタマイズされて ListBox コントロールに追加されます。

4. **Button** コントロールの **Click** イベントを処理する次のコードをページに追加します。

Visual Basic

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    C1FilePicker1.ClearSelection()
End Sub
Private Sub button2_Click(sender As Object, e As RoutedEventArgs)
    ListBox.Items.Clear()
End Sub
```

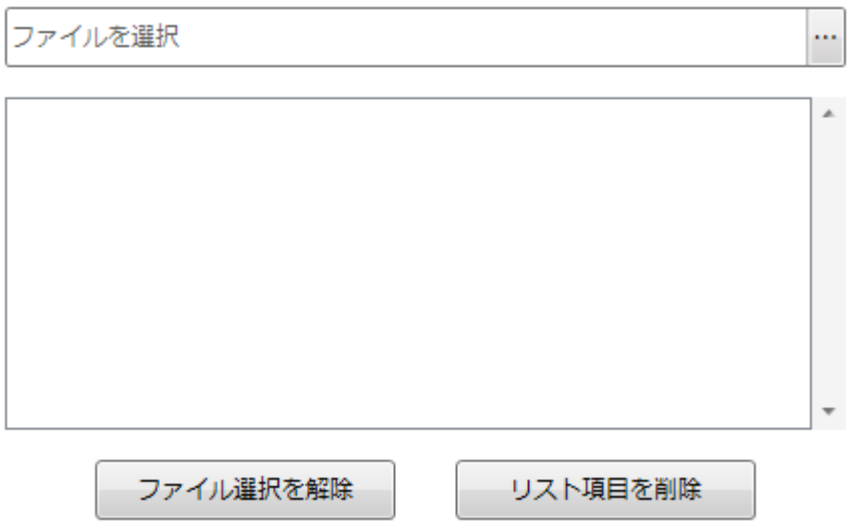
C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    C1FilePicker1.ClearSelection();
}
private void button2_Click(object sender, RoutedEventArgs e)
{
    ListBox.Items.Clear();
}
```

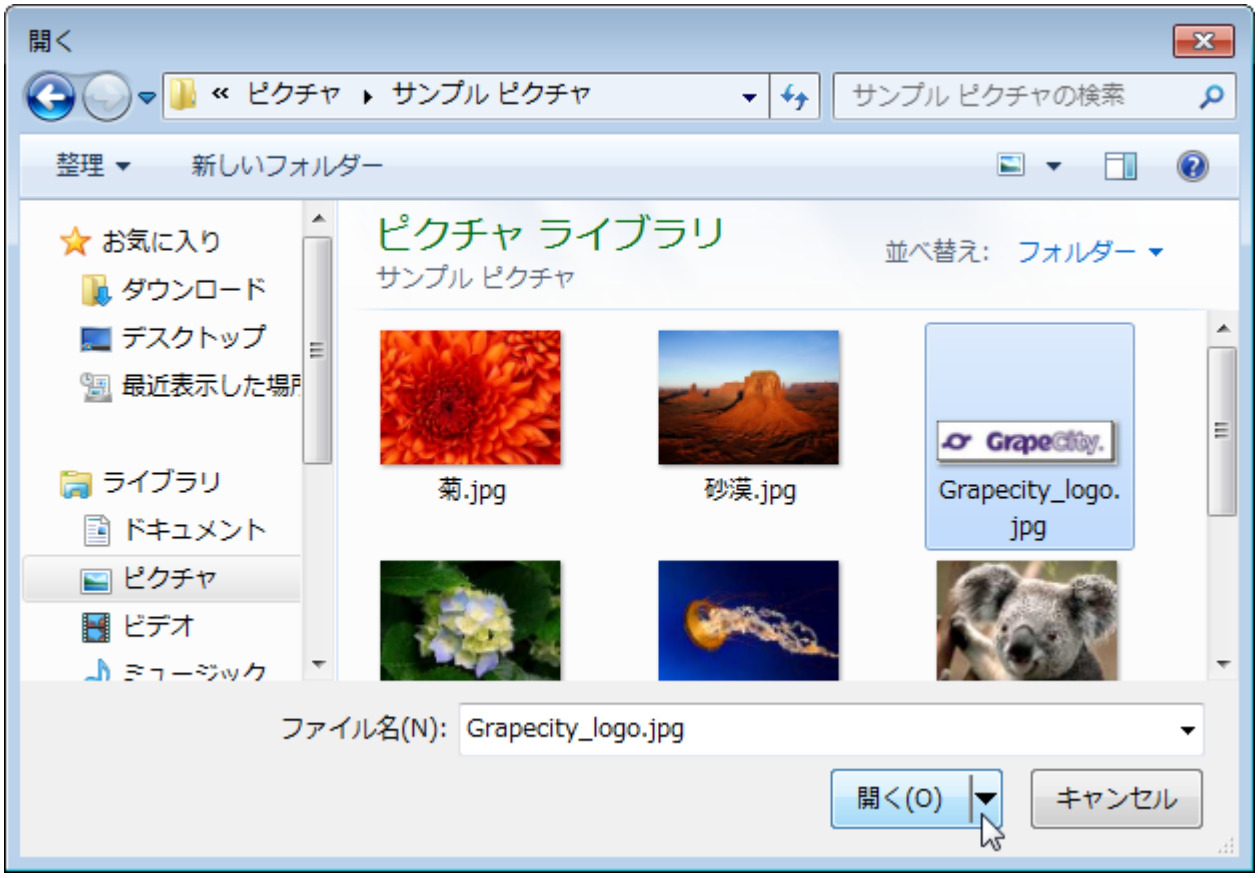
手順 3アプリケーションの実行

WPF アプリケーション と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。



2. C1FilePicker コントロールで省略符ボタン(...)をクリックします。[開く]ダイアログボックスが表示されます。
3. [開く]ダイアログボックスで画像を探して選択し、[開く]をクリックして、選択した画像を開きます。



4. 選択した画像がリストボックスに表示され、ファイルの名前が **C1FilePicker** コントロールに表示されます。

5. [**ファイル選択を削除**]ボタンをクリックします。**C1FilePicker** コントロールにファイルの名前が表示されなくなります。

6. **C1FilePicker** コントロールで省略符ボタンをもう一度クリックします。[**開く**]ダイアログボックスが表示されます。
7. [開く]ダイアログボックスで、複数の画像を選択してみます。ここでは、複数の画像を選択できません。これは、デフォルトでは、**Multiselect** プロパティが **False** に設定されているためです。複数のファイルを一度に選択するには、**Multiselect** プロパティを「True」に設定する必要があります。
8. [開く]ダイアログボックスでファイルを選択し、[開く]をクリックします。2番目のファイルが C1FilePicker コントロールにリストされ、リストボックスに追加されていることに注意してください。

9. [**リスト項目を削除**]ボタンをクリックします。ファイルのリストがクリアされます。

ここまでの結果

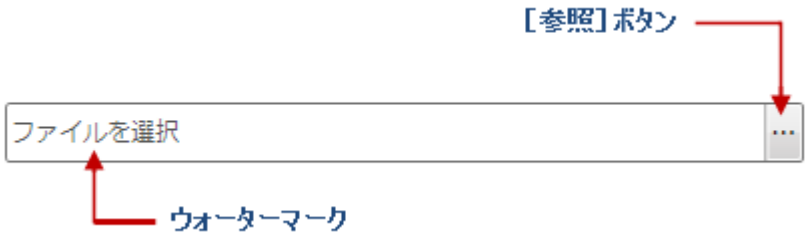
おめでとうございます。これで **FilePicker for WPF/Silverlight** クイックスタートは終了です。**FilePicker for WPF/Silverlight** を使用して画像ファイルを選択する簡単なアプリケーションを作成しました。ここで選択された画像は他のコントロールに表示されます。

ファイルピッカー要素

FilePicker for WPF/Silverlightは、ファイルの選択機能を提供します。選択したファイルは、**Upload for**

WPF/Silverlightを使用してアップロードしたり、他のコントロールで使用することができます。基本 **C1FilePicker** コントロールは、次の図のようになります。

コントロールは、ウォーターマークまたは選択したファイルの名前をリストするボックス、および**省略符**で示される[**参照**]ボタンで構成されています。



[参照]ボタン

ユーザーが実行時に[**参照**]ボタンをクリックすると、**OpenFileDialog** ダイアログボックスが表示されます。ユーザーは **OpenFileDialog** を使用して、ローカルコンピュータまたはネットワークで接続されたコンピュータ上にある1つまたは複数のファイルを選択できます。選択したファイルは、**SelectedFiles** コレクションに追加されます。詳細については、「[開く]ダイアログボックス」トピックを参照してください。

[参照]ボタンのコンテンツおよび外観をカスタマイズすることができます。それには、**BrowseContent** プロパティをオブジェクトに設定します。選択したオブジェクトは[**参照**]ボタンとして表示されます。

ウォーターマーク

デフォルトでは、ウォーターマークは **C1FilePicker** コントロールのテキスト領域に表示されます。ウォーターマークにより、指示や推奨事項が実行時にユーザーに示されます。たとえば、デフォルトのウォーターマークテキストは「ファイルを選択」です。

テキストは淡色表示されることに注意してください。1つまたは複数のファイルを選択すると、ファイルはこのテキスト領域に表示されます。このとき、ウォーターマークと区別するために、ファイルは濃い色で表示されます。ファイルを選択した後に、ClearSelection メソッドを使用して **C1FilePicker** コントロールからファイルをクリアすると、ウォーターマークが再度表示されます。例については、「**選択したファイルをクリアする**」を参照してください。

ウォーターマークをカスタマイズするには、Watermark プロパティを、表示する値に設定します。例については、「ウォーターマークを削除する」を参照してください。ウォーターマークを表示しない場合は、**Watermark** プロパティを空白の値（スペース文字など）に設定します。

ウォーターマークを削除する

デフォルトでは、ウォーターマークは **C1FilePicker** コントロールのテキスト領域に表示され、実行時に指示や推奨事項をユーザーに示します。詳細については、「ウォーターマーク」トピックを参照してください。ウォーターマークテキストをカスタマイズするには、**Watermark** プロパティを、表示する値に設定します。

ウォーターマークを表示しない場合は、**Watermark** プロパティを空白の値に設定します（次の手順を参照）。

XAML の場合

たとえば、Watermark プロパティを設定するには、Watermark="" を <c1:C1FilePicker> タグに追加します。次のようになります。

XAML

```
<c1:C1FilePicker HorizontalAlignment="Left" Margin="112,36,0,0" Name="C1FilePicker1"
VerticalAlignment="Top" Width="161" Watermark="" />
```

コードの場合

たとえば、Watermark 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.C1FilePicker1.Watermark = ""
```

C#

```
this.c1FilePicker1.Watermark = "";
```

設計時

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

1. C1FilePicker 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、[**ウォーターマーク**]項目を探します。
3. [**ウォーターマーク**]項目の横にあるテキストボックスをクリックし、現在のテキストを削除します。

   これで、Watermark プロパティに空白の値が設定されます。

アプリケーションを実行し、表示される **C1FilePicker** コントロールのキャプションバーにウォーターマークが含まれていないことを確認します。

テキスト配置を変更する

通常、**C1FilePicker** コントロールでは、現在選択されているファイルがコントロールのテキスト領域に表示されます。デフォルトでは、このテキストは左揃えになっていますが、必要に応じてテキストの配置を変更できます。テキストの配置を設定するには、**TextAlignment** プロパティを使用します。テキスト配置のオプションには、Left(デフォルト)、Center、Right があります。詳細については、以下の手順を参照してください。

XAML の場合

たとえば、**TextAlignment** プロパティを Right に設定するには、**TextAlignment="Right" <c1:C1FilePicker>**を タグに追加します。次のようになります。

XAML

```
<c1:C1FilePicker HorizontalAlignment="Left" Margin="112,36,0,0" Name="C1FilePicker1"
VerticalAlignment="Top" Width="161" TextAlignment="Right" />
```

コードの場合

たとえば、TextAlignment プロパティを Right 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.C1FilePicker1.TextAlignment = TextAlignment.Right
```

C#

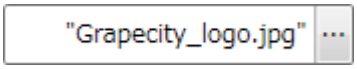
```
this.c1FilePicker1.TextAlignment = TextAlignment.Right;
```

設計時

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

1. **C1FilePicker** 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、[**TextAlignment**]項目を探します。
3. [**TextAlignment**]項目の横にあるドロップダウン矢印をクリックし、[**Right**]を選択します。

これで、テキストは右に揃えられます。アプリケーションを実行し、ファイルを選択します。C1FilePicker コントロールで、ファイルの名前が右揃えになっていることを確認します。



FilePicker の機能

ファイルフィルタを追加する

ユーザーが実行時に[**参照**]ボタンをクリックしたり **OpenFileDialog** ダイアログボックスを開いた場合、通常は開くファイルの種類を自由に選択できます。このとき、**OpenFileDialog** ダイアログボックスにフィルタオプションは表示されません。ユーザーに対して表示され、**OpenFileDialog** ダイアログボックスで選択できるファイルの種類を制限したい場合があります。たとえば、画像ファイルや特定のドキュメントタイプのみを選択できるようにする場合です。ユーザーが選択できるファイルの種類をカスタマイズするには、**Filter** プロパティを特定のフィルタに設定します。

基本的な使用方法では、**OpenFileDialog** ダイアログボックスでフィルタを適用すると、特定の拡張子を持つファイルのみを表示できます。**Filter** プロパティは、**FileDialog.Filter** プロパティと同様に機能します。**Filter** プロパティをフィルタ文字列に設定する必要があります。実装した各フィルタオプションのフィルタ文字列には、フィルタの説明に続いて縦棒(|)およびフィルタパターンが含まれています。異なるフィルタオプションの文字列は、縦棒で区切られています。

次に、フィルタ文字列の例を示します。

- テキストファイル (*.txt)|*.txt|すべてのファイル (*.*)|*.*

複数のフィルタパターンをフィルタに追加するには、次の例のように、ファイルの種類をセミコロンで区切ります。

- 画像ファイル (*.BMP;*.JPG;*.GIF)|*.BMP;*.JPG;*.GIF|すべてのファイル (*.*)|*.*

FilterIndex プロパティを使用して、ユーザーに最初に表示するフィルタオプションを設定します。デフォルトでは、**FilterIndex** プロパティは「1」に設定されています。つまり、フィルタロジックに最初にリストされているフィルタが最初に表示されます。たとえば、前述の例のように、1つの画像フィルタとすべてのファイルのオプションを指定した場合、すべてのファイルのオプションを最初に表示する(フィルタを適用しないでダイアログボックスを表示する)には、**FilterIndex** プロパティを「2」に設定します。

XAML の場合

たとえば、**Filter** プロパティをフィルタ画像ファイルに設定するには、Filter="画像ファイル (*.PNG;*.JPG;*.GIF)|*.PNG;*.JPG;*.GIF|すべてのファイル (*.*)|*.*" を <c1:C1FilePicker> タグに追加します。次のようになります。

XAML

```
<c1:C1FilePicker HorizontalAlignment="Left" Margin="112,36,0,0" Name="C1FilePicker1"
VerticalAlignment="Top" Width="161" Filter="画像ファイル
(*.PNG;*.JPG;*.GIF)|*.PNG;*.JPG;*.GIF|すべてのファイル (*.*)|*.*" />
```

コードの場合

たとえば、Filter プロパティをフィルタ画像ファイルに設定するには、プロジェクトに次のコードを追加します。

Visual Basic

```
Me.C1FilePicker1.Filter = "画像ファイル(*.BMP;*.JPG;*.GIF)|*.PNG;*.JPG;*.GIF|すべてのファイル (*.*)
```

C#

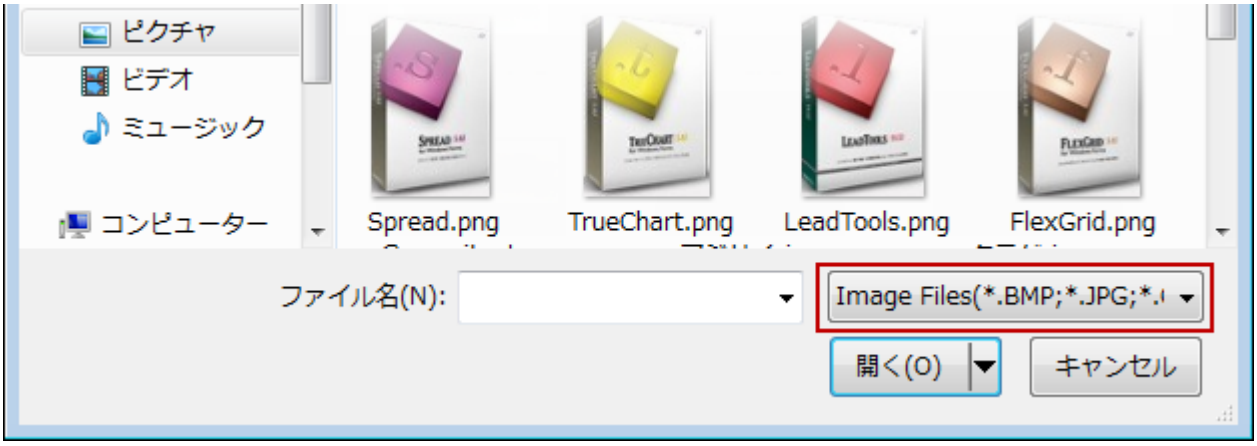
```
this.c1FilePicker1.Filter = "画像ファイル(*.BMP;*.JPG;*.GIF)|*.PNG;*.JPG;*.GIF|すべてのファイル (*.*)
```

設計時

設計時に、Filter プロパティをフィルタ画像ファイルに設定するには、次の手順に従います。

1. **C1FilePicker** 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、[フィルタ]項目を探します。
3. [フィルタ]項目の横にあるテキストボックスに、「画像ファイル(*.PNG;*.JPG;*.GIF)|*.PNG;*.JPG;*.GIF|すべてのファイル(*.*)|*.*」と入力します。

フィルタが OpenFileDialog ダイアログボックスに適用されます。アプリケーションを実行して、**C1FilePicker** コントロールの[**参照**]ボタンをクリックします。フィルタが適用され、拡張子が PNG、JPG、または GIF であるファイルのみがダイアログボックスに表示されることを確認します。



選択

複数のファイルを選択する

ユーザーは **FilePicker for WPF/Silverlight** を使用して、実行時に複数のファイルを選択できます。ただし、デフォルトでは、このオプションは選択されていません。デフォルトでは、ユーザーは実行時に1つのファイルを選択できます。ユーザーが複数のファイルを選択できるようにするには、次の手順に従って、**Multiselect** プロパティを True に設定します。

XAML の場合

たとえば、**Multiselect** プロパティを設定するには、Multiselect="True" を <c1:C1FilePicker> タグに追加します。次のようになります。

XAML

```
<c1:C1FilePicker HorizontalAlignment="Left" Margin="112,36,0,0" Name="C1FilePicker1"
VerticalAlignment="Top" Width="161" Multiselect="True" />
```

コードの場合

たとえば、**Multiselect** 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.C1FilePicker1.Multiselect = True
```

C#

```
this.c1FilePicker1.Multiselect = true;.
```

設計時

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

1. **C1FilePicker** 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、[Multiselect]項目を探します。
3. Multiselect 項目の横にあるチェックボックスをオンにします。

これで、ユーザーが実行時に OpenFileDialog ダイアログボックスを使用して複数のファイルを選択できるように、Multiselect プロパティが設定されます。

選択したファイルをクリアする

ユーザーが **C1FilePicker** コントロールを使用して実行時に1つのファイルを選択すると、そのファイルの名前がコントロールのテキスト領域に表示されます。**C1FilePicker** で選択したすべてのファイルをクリアし、選択したファイルの名前をコントロールから削除するには、**ClearSelection** メソッドを使用します。次の例では、**C1FilePicker** コントロールをクリアするボタンをアプリケーションに追加します。

たとえば、アプリケーションにボタンを追加して、ボタンの Click イベントハンドラに次のコードを追加します。

Visual Basic

```
C1FilePicker1.ClearSelection().
```

C#

```
c1FilePicker1.ClearSelection();
```

HeaderedContentControl

HeaderedContentControl for WPF/Silverlight を使用して、コンテンツブロックを簡単にレイアウトして表示できます。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。ヘッダーは水平または垂直に配置でき、コンテンツパネルはヘッダーのどちら側にも配置できます。

主要な機能

HeaderedContentControl 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、**HeaderedContentControl for WPF/Silverlight** を最大限に活用してください。

- **フレーム的な使用方法の実装**

  C1HeaderedContentControl を使用すると、アプリケーションでコンテナのスタイルを簡単に再利用できます。それには、**C1HeaderedContentControl** の新しいテンプレートを定義し、それを任意の場所で使用するだけです。これにより、カスタムコンテナを構成するすべてのビジュアル要素を何度もコピーして貼り付ける必要がなくなるため、時間を節約できます。

- **フィールドのグループ化**

  C1HeaderedContentControl を使用すると、フォーム内の互いに似たフィールドを分類して、利用性を向上させることができます。

- **Silverlight Toolkit テーマのサポート**

  Cosmopolitan、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、これらを使用して UI にスタイルを追加できます。

クイックスタート

このクイックスタートガイドは、**HeaderedContentControl 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 **C1HeaderedContentControl** 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 1/3：C1HeaderedControlControl を含むアプリケーションの作成

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

次の手順を実行します。

1. Visual Studio で、**[ファイル]**→**[新規作成]**→**[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で言語を選択し、テンプレートリストで**[WPF アプリケーション]**を選択します。
3. プロジェクトの**名前**と**場所**を入力し、**[OK]**をクリックします。作成された新しい WPF 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. XAML で次の名前空間を追加します。

```
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
```

5. 次のコードを使用して、C1HeaderedContentControl をプロジェクトに追加します。

```
<c1:C1HeaderedContentControl></c1:C1HeaderedContentControl>
```

この手順では、**C1HeaderedContentControl** を含む WPF アプリケーションを作成しました。次の手順では、**C1HeaderedContentControl** をカスタマイズします。

手順 2/3：C1HeaderedContentControl のカスタマイズ

前の手順では、**C1HeaderedContentControl** 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、コンテンツパネルにコンテンツを追加して、**C1HeaderedContentControl** をカスタマイズします。

次の手順を実行します。

1. **C1HeaderedContentControl** 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。
   - **BorderBrush** プロパティの 16 進数値を "#FF198315" に設定します。
   - **HeaderForeground** プロパティの 16 進数値を "#FF198315" に設定します。
   - **BorderThickness** プロパティを "4, 4, 4, 4" に設定します。
   - **Header** プロパティを "Map of the World" に設定します。
   - **HeaderFontFamily** を Arial に設定します。
   - **HeaderFontSize** を "15" に設定します。
   - **HeaderHorizontalContentAlignment** プロパティを Left に設定します。

   XAML は次のようになります。

   XAML copyCode

```
<c1:C1HeaderedContentControl Header="Map of the World" BorderBrush="#FF198315"
                             HeaderForeground="#FF198315"
                             BorderThickness="4, 4, 4, 4"
HeaderFontFamily="Arial" HeaderFontSize="15" HorizontalHeaderAlignment="Left">
    <c1:C1HeaderedContentControl.Content>
        <c1:C1Maps x:Name="c1Maps1" Height="200" Width="300"></c1:C1Maps>
    </c1:C1HeaderedContentControl.Content>
</c1:C1HeaderedContentControl>
```

2. **C1Maps** をダブルクリックしてコンテンツ領域に追加します。

この手順では、**C1HeaderedContentControl** をカスタマイズしました。次の手順では、プロジェクトを実行し、クイックスタートの結果を確認します。

手順 3/3：プロジェクトの実行

前の 2 つの手順では、**C1HeaderedContentControl** を含むプロジェクトを作成し、**C1HeaderedContentControl** をカスタマイズしました。この手順では、プロジェクトを実行し、このクイックスタートの結果を確認します。

次の手順を実行します。

1. [F5]キーを押してプロジェクトを実行し、次のように **C1HeaderedContentControl** が表示されることを確認します。

2. **C1Maps** コントロールの下矢印をクリックし、**C1Maps** コントロールがヘッダーに関係なく移動することを確認します。

おめでとうございます。これで、**HeaderedContentControl for WPF/Silverlight** の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、「**C1HeaderedContentControl** の要素」と「HeaderedContentControl for WPF/Silverlight タスクベースのヘルプ」を参照してください。

C1HeaderedContentControl の要素

C1HeaderedContentControl は **HeaderedContentControl** 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すると、**C1HeaderedContentControl** は空で表示されます。

プロジェクトに **C1HeaderedContentControl** を追加したら、そのコントロールに独自のヘッダーやコンテンツを追加できます。以下のトピックでは、**C1HeaderedContentControl** のヘッダーバーとコンテンツパネルの概要を示します。

C1HeaderedContentControl のヘッダー

デフォルトでは、**C1HeaderedContentControl** のヘッダーバーにはテキストが含まれていません。ヘッダーバー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を使用してテキストのスタイルを設定できます（「テキストのプロパティ」参照）。ヘッダーには、さまざまな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ベースのヘルプについては、「ヘッダーバーへのコンテンツの追加」を参照してください。

C1HeaderedContentControl の **HeaderPadding** プロパティを使用して、ヘッダーのパディングを調整できます。

属性構文とプロパティ要素構文

C1HeaderedContentControl 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。

```
<c1:C1HeaderedContentControl Header="Hello World"/>
```

ただし、コンテンツパネルに、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 copyCode

```
<c1:C1HeaderedContentControl Width="150" Height="55"
Name="C1HeaderedContentControl1">
    <c1:C1HeaderedContentControl.Header>
        <Grid HorizontalAlignment="Stretch">
            <Grid.ColumnDefinitions>
```

```
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
            <TextBlock Text="C1HeaderedContentControl Header" />
        </Grid>
    </c1:C1HeaderedContentControl.Header>
</c1:C1HeaderedContentControl>
```

C1HeaderedContentControl のコンテンツパネル

最初、**C1HeaderedContentControl** のコンテンツパネルは空です。コンテンツパネルには、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。Visual Studio 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パネル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。

コンテンツパネルにテキストを追加するには、**C1HeaderedContentControl** の **Content** プロパティを設定するか、**TextBox** 要素をコンテンツパネルに追加します。実行時にコンテンツパネルに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。

C1HeaderedContentControl などのコンテンツコントロールは、子要素を一度に 1 つだけ保持できます。ただし、この問題は、**C1HeaderedContentControl** 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、**C1HeaderedContentControl** 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、コンテンツパネルに複数のコントロールを表示できます。

コンテンツパネルへのコンテンツの追加に関するタスクベースのヘルプについては、「コンテンツパネルへのコンテンツの追加」を参照してください。

属性構文とプロパティ要素構文

C1HeaderedContentControl コンテンツパネルに単純な要素（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。

```
<c1:C1HeaderedContentControl Content="Hello World"/>
```

ただし、コンテンツパネルに、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 copyCode

```
<c1:C1HeaderedContentControl Width="150" Height="55"
Name="C1HeaderedContentControl_Content">
    <c1:C1HeaderedContentControl.Content>
        <StackPanel>
            <TextBlock Text="Hello"/>
            <TextBlock Text="World"/>
        </StackPanel>
    </c1:C1HeaderedContentControl.Content>
</c1:C1HeaderedContentControl>
```

HeaderedContentControl for WPF/Silverlight のレイアウトおよび外観

以下のトピックでは、**C1HeaderedContentControl** 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書式設定およびレイアウトしたり、操作をカスタマイズすることもできます。

HeaderedContentControl for WPF/Silverlight の外観プロパティ

HeaderedContentControl for WPF/Silverlight には、コントロールの外観をカスタマイズするためのプロパティがいくつか含まれています。HeaderedContentControl に表示されるテキストの外観を変更したり、グラフィック要素をカスタマイズすることができます。以下のトピックでは、これらの外観プロパティの一部について説明します。

テキストのプロパティ

以下のプロパティを使用すると、**C1HeaderedContentControl** のテキストの外観をカスタマイズできます。

プロパティ	説明
FontFamily	コントロールのフォントファミリを取得または設定します。これは依存プロパティです。
FontSize	フォントサイズを取得または設定します。これは依存プロパティです。
FontStretch	フォントを画面上で伸縮する比率を取得または設定します。これは依存プロパティです。
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。
FontWeight	指定されたフォントの太さを取得または設定します。これは依存プロパティです。
Header	HeaderedContentControl のヘッダーを取得または設定します。
HeaderFontFamily	ヘッダーのフォントファミリを取得または設定します。
HeaderFontStretch	ヘッダーのフォントストレッチを取得または設定します。
HeaderFontStyle	ヘッダーのフォントスタイルを取得または設定します。
HeaderFontWeight	ヘッダーのフォントウェイトを取得または設定します。

コンテンツ配置のプロパティ

次のプロパティを使用して、**C1HeaderedContentControl** のヘッダーおよびコンテンツパネルのコンテンツの配置をカスタマイズできます。

プロパティ	説明
HeaderPadding	ヘッダーのパディングを取得または設定します。
HeaderHorizontalContentAlignment	ヘッダーの水平方向のコンテンツ配置を取得または設定します。
HeaderVerticalContentAlignment	ヘッダーの垂直方向のコンテンツ配置を取得または設定します。
HorizontalContentAlignment	コントロールのコンテンツの水平方向の配置を取得または設定します。これは依存プロパティです。
VerticalContentAlignment	コントロールのコンテンツの垂直方向の配置を取得または設定します。これは依存プロパティです。

色のプロパティ

次のプロパティを使用して、コントロール自体に使用される色をカスタマイズできます。

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。これは依存プロパティです。
Foreground	前景色を描画するブラシを取得または設定します。これは依存プロパティです。
HeaderBackground	ヘッダーの背景ブラシを取得または設定します。
HeaderForeground	ヘッダーの前景ブラシを取得または設定します。

境界線のプロパティ

次のプロパティを使用して、コントロールの境界線をカスタマイズできます。

プロパティ	説明
BorderBrush	コントロールの境界線の背景を描画するブラシを取得または設定します。これは依存プロパティです。
BorderThickness	コントロールの境界線の太さを取得または設定します。これは依存プロパティです。

サイズのプロパティ

次のプロパティを使用して、**C1HeaderedContentControl** のサイズをカスタマイズできます。

プロパティ	説明
Height	要素の推奨高さを取得または設定します。これは依存プロパティです。
MaxHeight	要素の最大高さ制約を取得または設定します。これは依存プロパティです。
MaxWidth	要素の最大幅制約を取得または設定します。これは依存プロパティです。
MinHeight	要素の最小高さ制約を取得または設定します。これは依存プロパティです。
MinWidth	要素の最小幅制約を取得または設定します。これは依存プロパティです。
Width	要素の幅を取得または設定します。これは依存プロパティです。

タスクベースのヘルプ

タスクベースのヘルプは、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、**C1HeaderedContentControl** 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。**HeaderedContentControl for WPF/Silverlight** 製品を初めて使用される場合は、まず「HeaderedContentControl for WPF/Silverlight のクイックスタート」を参照してください。

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

また、タスクベースの各ヘルプトピックは、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。

ヘッダーバーへのコンテンツの追加

C1HeaderedContentControl のヘッダーには、単純なテキストや Silverlight コントロール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体系的に提供します。

ヘッダーバーの詳細については、「C1HeaderedContentControl のヘッダー」も参照してください。

ヘッダーバーへのテキストの追加

デフォルトでは、**C1HeaderedContentControl** のヘッダーは空になります。HeaderedContentControl のヘッダーにテキストを追加するには、Visual Studio、XAML、またはコードで Header プロパティに文字列を設定します。

設計時の場合

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

1. **C1HeaderedContentControl** を 1 回クリックして選択します。
2. [プロパティ]タブで、Header プロパティに文字列("Hello World" など)を設定します。

XAML の場合

XAML で Header プロパティを設定するには、`Header="Hello World"` を `<c1:C1HeaderedContentControl>` タグに追加します。次のようになります。

```
<c1:C1HeaderedContentControl 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. `x:Name="C1HeaderedContentControl_Ct"` を `<c1:C1HeaderedContentControl>` タグに追加します。これにより、このコントロール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。

   Visual Basic

   ```
   C1HeaderedContentControl_Ct.Header = "Hello World"
   ```

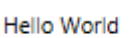
   C#

   ```
   C1HeaderedContentControl_Ct.Header = "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

C1HeaderedContentControl のヘッダーに、"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。



デフォルトでは、コンテンツパネルには文字列 "C1HeaderedContentControl" が表示されます。コンテンツパネルにさまざまなコンテンツを追加する方法については、「コンテンツパネルへのコンテンツの追加」を参照してください。

ヘッダーへのコントロールの追加

C1HeaderedContentControl のヘッダーバーは、他のコントロールを保持することができます。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。

XAML の場合

XAML で Button コントロールをヘッダーに追加するには、`<c1:C1HeaderedContentControl>` タグと `</c1:C1HeaderedContentControl>` タグの間に、次の XAML マークアップを配置します。

```
<c1:C1HeaderedContentControl.Header>
```

```
        <Button Content="Button" Height="Auto" Width="50"/>
</c1:C1HeaderedContentControl.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="C1HeaderedContentControl1"` を `<c1:C1HeaderedContentControl>` タグに追加します。これにより、このコントロール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。

   Visual Basic

   ```
   'Button コントロールを作成します。
   Dim NewButton As New Button()

   NewButton.Content = "Button"

   '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

   'ヘッダーにボタンを追加します。
   C1HeaderedContentControl1.Header = (NewButton)
   ```

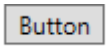
   C#

   ```
   //Button コントロールを作成します。
   Button NewButton = new Button();

   NewButton.Content = "Button";

   //ButtonコントロールのWidthおよびHeightぷrパティを設定します。
   NewButton.Width = 50;
   NewButton.Height = Double.NaN;

   //ヘッダーにコントロールを追加します。

   C1HeaderedContentControl1.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

ヘッダーバー内に 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。



コンテンツパネルへのコンテンツの追加

C1HeaderedContentControl のコンテンツパネルには、単純なテキストや WPF コントロール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体系的に提供します。

ヘッダーバーの詳細については、「C1HeaderedContentControl のコンテンツパネル」も参照してください。

コンテンツパネルへのテキストの追加

C1HeaderedContentControl のコンテンツパネルに単純なテキスト行を簡単に追加するには、Visual Studio、XAML、またはコードで **Content** プロパティに文字列を設定します。

メモ: コンテンツパネルに **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、コンテンツパネルにテキストを追加することもできます。コンテンツパネルにコントロールを追加する方法については、「コンテンツパネルへのコントロールの追加」を参照してください。

設計時の場合

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

1. **C1HeaderedContentControl** を 1 回クリックして選択します。
2. **[プロパティ]**タブで、**Content** プロパティに文字列（"Hello World" など）を設定します。
3. プログラムを実行し、**C1HeaderedContentControl** 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. `<c1:C1HeaderedContentControl>` タグに `Content="This is Content Panel"` を追加します。次のようになります。

```
<c1:C1HeaderedContentControl Content="This is Content Panel" Width="150" Height="55">
```

2. プログラムを実行し、**C1HeaderedContentControl** 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. `x:Name="C1HeaderedContentControl_Ct"` を `<c1:C1HeaderedContentControl>` タグに追加します。これにより、このコントロール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。

Visual Basic

```
C1HeaderedContentControl_Ct.Content = "This is Content Panel"
```

C#

```
C1HeaderedContentControl_Ct.Content = "This is Content Panel";
```

3. プログラムを実行し、**C1HeaderedContentControl** を展開します。

このトピックの作業結果

C1HeaderedContentControl を展開すると、"Hello World" と表示されます。このトピックの結果は、次のようになります。

Hello World
This is Content Panel

ヘッダーがないことに注意してください。これは、デフォルトでは Header プロパティが設定されないためです。ヘッダーバーにテキストやコントロールを追加する方法については、「ヘッダーバーへのコンテンツの追加」を参照してください。

コンテンツパネルへのコントロールの追加

C1HeaderedContentControl は、コンテンツパネル内に 1 つの子コントロールを保持します。このトピックでは、XAML およびコードを使用して、WPF ボタンコントロールを追加する方法について説明します。

XAML の場合

XAML で **C1HeaderedContentControl** のコンテンツパネルにボタンコントロールを追加するには、次の手順に従います。

1. <c1:C1HeaderedContentControl> タグと </c1:C1HeaderedContentControl> タグの間に、次のマークアップを配置します。

```
<Button Content="Click" Height="Auto" Width="Auto"/>
```

2. プログラムを実行し、**C1HeaderedContentControl** を展開します。

コードの場合

コードで **C1HeaderedContentControl** のコンテンツパネルにボタン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="C1HeaderedContentControl1" を <c1:C1HeaderedContentControl> 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. 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 ContentButton As New Button()
ContentButton.Content = "Click"

'Button コントロールのWidthおよびHeightプロパティを設定します。
ContentButton.Width = Double.NaN
ContentButton.Height = Double.NaN

'コンテンツパネルにボタンを追加します。
C1HeaderedContentControl1.Content = (ContentButton)
```

C#

```
//Button コントロールを作成します。
Button ContentButton = new Button();
ContentButton.Content = "Click";

//Buttonコントロールの Widthおよび Heightぷrパティを設定します。
ContentButton.Width = double.NaN;
ContentButton.Height = double.NaN;

//ヘッダーにコントロールを追加します。

C1HeaderedContentControl1.Content= (ContentButton);
```

3. プログラムを実行し、C1HeaderedContentControl を展開します。

このトピックの作業結果

C1HeaderedContentControl を展開すると、コンテンツパネルにボタンコントロールが表示されます。次の図のようになります。

ヘッダーがないことに注意してください。これは、デフォルトでは Header プロパティが設定されないためです。ヘッダーバーにテキストやコントロールを追加する方法については、「ヘッダーバーへのコンテンツの追加」を参照してください。

コンテンツパネルへの複数のコントロールの追加

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 **C1HeaderedContentControl** のコンテンツパネルに追加することで、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、**C1HeaderedContentControl** のコンテンツパネル内に各コントロールが表示されます。

XAML の場合

XAML で **C1HeaderedContentControl** のコンテンツパネルにボタンコントロールを追加するには、次の手順に従います。

1. <c1:C1HeaderedContentControl> タグと </c1:C1HeaderedContentControl> タグの間に次の XAML マークアップを配置します。

```
<c1:C1HeaderedContentControl.Content>
   <StackPanel>
     <TextBlock Text="1st TextBlock"/>
     <TextBlock Text="2nd TextBlock"/>
     <TextBlock Text="3rd TextBlock"/>
   </StackPanel>
</c1:C1HeaderedContentControl.Content>
```

2. プログラムを実行します。
3. **C1HeaderedContentControl** を展開し、3 つの **TextBlock** コントロールがコンテンツパネルに表示されることを確認します。

コードの場合

コンテンツパネル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. 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

```
'スタックパネルを作成してC1HeaderContentControlに追加します。
Dim StackPanel1 As New StackPanel()
C1HeaderedContentControl_CP.Content = (StackPanel1)

'三つのTextBlockコントロールを作成してそれぞれのTextプロパティを設定します。

Dim TextBlock1 As New TextBlock()

Dim TextBlock2 As New TextBlock()

Dim TextBlock3 As New TextBlock()

TextBlock1.Text = "1st TextBlock"

TextBlock2.Text = "2nd TextBlock"

TextBlock3.Text = "3rd TextBlock"

'StackPanelにTextBloackコントロールを追加します。

StackPanel1.Children.Add(TextBlock1)

StackPanel1.Children.Add(TextBlock2)

StackPanel1.Children.Add(TextBlock3)
```

C#

```
//スタックパネルを作成してC1HeaderContentControlに追加します。

StackPanel StackPanel1 = new StackPanel();
C1HeaderedContentControl_CP.Content = (StackPanel1);

//三つのTextBlockコントロールを作成してそれぞれのTextプロパティを設定します。

TextBlock TextBlock1 = new TextBlock();

TextBlock TextBlock2 = new TextBlock();

TextBlock TextBlock3 = new TextBlock();
```

```
TextBlock1.Text = "1st TextBlock";

TextBlock2.Text = "2nd TextBlock";

TextBlock3.Text = "3rd TextBlock";

//StackPanelにTextBloackコントロールを追加します。

StackPanel1.Children.Add(TextBlock1);

StackPanel1.Children.Add(TextBlock2);

StackPanel1.Children.Add(TextBlock3);
```

2. プログラムを実行します。
3. **C1HeaderedContentControl** を展開し、3 つの **TextBlock** コントロールがコンテンツパネルに表示されることを確認します。

このトピックの作業結果

C1HeaderedContentControl を展開すると、3 つの **TextBlock** コントロールが次のようにコンテンツパネルに表示されます。

1st TextBlock
2nd TextBlock
3rd TextBlock

HyperPanel

HyperPanel for WPF//Silverlightは、マウスの近くにある項目に自動ズーム効果を提供する StackPanel です。このパネルに任意の要素を配置して、カルーセル風の効果を出したり、スクロールバーを使用しなくても比較的小さなコンテナに多数の要素を表示することができます。生成される効果は、Mac OS X のシステムツールバー（Dock）に似ています。

- **動的ズーム**

  HyperPanel for WPF/Silverlight を使用すると、狭い領域に多数の項目を表示できます。マウスから遠くにある項目は小さく表示され、あまり場所をとりません。

- **ズーム効果の制御**

  パネル上でマウスを移動したときに適用されるズームの量を決定します。**Distribution** プロパティを使用して、ズーム効果の強さを制御します。

- **ズーム効果の制限**

  MinElementScale プロパティを使用して、項目が小さくなりすぎないようにします。

- **項目の不透明度の制御**

  ApplyOpacity プロパティを使用して、マウスの近くにある要素を不透明にします。マウスから遠くにある項目ほど透明になるため、その距離感が伝わります。

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

クイックスタート

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 **HyperPanel for WPF/Silverlight** を使用してアプリケーションを作成します。

プロジェクトを設定するには、次の手順に従います。

1. Visual Studio で、[**ファイル**]→[**新規作成**]→[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 アプリケーション]または[Silverlight アプリケーション]を選択します。プロジェクトの名前(たとえば、「QuickStart」)を入力し、[OK]をクリックします。プロジェクトが作成され、MainWindow.xaml ファイルが開きます。
3. 参照の追加ダイアログボックスで、以下のアセンブリを見つけて選択し、[OK]をクリックしてプロジェクトに参照を追加します。
   - **WPF**: C1.WPF.4.dll
   - **Silverlight**: C1.Silverlight.5.dll
4. ツールボックスに移動し、[**C1HyperPanel**]アイコンをダブルクリックして、プロジェクトにパネルを追加します。
5. **x:Name="c1hp1"** を **<c1:C1HyperPanel>** タグに追加して、コントロールに名前を付けます。次のようになります。

```
XAML
<c1:C1HyperPanel x:Name="c1hp1"></c1:C1HyperPanel>
```

それに一意の識別子を付けると、コードでそのコントロールにアクセスできるようになります。

9. **<c1:C1HyperPanel>** タグに **Height="Auto" Width="Auto"** を追加して、コントロールをサイズ変更します。次のようになります。

```
XAML
<c1:C1HyperPanel x:Name="c1hp1" Height="Auto" Width="Auto"></c1:C1HyperPanel>
```

手順 2:コンテンツの追加

前の手順では、新しい WPF/Silverlightプロジェクトを作成し、アプリケーションに1つの **C1HyperPanel** パネルを追加しました。この手順では、引き続き、パネルに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、**<c1:C1HyperPanel>** タグと **</c1:C1HyperPanel>** タグの間にカーソルを置きます。これで、項目を追加すると、その項目がパネルに追加されます。
2. ツールボックスに移動し、[**ContentControl**]アイコンをダブルクリックして、**C1HyperPanel** にコントロールを追加します。Grid や Canvas などの他のパネルと同様に、**C1HyperPanel** パネルに項目を追加するために必要な作業はこれだけです。
3. XAML ビューに切り替えて、ContentControl のマークアップを更新します。次のようになります。

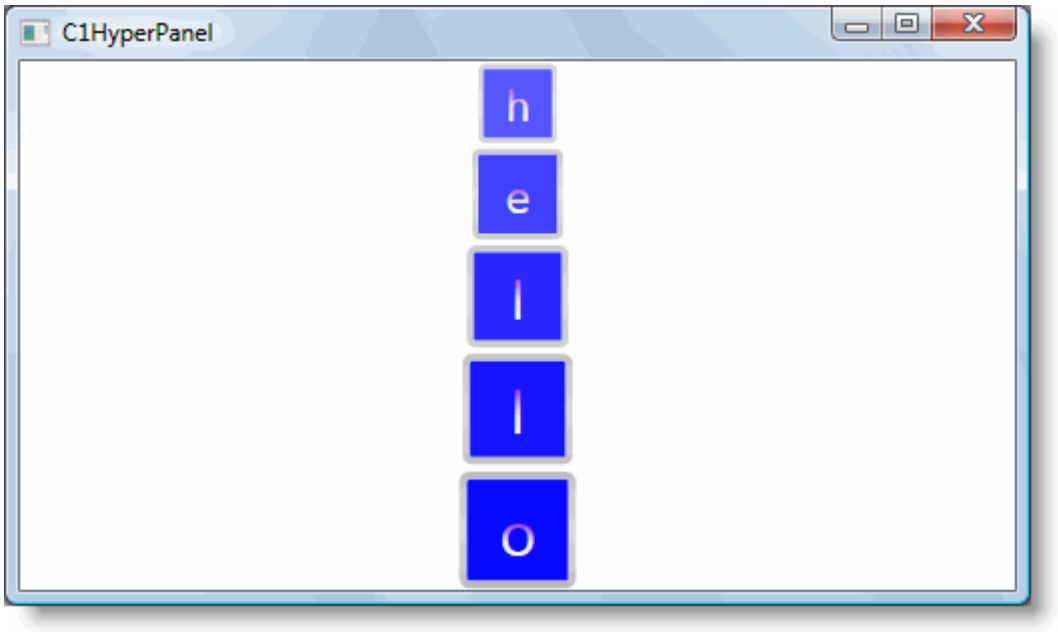
```
XAML
<ContentControl Content="h" ContentTemplate="{StaticResource letterTemplate}"/>
```

4. C1HyperPanel 内の ContentControl のマークアップの下に次の XAML を入力して、パネルにコントロールを追加します。

```
XAML
<ContentControl Content="e" ContentTemplate="{StaticResource letterTemplate}"/>
```

```
<ContentControl Content="l" ContentTemplate="{StaticResource letterTemplate}"/>
<ContentControl Content="l" ContentTemplate="{StaticResource letterTemplate}"/>
<ContentControl Content="o" ContentTemplate="{StaticResource letterTemplate}"/>
<ContentControl Content=" " ContentTemplate="{StaticResource letterTemplate}"/>
<ContentControl Content="w" ContentTemplate="{StaticResource letterTemplate}"/>
<ContentControl Content="o" ContentTemplate="{StaticResource letterTemplate}"/>
<ContentControl Content="r" ContentTemplate="{StaticResource letterTemplate}"/>
<ContentControl Content="l" ContentTemplate="{StaticResource letterTemplate}"/>
<ContentControl Content="d" ContentTemplate="{StaticResource letterTemplate}"/>
<ContentControl Content="!" ContentTemplate="{StaticResource letterTemplate}"/>
```

5. [**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。次のように表示されます.



これで、WPF/Silverlight アプリケーションが正しく作成され、アプリケーションに **C1HyperPanel** といくつかのコントロールが追加されました。次の手順では、**C1HyperPanel** をカスタマイズします。

手順 3：コントロールの設定

これまでの手順では、新しい WPF/Silverlight プロジェクトを作成し、アプリケーションに1つの **C1HyperPanel** パネルといくつかの **ContentControl** を追加しました。この手順では、引き続き、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。

次の手順に従います。

1. XAML ビューで、**<c1:C1HyperPanel>** タグに **Orientation="Horizontal"** を追加して、**C1HyperPanel** コントロールの方向を設定します。次のようになります。

XAML

```
<c1:C1HyperPanel x:Name="c1hp1" Height="Auto" Width="Auto"
Orientation="Horizontal">
```

Orientation プロパティは、パネル内の項目を水平方向に表示するか、垂直方向に表示するかを決定します。デフォルトでは、**Orientation** は **Vertical** に設定されており、パネルのコンテンツは垂直方向に表示されます。**Orientation** プロパティを **Horizontal** に設定すると、コンテンツが水平方向に表示されます。

2. **<c1:C1HyperPanel>** タグに **Distribution="0.2"** を追加して、**C1HyperPanel** コントロールのスケール差を設定します。次のようになります。

```
XAML
<c1:C1HyperPanel x:Name="c1hp1" Height="Auto" Width="Auto"
Orientation="Horizontal" Distribution="0.2">
```

Distribution プロパティは 0.1～1.0 の間の数値で、パネルの中心近くの要素をどの程度ズームするかを制御します。値を小さくすると、ズーム効果が大きくなります。デフォルトでは、このプロパティは 0.5 に設定されています。**Distribution** を「0.02」に設定すると、中心にある要素はパネルの端にある要素よりかなり大きくズームインされます。

3. **<c1:C1HyperPanel>**タグに **MinElementScale="0.5"** を追加して、**C1HyperPanel** コントロールの最小スケールを設定します。次のようになります。

```
XAML
<c1:C1HyperPanel x:Name="c1hp1" Height="Auto" Width="Auto"
Orientation="Horizontal" Distribution="0.2" MinElementScale="0.5">
```

MinElementScale プロパティは0～1.0 の間の数値で、パネルの端近くの要素を中心近くの要素に比較してどの程度小さく表示するかを決定します。デフォルトでは、このプロパティは0に設定されています。**MinElementScale** を設定することで、パネルの端近くの要素を最小でも元のサイズの半分に表示できます。

4. **<c1:C1HyperPanel>**タグに **Center="0.1"** を追加して、**C1HyperPanel** コントロールの中心を設定します。次のようになります。

```
XAML
<c1:C1HyperPanel x:Name="c1hp1" Height="Auto" Width="Auto"
Orientation="Horizontal" Distribution="0.2" MinElementScale="0.5" Center="0.1">
```

Center プロパティは0～1.0 の間の数値で、アプリケーションを実行したときに、最初にパネルの中心（最もズームインされる要素）をどこに置くかを設定します。デフォルトでは、このプロパティは「0.5」に設定されており、中心は中央になります。**Center** を設定することで、最初にズームされる要素はコントロールの左側に移動します。この値は、実行時にパネル上でマウスを移動すると、自動的に更新されます.

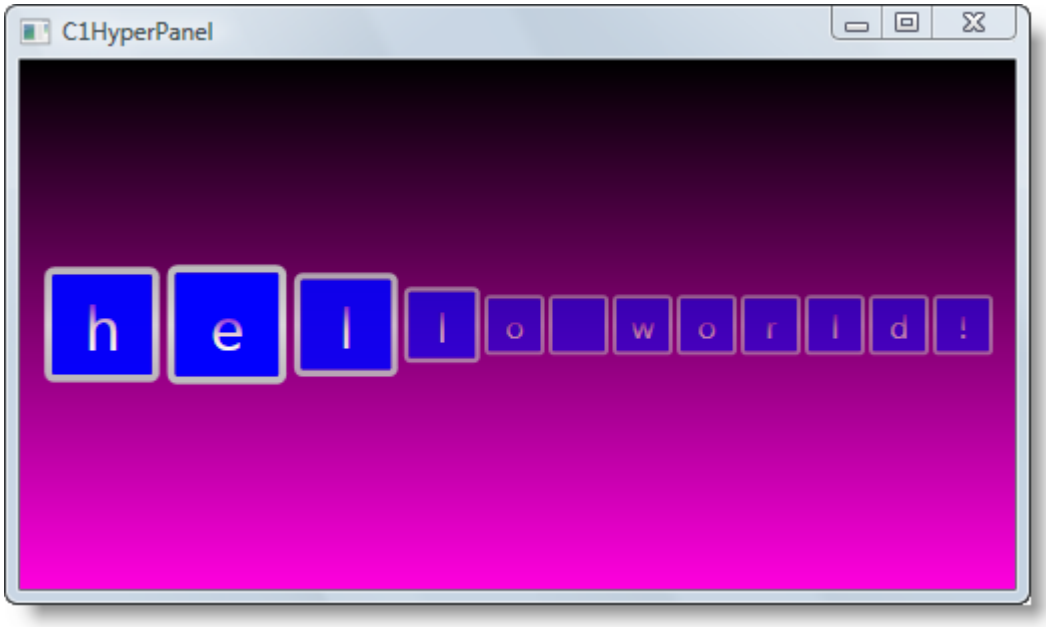
5. **<C1HyperPanel>** タグの直後でコンテンツ要素の前に、次のマークアップを追加します。

```
XAML
<c1:C1HyperPanel.Background>
    <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
        <GradientStop Color="#FF000000" Offset="0"/>
        <GradientStop Color="#FFFF00DF" Offset="1"/>
    </LinearGradientBrush>
</c1:C1HyperPanel.Background>
```

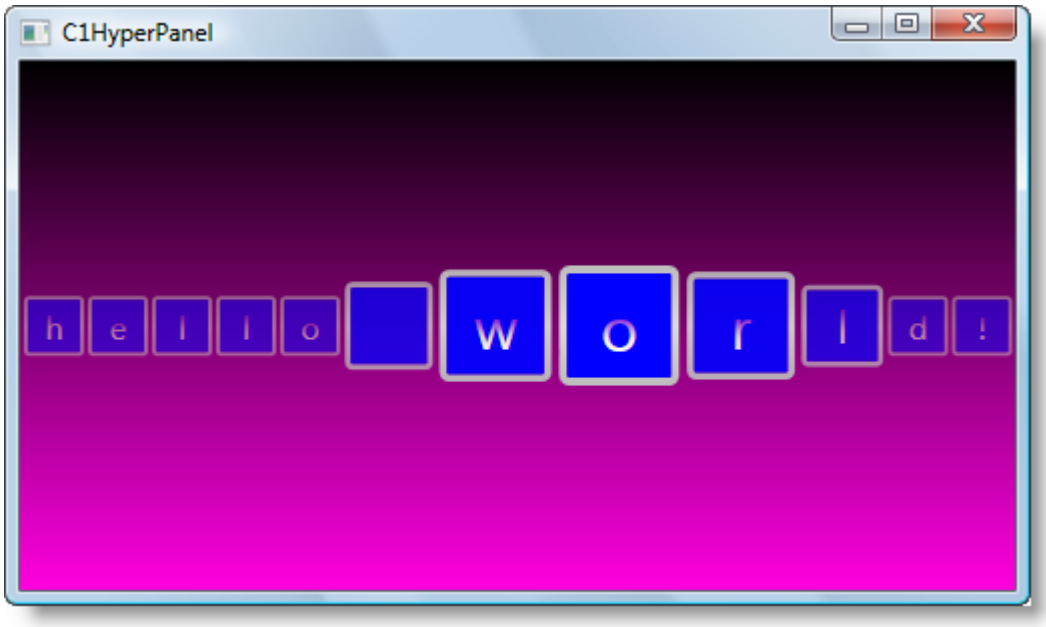
このマークアップは、**C1HyperPanel** にグラデーションの背景を追加します。

6. [**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、初めに次の図のように表示されます。



7. パネル内でマウスを移動すると、コンテンツの **Center** が変化することがわかります。

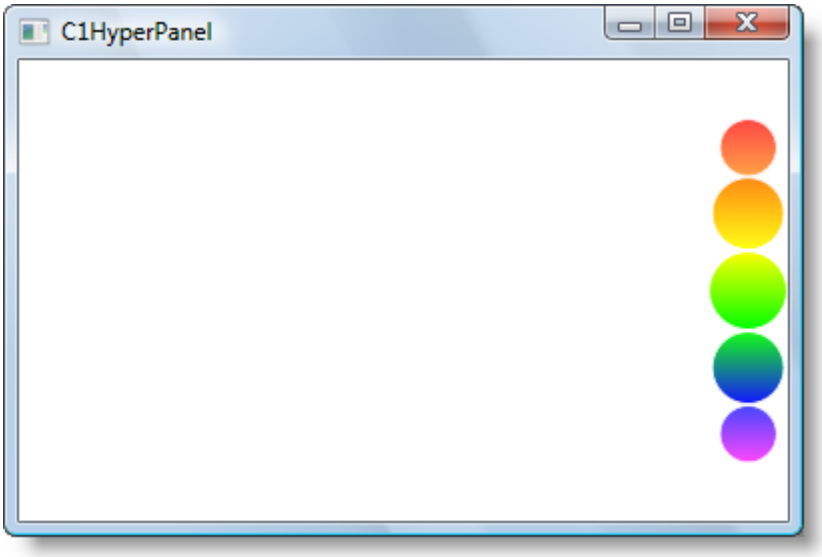


HyperPanel の機能

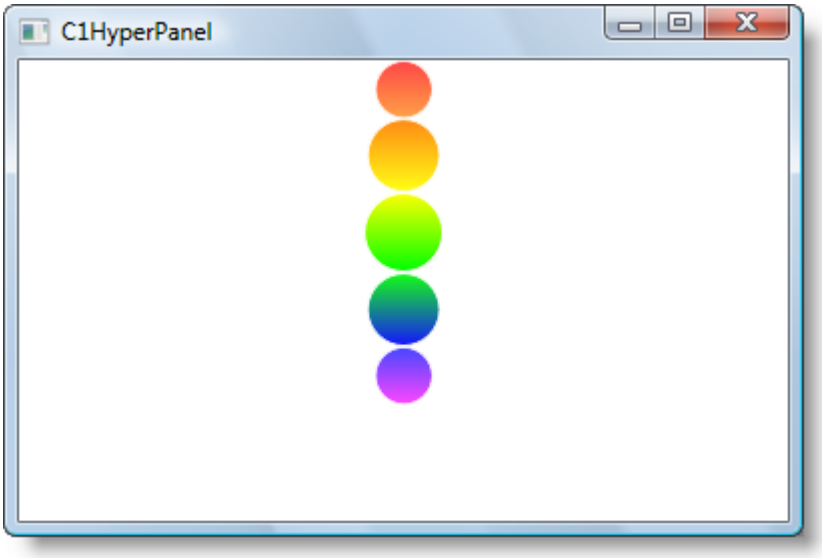
配置

HorizontalAlignment プロパティと **VerticalAlignment** プロパティは、**C1HyperPanel** パネルを含むパネルまたはウィンドウ内で、そのパネルをどのように配置するかを制御します。デフォルトでは、どちらのプロパティも **Stretch** に設定されており、有効な領域いっぱいまでパネルが拡大されます。

HorizontalAlignment のオプションは、Left、Center、Right、および Stretch です。たとえば、**HorizontalAlignment** を Right に設定した場合、パネルは、次の図のように有効な領域の右側に表示されます。



VerticalAlignment のオプションは、Top、Center、Bottom、および Stretch です。たとえば、**VerticalAlignment** を Top に設定した場合、パネルは、次の図のように有効な領域の上側に表示されます。



コンテンツ

デフォルトでは、**C1HyperPanel** パネルは空で表示され、コンテンツがありません。

パネルにコントロールなどのコンテンツを追加する作業は、Canvas や Grid などの他のパネルにコンテンツを追加する場合と同様に簡単です。次の手順では、**C1HyperPanel** にボタンを追加します。

XAML の場合

C1HyperPanel にボタンを追加するには、**<c1:C1HyperPanel>** タグの後に **<Button>** タグを追加します。次のようになります。

XAML

```
<c1:C1HyperPanel Name="C1HyperPanel1">
    <Button Height="50" Name="button1" Width="50"></Button>
</c1:C1HyperPanel>
```

コードの場合

C1HyperPanel にボタンを追加するには、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、Window_Loaded イベントハンドラを追加してから、次のようにコードを追加します。

Visual Basic

```
Private Private Sub UserControl_Loaded(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs)
    Dim button1 = New Button
    button1.Height = 50
    button1.Width = 50
    Me.C1HyperPanel1.Children.Add(button1)
End Sub
```

または

```
Private Sub Window1_Loaded(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles MyBase.Loaded
    Dim button1 = New Button
    button1.Height = 50
    button1.Width = 50
    Me.C1HyperPanel1.Children.Add(button1)
End Sub
```

C#

```
private void UserControl_Loaded(object sender, RoutedEventArgs e)
{
    Button button1 = new Button();
    button1.Height = 50;
    button1.Width = 50;
    this.c1HyperPanel1.Children.Add(button1);
}
```

または

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    Button button1 = new Button();
    button1.Height = 50;
    button1.Width = 50;
    this.c1HyperPanel1.Children.Add(button1);
}
```

設計時

設計時に **C1HyperPanel** にボタンを追加するには、次の手順に従います。

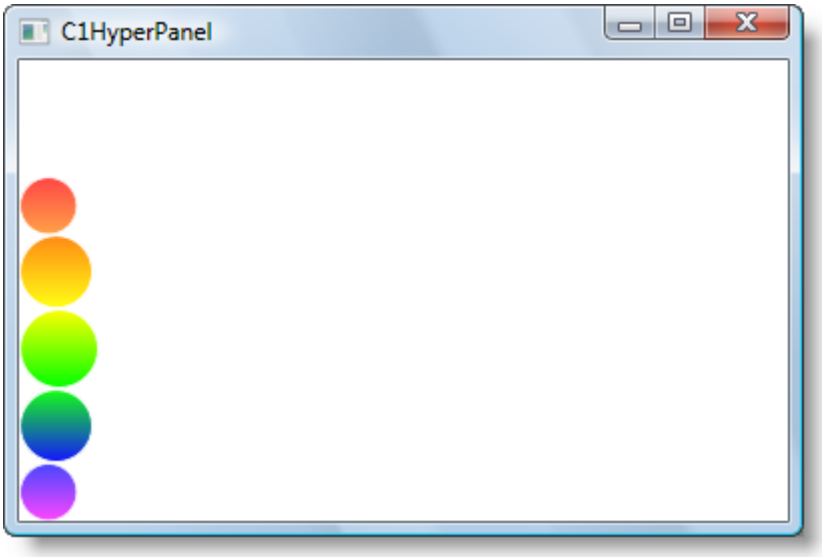
1. **C1HyperPanel** をクリックして選択します。
2. Visual Studio のツールボックスに移動し、**Button** コントロールをダブルクリックします。コントロールがパネルに追加されます。

コンテンツの配置

HorizontalContentAlignment プロパティと **VerticalContentAlignment** プロパティは、**C1HyperPanel** パネル内のコンテンツをどのように配置するかを制御します。デフォルトでは、どちらのプロパティも Center に設定されており、コンテンツがパ

ネル内の中央に配置されます。

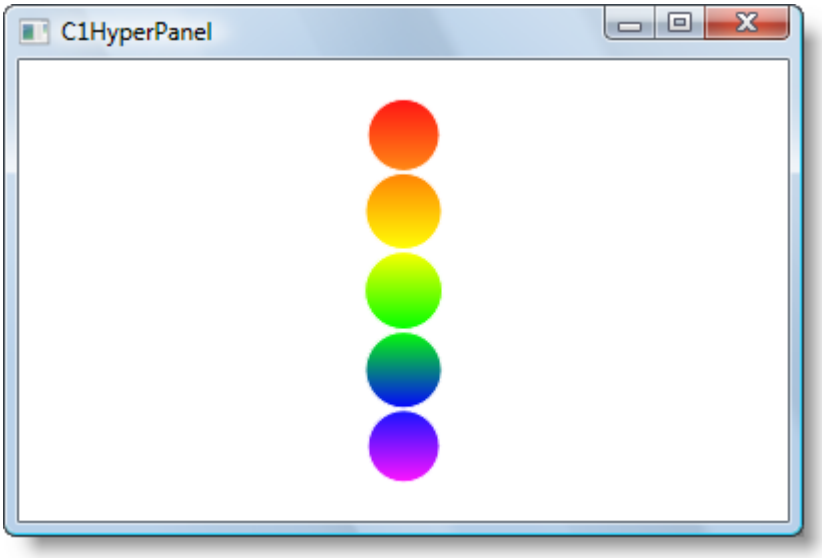
HorizontalContentAlignment のオプションは、Left、Center、Right、および Stretch です。**VerticalContentAlignment** のオプションは、Top、Center、Bottom、および Stretch です。たとえば、**HorizontalContentAlignment** を Left、**VerticalContentAlignment** を Bottom に設定した場合、次の図のように、コンテンツはパネルの左下隅に表示されます。



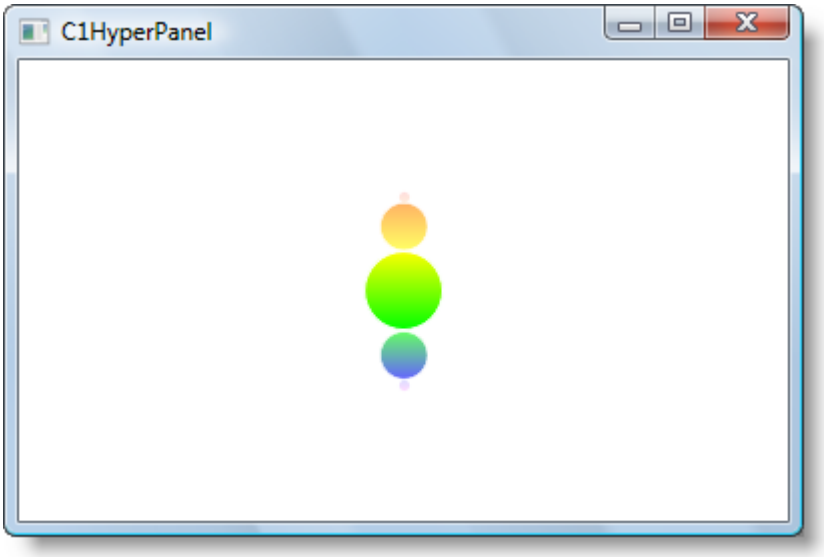
ディストリビューション

Distribution プロパティは、パネルの中心近くにある要素に適用するズームの量を制御する 0.1～1.0 の間の値を取得または設定します。この値を小さくすると、パネルの中心から遠くにある項目が離れて表示されます。デフォルトでは、**Distribution** は「0.5」に設定されています。

Distribution を「1」に設定すると、次の図のように、すべての要素が同じズームレベルで表示されます。



Distribution を「0.2」などの小さな値に設定すると、中心から遠くにある要素ほど遠くにズームアウトされて表示されます。



XAML の場合

XAML で、中心から遠くにある項目が大きくズームアウトされるように **Distribution** を設定するには、**<c1:C1HyperPanel>** タグに **Distribution="0.2"** を追加します。次のようになります。

XAML

```
<c1:C1HyperPanel Name="C1HyperPanel1" Distribution="0.2">
```

コードの場合

コードで、中心から遠くにある項目が大きくズームアウトされるように **Distribution** を設定するには、プロジェクトに次のコードを追加します。

VisualBasic

```
Me.C1HyperPanel1.Distribution = "0.2"
```

C#

```
this.c1HyperPanel1.Distribution = "0.2";
```

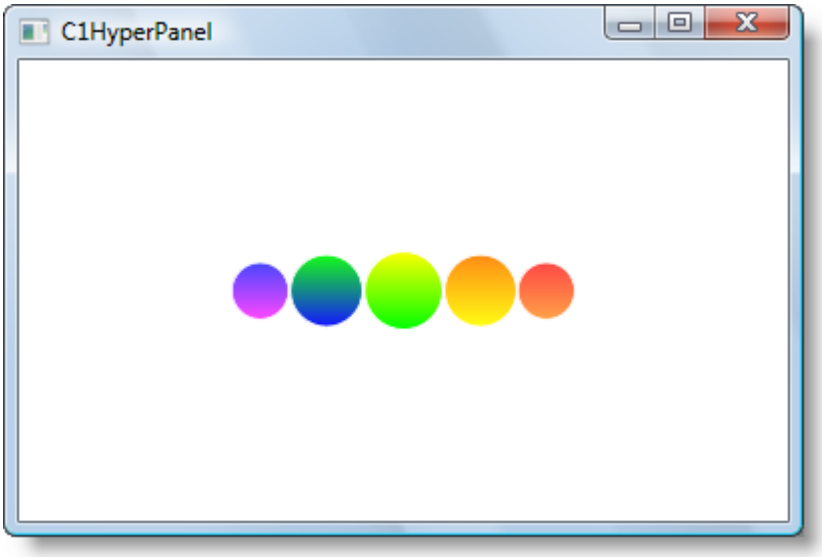
設計時

設計時に、中心から遠くにある項目が大きくズームアウトされるように **Distribution** を設定するには、次の手順に従います。

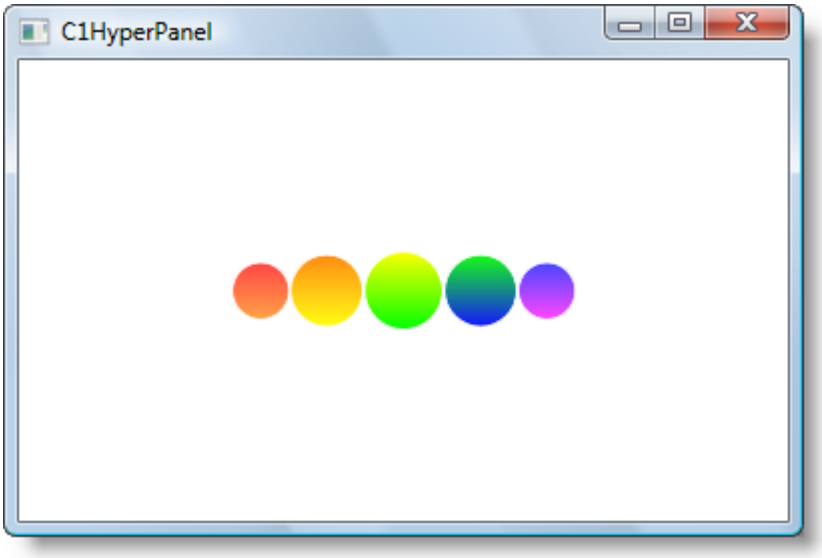
1. **C1HyperPanel** をクリックして選択します。
2. [プロパティ]ウィンドウに移動し、**Distribution** プロパティを見つけます。
3. **Distribution** プロパティの横にあるテキストボックスをクリックし、「0.2」と入力します。

配置方向

Orientation プロパティは、**C1HyperPanel** パネル内でコンテンツをどのようにレイアウトするかを決定します。デフォルトでは、**Orientation** は **Vertical** に設定されており、パネル内でコンテンツが上から下に垂直に並んで表示されます。



Orientation を **Horizontal** に設定すると、パネル内でコンテンツが左から右に水平に並んで表示されます。



XAML の場合

XAML でコンテンツを水平方向に並べて表示するように **Orientation** を設定するには、**<c1:C1HyperPanel>**タグに **Orientation="Horizontal"** を追加します。次のようになります。

XAML

```
<c1:C1HyperPanel Name="C1HyperPanel1" Orientation="Horizontal">
```

コードの場合

コードでコンテンツを水平方向に並べて表示するように **Orientation** を設定するには、プロジェクトに次のコードを追加します。

VisualBasic

```
Me.C1HyperPanel1.Orientation = Orientation.Horizontal
```

C#

```
this.c1HyperPanel1.Orientation = Orientation.Horizontal;
```

設計時

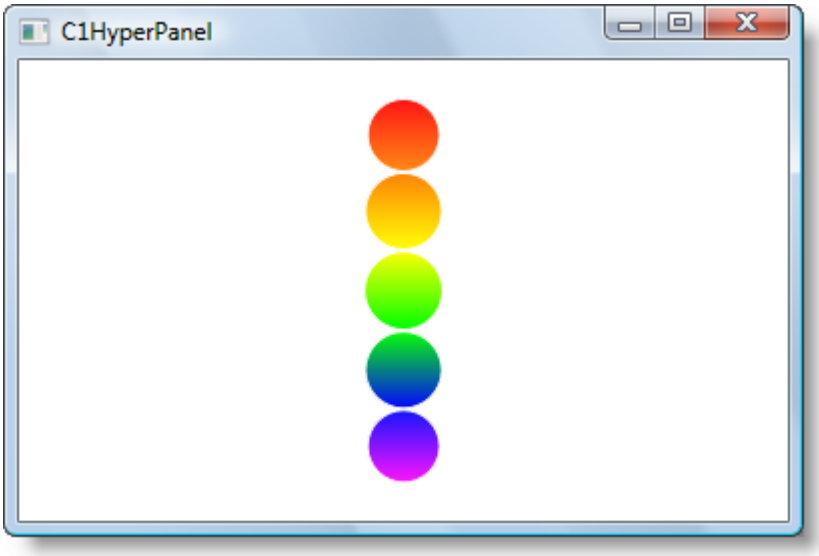
設計時に、コンテンツを水平方向に並べて表示するように **Orientation** を設定するには、次の手順に従います。

1. **C1HyperPanel** 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、**Orientation** プロパティを見つけます。
3. **Orientation** プロパティの横にあるドロップダウン矢印を選択し、**Horizontal** を選択します。

スケール

MinElementScale プロパティは、中心から遠くにある要素に適用する最小スケールを決定する値を取得または設定します。**MinElementScale** を「0.1」に設定すると、中心から最も遠くにある項目が最小でも元のサイズの 10% で表示されます。

デフォルトでは、**MinElementScale** は「0」に設定されており、パネルの中心から最も遠くにある項目は完全にズームアウトします。値を大きくするほど、中心から遠くにある要素がズームインされます。たとえば、**MinElementScale** を「1」に設定すると、次の図のように、すべての項目が同じ距離で表示され、要素は元のサイズの 100% で表示されます。パネル内の要素の上でマウスを移動しても、要素はまったくズームインやズームアウトされず、静止して表示されます。



XAML の場合

XAML

```
<c1:C1HyperPanel Name="C1HyperPanel1" MinElementScale="1">
```

コードの場合

コードですべての **C1HyperPanel** 要素を同じサイズに設定するには、プロジェクトに次のコードを追加します。

VisualBasic

```
Me.C1HyperPanel1.MinElementScale = "1"
```

C#

```
this.c1HyperPanel1.MinElementScale = "1";
```

設計時

設計時に、すべての **C1HyperPanel** 要素を同じサイズに設定するには、次の手順に従います。

1. **C1HyperPanel** 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、**MinElementScale** プロパティを見つけます。
3. **MinElementScale** プロパティの横にあるテキストボックスをクリックし、「1」と入力します。

LayoutPanels (Silverlight のみ)

Layout Panels for Silverlight を使って Silverlight アプリケーションのコンテンツのフローと配置を制御します。

これらのコントロールは、Silverlightでのみ用意されています。

C1WrapPanel を使用すると、コンテンツを垂直または水平方向に折り返すことができます。

- 水平/垂直方向のフローレイアウト

  パネルの寸法を設定して、レイアウトの方向を決定できます。これにより、子要素を水平または垂直方向に連続的に配置します。

C1DockPanel を使用すると、パネルの端にコンテンツをドッキングできます。

- **ドッキング要素**

  C1DockPanel for Silverlight コントロールを使用すると、要素を上下左右にドッキングできます。

- **ドッキングパネルのその他の要素**

  子要素は、XAML で宣言された順序でドッキングパネルに配置されます。

C1UniformGrid を使用すると、コンテンツをグリッドに表示できます。

- **列と行の表示/非表示**

  C1UniformGrid for silverlight コントロールを使用すると、列または行全体を表示または非表示にできます。

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

クイックスタートは、**Layout Panels for Silverlight** の各コントロールについて1つずつ作成されています。各クイックスタートでは、最初に Silverlight アプリケーションを作成し、次にコントロールのスタイルを追加します。**WrapPanel** では、複数の HyperlinkButtons を折り返します。**DockPanel** では、複数の要素を持つパネルを作成し、C1DockPanel の4つのドッキングオプションを示します。UniformGrid では、3つの列を持つグリッドが作成され、最初の行に2つの空のセルが作成されます。

クイックスタートを開始するには、次のいずれかを選択します。

- WrapPanel クイックスタート

WrapPanel クイックスタート

このクイックスタートは、**WrapPanel 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、Visual Studio で新しいプロジェクトを作成し、スタイルが設定された折り返し可能な **HyperlinkButtons** を追加して、ボタンの並ぶ方向を変更します。

ステップ 1

プロジェクトを設定するには、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で、左ペインから言語を選択し(この例では C# を使用)、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。
3. プロジェクト名に「C1WrapPanel」と入力し、[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。
4. 必要に応じて、[Silverlight アプリケーションを新しい Web サイトでホストする]ボックスをオフにし、[OK]をクリックします。MainPage.xaml ファイルが開きます。

次のステップでは、複数の HyperlinkButtons ボタンを追加し、スタイルを設定し、折り返します。

ステップ 2

シンプルな **HyperlinkButtons** ボタンを使用して、コンテンツを垂直または水平方向に折り返す方法について説明します。これは Web アプリケーションで頻繁に使用される **TagCloud** ビューを作成する一般的なシナリオです。

1. 最初に、プロジェクトから Grid タグを取り除きます。
2. Visual Studio のツールボックスで、**C1WrapPanel** コントロールをページにドラッグします。
3. **<c1:C1WrapPanel>** タグの間にカーソルを置き、次の XAML を使って **HyperlinkButtons** を追加します。

XAML

```
  <HyperlinkButton Content="サンプルテキスト" FontSize="25" />
      <HyperlinkButton Content="テキストを折り返すための長い文字列" />
      <HyperlinkButton Content="改行を追加しましょう" />
      <HyperlinkButton c1:C1WrapPanel.BreakLine="After" Content="後ろに改行を追加します" />
      <HyperlinkButton Content="C1WrapPanel" />
      <HyperlinkButton Content="垂直方向の折り返し" />
      <HyperlinkButton Content="水平方向の折り返し" FontSize="20"  />
      <HyperlinkButton c1:C1WrapPanel.BreakLine="Before" Content="前に改行を追加します" />
      <HyperlinkButton Content="コントロール" FontSize="8" />
      <HyperlinkButton Content="Silverlight" />
      <HyperlinkButton Content="コンポーネント" FontSize="18" />
      <HyperlinkButton c1:C1WrapPanel.BreakLine="AfterAndBefore" Content="前後に改行を追加します" />
      <HyperlinkButton Content="垂直または水平方向にコンテンツを折り返すフローレイアウト" />
```

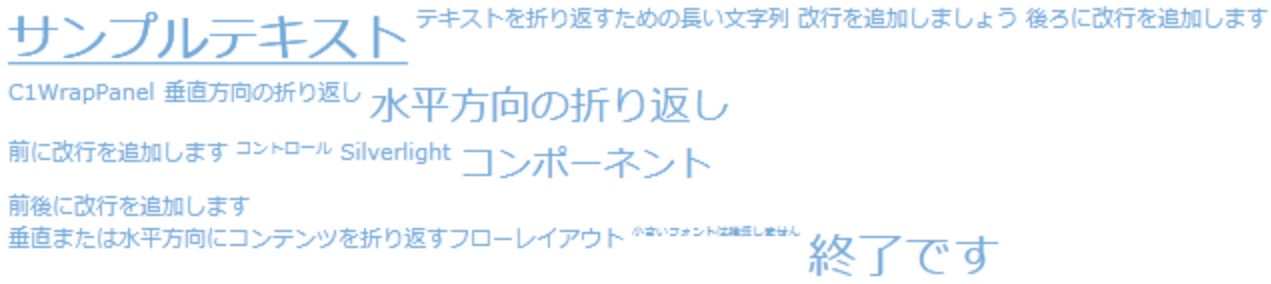
```
        <HyperlinkButton Content="小さいフォントは推奨しません" FontSize="6" />
        <HyperlinkButton Content="終了です" FontSize="24" />
```

次の手順では、このアプリケーションを実行します。

ステップ 3

これで、アプリケーションを実行する準備ができました。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]を選択します。アプリケーションは次のように表示されます。



2. [デバッグの停止]ボタンをクリックしてアプリケーションを終了します。
3. MainPage.xaml に戻ります。<c1:C1WrapPanel> タグで、Orientation プロパティを "Vertical" に設定します。XAML は次のようになります。

```
XAML
<c1:C1WrapPanel Orientation="Vertical">
```

4. [デバッグ]メニューで再度[デバッグ開始]をクリックします。アプリケーションは次のようになります。

ボタンが垂直方向に積み重なっています。

おめでとうございます。これで、**WrapPanel for Silverlight** クイックスタートは終了です。

DockPanel クイックスタート

このクイックスタートは、**DockPanel 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、Visual Studio で新しいプロジェクトを作成し、上下左右にドッキングする要素を追加し、さらに **C1DockPanel** を塗りつぶします。

ステップ 1

プロジェクトを設定するには、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で、左ペインから言語を選択し(この例では C# を使用)、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。

3. プロジェクト名に「C1DockPanel」と入力し、[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。
4. 必要に応じて、[Silverlight アプリケーションを新しい Web サイトでホストする]ボックスをオフにし、[OK]をクリックします。MainPage.xaml ファイルが開きます。

次の手順では、**C1DockPanels** を追加してカスタマイズします。

ステップ 2

この手順では、複数の **C1DockPanels** を追加してスタイルを設定します。

1. 最初に、プロジェクトから Grid タグを取り除きます。
2. Visual Studio のツールボックスで、**C1DockPanel** コントロールをページにドラッグします。
3. **<c1:C1DockPanel>** タグの間にカーソルを置き、次の XAML を入力して **C1DockPanels** を左、右、上、下に追加し、画面の中心を塗りつぶします。

XAML

```
<Border c1:C1DockPanel.Dock="Top" Height="50" Background="Red">
   <TextBlock Text="Top" />
</Border>
<Border c1:C1DockPanel.Dock="Bottom" Height="50" Background="Blue">
   <TextBlock Text="Bottom" />
</Border>
<Border c1:C1DockPanel.Dock="Right" Width="50" Background="Yellow">
   <TextBlock Text="Right" />
</Border>
<Border c1:C1DockPanel.Dock="Left" Background="Green" Width="50" >
   <TextBlock Text="Left" />
</Border>
<Border Background="White" >
   <TextBlock Text="Fill" />
</Border>
```

次の手順では、このアプリケーションを実行します。

ステップ 3

これで、アプリケーションを実行する準備ができました。[**デバッグ**]メニューから[**デバッグ開始**]を選択します。アプリケーションは次のようになります。上、右、左、下の4つの **C1DockPanels** があり、中心のブロックは塗りつぶされています。

おめでとうございます。これで、**DockPanel for Silverlight** クイックスタートは終了です。

UniformGrid クイックスタート

このクイックスタートは、**UniformGrid 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、Visual Studio で新しいプロジェクトを作成し、**C1UniformGrid** をアプリケーションに追加し、**Columns**、**FirstColumn**、**Width** の各プロパティを設定してから、アプリケーションを実行します。

ステップ 1

プロジェクトを設定するには、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で、左ペインから言語を選択し(この例では C# を使用)、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。
3. プロジェクト名に「C1UniformGrid」と入力し、[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。
4. 必要に応じて、[Silverlight アプリケーションを新しい Web サイトでホストする]ボックスをオフにし、[OK]をクリックします。MainPage.xaml ファイルが開きます。

次のステップでは、複数の HyperlinkButtons ボタンを追加し、スタイルを設定し、折り返します

ステップ 2

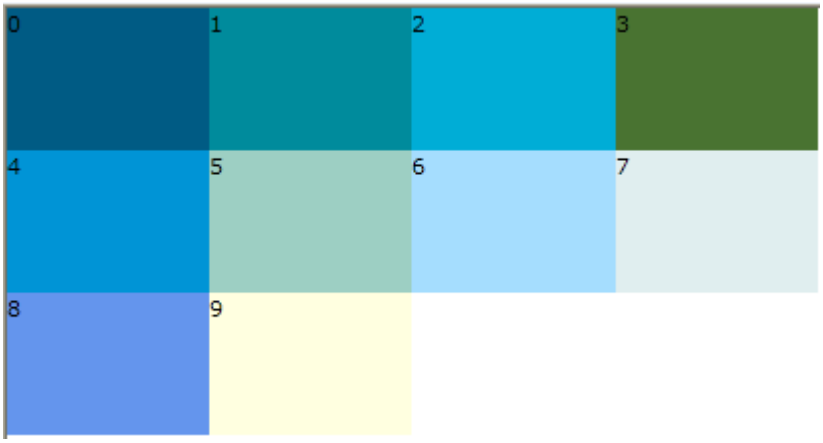
1. 最初に、プロジェクトから Grid タグを取り除きます
2. Visual Studio のツールボックスで、**C1UniformGrid** コントロールをページにドラッグします。
3. **<c1:C1UniformGrid>**タグの間にカーソルを置き、次の XAML を入力してグリッドの子要素(この場合は番号の付いたセル)を追加します。

XAML

```
<Border Background="#FF005B84" >
    <TextBlock Text="0" />
</Border>
```

```
<Border Background="#FF008B9C" >
    <TextBlock Text="1" />
</Border>
<Border Background="#FF00ADD6" >
    <TextBlock Text="2" />
</Border>
<Border Background="#FF497331" >
    <TextBlock Text="3" />
</Border>
<Border Background="#FF0094D6" >
    <TextBlock Text="4" />
</Border>
<Border Background="#FF9DCFC3" >
    <TextBlock Text="5" />
</Border>
<Border Background="#FFA5DDFE" >
    <TextBlock Text="6" />
</Border>
<Border Background="#FFE0EEEF" >
    <TextBlock Text="7" />
</Border>
<Border Background="CornflowerBlue" >
    <TextBlock Text="8" />
</Border>
<Border Background="LightYellow" >
    <TextBlock Text="9" />
</Border>
```

この時点でアプリケーションを実行すると、次の図のようになります。



ステップ 3

この手順では、**Columns**、**FirstColumn**、および **Width** プロパティのセットを設定します。**Columns** は、グリッド内の列の数を設定し、**Width** プロパティは幅をピクセル単位で設定し、**FirstColumn** プロパティは最初の行に表示する空のセルの数を設定します。

1. 開始側の **<c1:C1UniformGrid>** グリッドタグの中にカーソルを置きます。
2. 次の XAML を使って **Columns**、**FirstColumn**、**Width** の各プロパティを設定します。

```
XAML
```

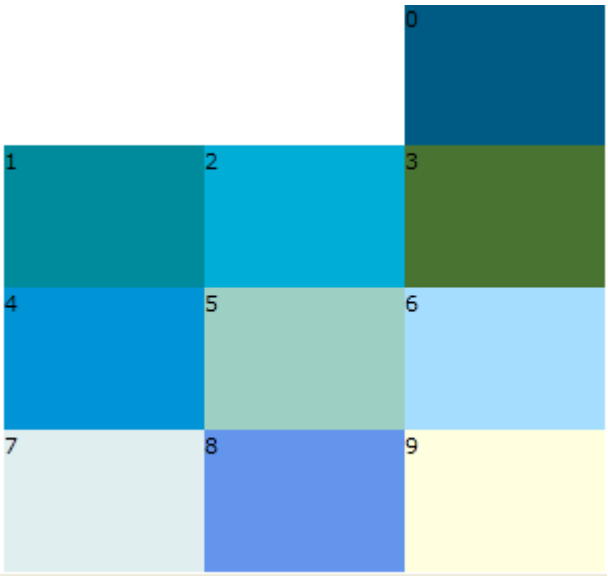
```
<c1:C1UniformGrid Columns="3" FirstColumn="2" Width="300">
```

FirstColumn プロパティによる指定に基づいて、グリッドの最初の行に空の列が2つできます。

次の手順では、アプリケーションを実行して結果を見ます。

ステップ 4

これで、アプリケーションを実行する準備ができました。[**デバッグ**]メニューから[**デバッグ開始**]を選択します。アプリケーションは次のように表示されます。



最初の行に空のセルが2つあります。

おめでとうございます。これで、**UniformGrid for Silverlight** クイックスタートは終了です。

レイアウトパネルの機能

項目を折り返す

C1WrapPanel.BreakLine Attached プロパティを使用する方法

C1WrapPanel.BreakLine Attached プロパティを使用すると、項目を折り返すことができます。この例では、HyperlinkButtons を使用します。次の手順に従います。

1. 1.Silverlight プロジェクトで、ツールボックスから **C1WrapPanel** コントロールをドラッグし、.xaml ページの 終了タグの前に置きます。
2. **<c1:C1WrapPanel>**タグの間にカーソルを置き、**[Enter]**キーを押します。
3. 3.次の XAML を追加して HyperlinkButtons を折り返します。

XAML

```
  <Border  Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Orange">
          <HyperlinkButton Foreground="White"  Content="サンプルテキスト"
FontSize="25" />
      </Border>
      <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Green" c1:C1WrapPanel.BreakLine="After">
```

```
          <HyperlinkButton Foreground="White" Content="後ろに改行を追加します" />
      </Border>
      <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Blue">
          <HyperlinkButton Foreground="White" Content="C1WrapPanel"
FontSize="16"/>
      </Border>
      <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Red">
          <HyperlinkButton Foreground="White" Content="Silverlight" />
      </Border>
      <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Purple">
          <HyperlinkButton Foreground="White" Content="水平方向に折り返し"
FontSize="20"  />
      </Border>
```

2番目の HyperlinkButton では、**C1WrapPanel.BreakLine** プロパティは "**After**" に設定されています。これにより、ボタンの後に改行が追加されます。

4. 4.プロジェクトを実行します。**C1WrapPanel** は、次の図のようになります

水平方向に折り返す方法

デフォルトでは、項目は水平方向に折り返されます。ただし、垂直方向に折り返すこともできます。垂直方向の折り返しを指定するには、Orientation プロパティを設定します。この例では、HyperlinkButtons を使用します。次の手順に従います。

1. Silverlight プロジェクトで、ツールボックスから **C1WrapPanel** コントロールをドラッグし、.xaml ページの </Grid> 終了タグの前に置きます。
2. <c1:C1WrapPanel> タグで、Orientation プロパティを "Vertical" に設定します。XAML は次のようになります。

XAML

```
<c1:C1WrapPanel Orientation="Vertical">
```

3. **<c1:C1WrapPanel>** タグの間にカーソルを置き、[Enter]キーを押します。
4. 次の XAML を追加して HyperlinkButtons を折り返します。

XAML

```
    <Border  Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Orange">
          <HyperlinkButton Foreground="White"  Content="サンプルテキスト"
FontSize="25" />
      </Border>
      <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Green" c1:C1WrapPanel.BreakLine="After">
          <HyperlinkButton Foreground="White" Content="後ろに改行を追加します"
/>
```

```
        </Border>
        <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Blue">
            <HyperlinkButton Foreground="White" Content="C1WrapPanel"
FontSize="16"/>
        </Border>
        <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Red">
            <HyperlinkButton Foreground="White" Content="Silverlight" />
        </Border>
        <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Purple">
            <HyperlinkButton Foreground="White" Content="垂直方向に折り返し"
FontSize="20"  />
        </Border>
```

5. プロジェクトを実行します。**C1WrapPanel** は、次の図のようになります。

ListBox

連結データをリスト表示する場合は、**ListBox forWPF/Silverlight** に含まれる高パフォーマンスな2つのコントロールを利用してください。**C1ListBox** コントロールや **C1TileListBox** コントロールを使用して、リストをタイル状にレイアウトして表示したり、光学ズーム付きで表示することができます。これらのコントロールは UI の仮想化をサポートしています。そのため、動作は極めて高速であり、数千個の項目を表示してもパフォーマンスはほとんど低下しません。

ListBox for WPF/Silverlight には、次の主要な機能があります。

- **水平方向または垂直方向**

  ListBox コントロールは、水平方向と垂直方向の両方をサポートしています。このため、より多くのレイアウトシナリオに対応することができます。

- **項目のタイル表示**

  C1TileListBox では、項目を複数の行と列に並べて、タイル状に表示することができます。各項目のサイズとテンプレートを指定し、任意の方向を選択します。

- **光学ズーム**

  C1ListBox コントロールは、光学ズーム機能をサポートしています。ユーザーは、ピンチジェスチャを使用して、項目のサイズ変更操作を直観的に行うことができます。ズーム動作は流れるようにスムーズです。アプリケーションのパフォーマンスが犠牲になることはありません。

- **UI の仮想化**

  ListBox コントロールは UI の仮想化をサポートしています。そのため、動作は極めて高速であり、数千個の項目を表示してもパフォーマンスはほとんど低下しません。各レイアウトパスにレンダリングする項目の数を決定するには、**ViewportGap** プロパティと **ViewportPreviewGap** プロパティを設定します。これらのプロパティをシナリオに応じて調整できます。

- **プレビュー状態**

  パフォーマンスを可能な限り高めるために、**ListBox** コントロールは、ビューポートの外の項目をプレビュー状態でレンダリングすることができます。標準の ItemTemplate と同様に、Preview テンプレートは、ズームアウト中または高速スクロール中などのプレビュー状態にある項目の外観を定義します。その後、項目のスクロールやズームが終了すると、完全な項目テンプレートに切り替えられます。

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

C1ListBox クイックスタート

このクイックスタートは、**C1ListBox** コントロール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ガイドでは、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 **C1ListBox** 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。

ステップ 1

この手順では、Visual Studio で、**ListBox for WPF/Silverlight** を使用して 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**]→[**新規作成**]→[**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 アプリケーション]または[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 アプリケーション]または[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[WPF アプリケーション]または[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. 参照の追加ダイアログボックスで、以下のアセンブリを見つけて選択し、[OK]をクリックしてプロジェクトに参照を追加します。
   - **WPF**: C1.WPF.4.dll
   - **Silverlight**: C1.Silverlight.5.dll
5. <Grid> タグと タグの間に次の <StackPanel>> マークアップを追加して、**TextBlock** と **ProgressBar** を含む **StackPanel** を追加します。

   XAML

   ```
   <StackPanel x:Name="loading" VerticalAlignment="Center">
       <TextBlock Text="Retrieving data from Flickr..." TextAlignment="Center"/>
       <ProgressBar IsIndeterminate="True" Width="200"  Height="4"/>
   </StackPanel>
   ```

6. ツールボックスに移動し、[C1ListBox]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。これで、参照と XAML 名前空間が自動的に追加されます。
7. <c1:C1ListBox>タグを編集して、コントロールをカスタマイズします。

   XAML

   ```
   <c1:C1ListBox x:Name="listBox" ItemsSource="{Binding}" Background="Transparent"
   Visibility="Collapsed" ItemWidth="800" ItemHeight="600" Zoom="Fill"
   ViewportGap="0" RefreshWhileScrolling="False"></c1:C1ListBox>
   ```

これは、コントロールに名前を付け、コントロールの連結、背景、表示/非表示、サイズ、および更新機能をカスタマイズします。

8. <c1:C1ListBox> タグと <<c1:C1ListBox>> タグの間に、次のマークアップを追加します。

XAML

```
<c1:C1ListBox.PreviewItemTemplate>
    <DataTemplate>
        <Grid Background="Gray">
            <Image Source="{Binding Thumbnail}" Stretch="UniformToFill"/>
        </Grid>
    </DataTemplate>
</c1:C1ListBox.PreviewItemTemplate>
<c1:C1ListBox.ItemTemplate>
    <DataTemplate>
        <Grid>
            <Image Source="{Binding Content}" Stretch="UniformToFill"/>
            <TextBlock Text="{Binding Title}" Foreground="White" Margin="4 0 0
4" VerticalAlignment="Bottom"/>
        </Grid>
    </DataTemplate>
</c1:C1ListBox.ItemTemplate>
```

このマークアップは、**C1 ListBox** コントロールのコンテンツのデータテンプレートを追加します。このコントロールの連結はコードで行います。

ステップ 2

この手順では、フォトストリームから画像を表示するコードを追加します。

プログラムでコントロールにデータを追加するには、次の手順に従います。

1. [**ページ**]を選択して[プロパティ]ウィンドウに移動し、稲妻の[**イベント**]ボタンをクリックしてイベントを表示します。次に、下にスクロールして、**Loaded** イベントの横にある領域をダブルクリックします。

   これで、コードエディタが開き、**Page_Loaded** イベントが追加されます。

2. 次の imports 文をページの先頭に追加します。

Visual Basic

```
'WPF
Imports System.Linq
Imports System.Windows.Controls
Imports System.Windows
Imports System.Collections.Generic
Imports System.Net
Imports System.Xml.Linq
Imports System
Imports C1.WPF
OR
'Silverlight
Imports C1.Silverlight
Imports System
Imports System.Collections.Generic
Imports System.IO
Imports System.Linq
Imports System.Xml.Linq
```

```
Imports System.Net
Imports System.Collections.ObjectModel
```

C#

```
//WPF
using System.Linq;
using System.Windows.Controls;
using System.Windows;
using System.Collections.Generic;
using System.Net;
using System.Xml.Linq;
using System;
using C1.WPF;
OR
//Silverlight
using C1.Silverlight;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Xml.Linq;
using System.Collections.ObjectMode;
```

3. 最初のイベントハンドラ内に次のコードを追加します。

Visual Basic

```
DataContext = Enumerable.Range(0, 100)
AddHandler Loaded, AddressOf ListBoxSample_Loaded
```

C#

```
DataContext = Enumerable.Range(0, 100);
Loaded += new System.Windows.RoutedEventHandler(ListBoxSample_Loaded);
```

4. MainPage or MainWindow class クラス内に次のコードを追加します。

Visual Basic

```
Private Sub ListBoxSample_Loaded(sender As Object, e As RoutedEventArgs)
        LoadPhotos()
    End Sub
    Private Sub LoadPhotos()
        Dim flickrUrl = "http://api.flickr.com/services/feeds/photos_public.gne"
        Dim AtomNS = "http://www.w3.org/2005/Atom"
        loading.Visibility = Visibility.Visible
        retry.Visibility = Visibility.Collapsed
        Dim photos = New List(Of Photo)()
        Dim client = New WebClient()
        AddHandler client.OpenReadCompleted, Function(s, e)
                                                 Try
                                                     '#Region "** flickr データの
```

```
解析"
                                                    Dim doc =
XDocument.Load(e.Result)
                                                    For Each entry In
doc.Descendants(XName.[Get]("entry", AtomNS))
                                                        Dim title =
entry.Element(XName.[Get]("title", AtomNS)).Value
                                                        Dim enclosure =
entry.Elements(XName.[Get]("link", AtomNS)).Where(Function(elem)
elem.Attribute("rel").Value = "enclosure").FirstOrDefault()
                                                        Dim contentUri =
enclosure.Attribute("href").Value
                                                        photos.Add(New Photo()
With { _
                                                         .Title = title, _
                                                         .Content = contentUri,
_
                                                         .Thumbnail =
contentUri.Replace("_b", "_m") _
                                                        })
                                                    Next
                                                    '#End Region
                                                    listBox.ItemsSource =
photos
                                                    loading.Visibility =
Visibility.Collapsed
                                                    listBox.Visibility =
Visibility.Visible
                                                Catch
                                                    MessageBox.Show("There was
an error when attempting to download data from Flickr.")
                                                    loading.Visibility =
Visibility.Collapsed
                                                    retry.Visibility =
Visibility.Visible
                                                End Try
                                            End Function
        client.OpenReadAsync(New Uri(flickrUrl))
    End Sub
    Private Sub Retry_Click(sender As Object, e As RoutedEventArgs)
        LoadPhotos()
    End Sub
#Region "** public properties"
    Public Property Orientation() As Orientation
        Get
            Return listBox.Orientation
        End Get
        Set(value As Orientation)
            listBox.Orientation = value
        End Set
    End Property
    Public Property ItemWidth() As Double
        Get
```

```
            Return listBox.ItemWidth
        End Get
        Set(value As Double)
            listBox.ItemWidth = value
        End Set
    End Property
    Public Property ItemHeight() As Double
        Get
            Return listBox.ItemHeight
        End Get
        Set(value As Double)
            listBox.ItemHeight = value
        End Set
    End Property
    Public Property ZoomMode() As ZoomMode
        Get
            Return listBox.ZoomMode
        End Get
        Set(value As ZoomMode)
            listBox.ZoomMode = value
        End Set
    End Property
#End Region
```

C#

```
 void ListBoxSample_Loaded(object sender, RoutedEventArgs e)
        {
            LoadPhotos();
        }
        private void LoadPhotos()
        {
            var flickrUrl =
"http://api.flickr.com/services/feeds/photos_public.gne";
            var AtomNS = "http://www.w3.org/2005/Atom";
            loading.Visibility = Visibility.Visible;
            retry.Visibility = Visibility.Collapsed;
            var photos = new List<Photo>();
            var client = new WebClient();
            client.OpenReadCompleted += (s, e) =>
            {
                try
                {
                    #region ** parse flickr data
                    var doc = XDocument.Load(e.Result);
                    foreach (var entry in doc.Descendants(XName.Get("entry",
AtomNS)))
                    {
                        var title = entry.Element(XName.Get("title",
AtomNS)).Value;
                        var enclosure = entry.Elements(XName.Get("link",
AtomNS)).Where(elem => elem.Attribute("rel").Value ==
"enclosure").FirstOrDefault();
```

```
                        var contentUri = enclosure.Attribute("href").Value;
                        photos.Add(new Photo() { Title = title, Content =
contentUri, Thumbnail = contentUri.Replace("_b", "_m") });
                    }
                    #endregion
                    listBox.ItemsSource = photos;
                    loading.Visibility = Visibility.Collapsed;
                    listBox.Visibility = Visibility.Visible;
                    retry.Visibility = Visibility.Collapsed;
                }
                catch
                {
                    MessageBox.Show("There was an error when attempting to
download data from Flickr.");
                    listBox.Visibility = Visibility.Collapsed;
                    loading.Visibility = Visibility.Collapsed;
                    retry.Visibility = Visibility.Visible;
                }
            };
            client.OpenReadAsync(new Uri(flickrUrl));
        }
        private void Retry_Click(object sender, RoutedEventArgs e)
        {
            LoadPhotos();
        }
        #region ** public properties
        public Orientation Orientation
        {
            get
            {
                return listBox.Orientation;
            }
            set
            {
                listBox.Orientation = value;
            }
        }
        public double ItemWidth
        {
            get
            {
                return listBox.ItemWidth;
            }
            set
            {
                listBox.ItemWidth = value;
            }
        }
        public double ItemHeight
        {
            get
            {
                return listBox.ItemHeight;
```

```
            }
            set
            {
                listBox.ItemHeight = value;
            }
        }
        public ZoomMode ZoomMode
        {
            get
            {
                return listBox.ZoomMode;
            }
            set
            {
                listBox.ZoomMode = value;
            }
        }
        #endregion
```

5. 上のコードは、Flickr のパブリックフォトストリームから画像を取得し、それらの画像のリストに C1ListBox を連結します。

6. MainPage クラスの直後に次のコードを追加します。

Visual Basic

```
Public Class Photo
    Public Property Title() As String
        Get
            Return m_Title
        End Get
        Set(value As String)
            m_Title = Value
        End Set
    End Property
    Private m_Title As String
    Public Property Thumbnail() As String
        Get
            Return m_Thumbnail
        End Get
        Set(value As String)
            m_Thumbnail = Value
        End Set
    End Property
    Private m_Thumbnail As String
    Public Property Content() As String
        Get
            Return m_Content
        End Get
        Set(value As String)
            m_Content = Value
        End Set
    End Property
```

```
    Private m_Content As String
End Class
```

C#

```
public class Photo
    {
        public string Title { get; set; }
        public string Thumbnail { get; set; }
        public string Content { get; set; }
    }
```

これで、**C1TileListBox** データ追加できました。次の「手順 3：ListBox アプリケーションの実行」では、**ListBox for** の機能について説明します。

ステップ 3

これまでに WPF/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、**ListBox for WPF/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、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。
2. アプリケーションが表示され、画像が表示されます。
3. コントロールの右側にあるスクロールバーを使用して、イメージストリームをスクロールします。

タッチ機能がある場合は、ピンチして画像をズームしてみてください。

おめでとうございます!

これで **ListBox for WPF/Silverlight** クイックスタートは完了です。**C1ListBox** コントロールを使用するアプリケーションを作成し、アプリケーションの実行時機能をいくつか確認することができました。

C1TileListBox クイックスタート

このクイックスタートは、**C1TileListBox** コントロール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ガイドでは、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 **C1TileListBox** 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。

ステップ 1

この手順では、Visual Studio で、**TileListBox for WPF/Silverlight** を使用して WPF/Silverlight アプリケーションを作成します。

プロジェクトを設定するには次の手順に従います。

1. Visual Studio で、[**ファイル**]→[**新規作成**]→[**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 アプリケーション]または[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 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[WPF アプリケーション]または[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. 参照の追加ダイアログボックスで、以下のアセンブリを見つけて選択し、[OK]をクリックしてプロジェクトに参照を追加します。

- **WPF**: C1.WPF.4.dll
- **Silverlight**: C1.Silverlight.5.dll

5. カーソルを <Grid> タグと </Grid> タグの間に置き、ツールボックスに移動します。**C1TileListBox** アイコンをダブルクリックして、このコントロールをグリッドに追加します。これで、参照と XAML 名前空間が自動的に追加されます。
6. **<c1:C1TileListBox>** タグを編集して、コントロールをカスタマイズします。

XAML

```
<c1:C1TileListBox x:Name="tileListBox" ItemsSource="{Binding}" ItemWidth="130"
ItemHeight="130"></c1:C1TileListBox>
```

これは、コントロールに名前を付け、**ItemsSource** プロパティを設定し（後でこのプロパティをコードでカスタマイズします）、コントロールのサイズを設定します。

7. **<c1:C1TileListBox>** タグと **</c1:C1TileListBox>** タグの間にマークアップを追加します。コードは次のようになります。

XAML

```
<c1:C1TileListBox x:Name="tileListBox" ItemsSource="{Binding}" ItemWidth="130"
ItemHeight="130">
    <c1:C1TileListBox.PreviewItemTemplate>
        <DataTemplate>
            <Grid Background="Gray" />
                </DataTemplate>
                </c1:C1TileListBox.PreviewItemTemplate>
                <c1:C1TileListBox.ItemTemplate>
                <DataTemplate>
                <Grid Background="LightBlue">
                    <Image Source="{Binding Thumbnail}" Stretch="UniformToFill"
/>
                    <TextBlock Text="{Binding Title}" Margin="4 0 0 4"
VerticalAlignment="Bottom" />
                </Grid>
        </DataTemplate>
    </c1:C1TileListBox.ItemTemplate>
</c1:C1TileListBox>
```

このマークアップは、**C1TileListBox** コントロールのコンテンツのデータテンプレートを追加します。このコントロールの連結はコードで行います。

ステップ 2

前の手順では、**C1TileListBox** 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、コントロールをデータに連結するコードを追加します。

プログラムでコントロールにデータを追加するには、次の手順に従います。

1. ページを右クリックし、[**コードの表示**]を選択してコードエディタを開きます。
2. 次の imports 文をページの先頭に追加します

Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.Diagnostics
```

```
Imports System.Linq
Imports System.Net
Imports System.Windows;
Imports System.Windows.Controls;
Imports System.Xml.Linq;
Imports C1.WPF
OR
Imports System
Imports System.Collections.Generic
Imports System.IO
Imports System.Linq
Imports System.Xml.Linq
Imports System.Net
Imports C1.Silverlight
```

C#

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Xml.Linq;
using C1.WPF;
OR
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using C1.Silverlight;
```

3. **MainPage** クラス内の最初のイベントハンドラ内に次のコードを追加します。

Visual Basic

```
DataContext = Enumerable.Range(0, 100).[Select](Function(i) New Item() With
{.Title = i.ToString()})
```

C#

```
DataContext = Enumerable.Range(0, 100).Select(i => new Item { Title =
i.ToString() });
```

4. **MainPage** クラス内の最初のイベントハンドラの下に次のコードを追加します。

Visual Basic

```
#Region "** public properties"
    Public Property Orientation() As Orientation
        Get
            Return tileListBox.Orientation
        End Get
        Set(value As Orientation)
            tileListBox.Orientation = value
```

```
        End Set
    End Property
    Public Property ItemWidth() As Double
        Get
            Return tileListBox.ItemWidth
        End Get
        Set(value As Double)
            tileListBox.ItemWidth = value
        End Set
    End Property
    Public Property ItemHeight() As Double
        Get
            Return tileListBox.ItemHeight
        End Get
        Set(value As Double)
            tileListBox.ItemHeight = value
        End Set
    End Property
    Public Property ZoomMode() As ZoomMode
        Get
            Return tileListBox.ZoomMode
        End Get
        Set(value As ZoomMode)
            tileListBox.ZoomMode = value
        End Set
    End Property
#End Region
```

C#

```
#region ** public properties
        public Orientation Orientation
        {
            get
            {
                return tileListBox.Orientation;
            }
            set
            {
                tileListBox.Orientation = value;
            }
        }
        public double ItemWidth
        {
            get
            {
                return tileListBox.ItemWidth;
            }
            set
            {
                tileListBox.ItemWidth = value;
            }
        }
        public double ItemHeight
```

```
        {
            get
            {
                return tileListBox.ItemHeight;
            }
            set
            {
                tileListBox.ItemHeight = value;
            }
        }
        public ZoomMode ZoomMode
        {
            get
            {
                return tileListBox.ZoomMode;
            }
            set
            {
                tileListBox.ZoomMode = value;
            }
        }
#endregion
```

上のコードは、**C1TileListBox** を数値のリストに連結します。

5. **MainPage** クラスの直後に次のコードを追加します。

Visual Basic

```
Public Class Item
    Public Property Title() As String
        Get
            Return m_Title
        End Get
        Set(value As String)
            m_Title = Value
        End Set
    End Property
    Private m_Title As String
    Public Property Thumbnail() As String
        Get
            Return m_Thumbnail
        End Get
        Set(value As String)
            m_Thumbnail = Value
        End Set
    End Property
    Private m_Thumbnail As String
End Class
```

C#

```
public class Item
    {
```

```
        public string Title { get; set; }
        public string Thumbnail { get; set; }
    }
```

これで、**C1TileTileListBox** にデータを追加できました。次の「手順 3：TileListBox アプリケーションの実行」では、**TileListBox for** の機能について説明します。

ステップ 3

これまでに WPF/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、TileListBox for WPF/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、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。
2. アプリケーションが表示され、数値のリストがタイル表示されます。
3. コントロールの右側にあるスクロールバーを使用して、番号付きのタイルをスクロールします。
4. タッチ機能がある場合は、ピンチして画像をズームしてみてください。

おめでとうございます!

これで **TileListBox for WPF/Silverlight** クイックスタートは完了です。**C1TileListBox** コントロールを使用するアプリケーションを作成し、アプリケーションの実行時機能をいくつか確認することができました。

リストボックスの機能

C1ListBox コントロールでは次の機能が用意されています。

光学ズーム

ListBox for WPF/Silverlight コントロールは、光学ズーム機能をサポートしています。ユーザーは、ピンチジェスチャを使用して、項目のサイズ変更操作を直観的に行うことができます。ズーム動作は流れるようにスムーズです。アプリケーションのパフォーマンスが犠牲になることはありません。

ZoomMode プロパティと **Zoom** プロパティを使用して、ズームをカスタマイズできます。**ZoomMode** プロパティは、ズームが有効か無効かを取得または設定します。**Zoom** プロパティは、コントロールに適用されるズーム値です。**ZoomChanged** イベントは、コントロールのズーム値が変更されたときにトリガされます。

Orientation

ListBoxコントロールは、水平方向と垂直方向の両方をサポートしています。このため、より多くのレイアウトシナリオに対応することができます。コントロールの方向を設定するには、**Orientation** プロパティを **Horizontal** または **Vertical** に設定します。

プレビュー状態

パフォーマンスを可能な限り高めるために、**ListBox** コントロールは、ビューポートの外の項目をプレビュー状態でレンダリングすることができます。標準の **ItemTemplate** と同様に、**Preview** テンプレートは、ズームアウト中または高速スクロール中などのプレビュー状態にある項目の外観を定義します。その後、項目のスクロールやズームが終了すると、完全な項目テンプレートに切り替えられます。

UI の仮想化

ListBoxコントロールは UI の仮想化をサポートしています。そのため、動作は極めて高速であり、数千個の項目を表示してもパフォーマンスはほとんど低下しません。各レイアウトパスにレンダリングする項目の数を決定するには、**ViewportGap** プロパティと **ViewportPreviewGap** プロパティを設定します。これらのプロパティをシナリオに応じて調整できます。

ViewportGap プロパティは、毎回のレイアウトパスでビューポートのサイズを決定する係数を取得または設定します。0を指定すると、ビューポートのサイズはスクロールビューアのビューポートと同じになります。0.5 を指定すると、ビューポートが元のビューポートの両側に合わせて画面の半分だけ大きくなるように拡大されます。

ViewportPreviewGap プロパティは、毎回のレイアウトパスでプレビューモードの項目をレンダリングするためのビューポートのサイズを決定する係数を取得または設定します。

Masked Text Box

WPF/Silverlight アプリケーションで入力を検証します。**MaskedTextBox for WPF/Silverlight** は、Microsoft Windows Forms の標準 **MaskedTextBox** コントロールと同様に、自動的に入力を検証するためのマスクが付いたテキストボックスを提供します。

- **データを検証し、UI の機能を拡張する**

  C1MaskedTextBox は、自動的に入力を検証するためのマスクが付いたテキストボックスを提供します。編集マスクは、エンドユーザーがコントロールに無効な文字を入力できないようにすることで、UI の機能を拡張します。詳細については、「マスクの書式設定」と「通貨のマスクの追加」を参照してください。

- **ユーザーに必要な情報のヒントを提供する**

  マスク付きテキストボックスコントロールの **Watermark** プロパティを使用して、要求されている情報の種類をエンドユーザーに示すことができます。詳細については、「ウォーターマーク」を参照してください。

- **ClearStyle を使用して簡単に色を変更する**

  C1MaskedTextBox は、テンプレートを上書きしなくてもコントロールのブラシを簡単に変更できる **ComponentOne ClearStyle** 技術をサポートします。Visual Studio でブラシのプロパティをいくつか設定するだけで、アコーディオンの各部のスタイルを簡単に設定できます。ComponentOne ClearStyle 技術の詳細については、「ClearStyle プロパティ」を参照してください。

> **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。

クイックスタート

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 **MaskedTextBox for** を使用する WPF/Silverlight アプリケーションを作成します。**C1MaskedTextBox** コントロールをアプリケーションに追加するだけで、完全な機能を備えた入力エディタとして使用できます。さらに、そのコントロールをアプリケーションに合わせてカスタマイズできます。

プロジェクトをセットアップし、**C1MaskedTextBox** 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、[**ファイル**]→[**新しいプロジェクト**]を選択します。
2. MainWindow の Width を「400」に設定して、当初のウィンドウのサイズを変更します。
3. ツールボックスに移動し、[C1MaskedTextBox]アイコンをダブルクリックして、MainWindow にコントロールを追加します。この手順3を複数回繰り返して、合計4つの C1MaskedTextBox コントロールを追加します。
4. ツールボックスで、Label アイコンをダブルクリックして、このコントロールを MainWindow に追加します。この手順4を

複数回繰り返して、合計5つの標準の Label コントロールを追加します。

5. ツールボックスに移動し、[**C1MaskedTextBox**]アイコンをダブルクリックして、MainWindow にコントロールを追加します。この手順3を複数回繰り返して、合計4つの **C1MaskedTextBox** コントロールを追加します。

これで、WPF/Silverlight アプリケーションが正しく作成され、アプリケーションに **C1MaskedTextBox** コントロールが追加されました。次の手順では、コントロールをカスタマイズし、アプリケーションの設定を完成させます。

手順 2:コントロールの設定

前の手順では、新しい WPF/Silverlightプロジェクトを作成し、アプリケーションに4つの **C1MaskedTextBox** コントロールと5つの **Label** コントロールを追加しました。この手順では、引き続き、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。

次の手順に従います。

1. [**デザイン**]ビューで、**Label1** コントロールを1回クリックして選択し、[**プロパティ**]ウィンドウに移動し、その **Content** プロパティを「**社員情報**」に設定します。
2. 残りの各 **Label** コントロールを順に選択し、[**プロパティ**]ウィンドウに移動して、それぞれを次のように設定します。

   - Content プロパティのデフォルトの「Label」テキストを削除します。
   - **FontSize** プロパティを「**9**」に設定します。

3. [XAML] ビューに切り替え、**Watermark="名前"** を **<c1:C1MaskedTextBox>**タグに追加して、**C1MaskedTextBox1** を次のようにカスタマイズします。

```
XAML
<c1:C1MaskedTextBox Height="23" Margin="21,46,167,0" Name="C1MaskedTextBox1"
VerticalAlignment="Top" Watermark="名前" />
```

   これで、コントロールにウォーターマークが追加されます。

4. [XAML] ビューに切り替え,**Watermark="社員ID" Mask="000-00-0000"** を**<c1:C1MaskedTextBox>** タグに追加して、**C1MaskedTextBox2** を次のようにカスタマイズします。

```
XAML
<c1:C1MaskedTextBox Margin="14,98,12,0" Name="C1MaskedTextBox2" Height="23"
```

```
VerticalAlignment="Top" Watermark="社員ID" Mask="000-00-0000" />
```

これで、コントロールにウォーターマークとマスクが追加されます。

5. Watermark="入社日" Mask="00/00/0000" を <c1:C1MaskedTextBox> タグに追加して、C1MaskedTextBox3 を次のようにカスタマイズします。

XAML

```
<c1:C1MaskedTextBox Height="23" Margin="14,0,12,87" Name="C1MaskedTextBox3"
VerticalAlignment="Bottom" Watermark="入社日" Mask="00/00/0000"/>
```

これで、コントロールにウォーターマークとマスクが追加されます。

6. Watermark="電話番号" Mask="(999) 000-0000" を <c1:C1MaskedTextBox> タグに追加して、C1MaskedTextBox4 を次のようにカスタマイズします。

XAML

```
<c1:C1MaskedTextBox Height="23" Margin="14,0,12,33" Name="C1MaskedTextBox4"
VerticalAlignment="Bottom" Watermark="電話番号" Mask="(999) 000-0000"/>
```

これで、コントロールにウォーターマークとマスクが追加されます。

これで、アプリケーションのユーザーインターフェイスを設定できました。次の手順では、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. [**デザイン**]ビューで、**C1MaskedTextBox1** をダブルクリックして**コード**ビューに切り替え、**C1MaskedTextBox1_TextChanged** イベントハンドラを作成します。[**デザイン**]ビューに戻り、各 **C1MaskedTextBox** コントロールでこの手順を繰り返して、各コントロールに **TextChanged** イベントハンドラを作成します。
2. [**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
または
Imports C1.Silverlight
```

C#

```
using C1.WPF;
または
using C1.Silverlight;
```

3. **C1MaskedTextBox1_TextChanged** 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 C1MaskedTextBox1_TextChanged(ByVal sender As System.Object, ByVal e
```

```
As System.Windows.Controls.TextChangedEventArgs) Handles
C1MaskedTextBox1.TextChanged
Me.Label2.Content = "マスク:" & Me.C1MaskedTextBox1.Mask & " 値:" &
Me.C1MaskedTextBox1.Value & " テキスト:" & Me.C1MaskedTextBox1.Text
End Sub
```

C#

```
private void c1MaskedTextBox1_TextChanged(object sender, TextChangedEventArgs e)
{
this.label2.Content = "マスク:" + this.C1MaskedTextBox1.Mask + " 値:" +
this.C1MaskedTextBox1.Value + " テキスト:" + this.C1MaskedTextBox1.Text;
}
```

4. **C1MaskedTextBox2_TextChanged** 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 C1MaskedTextBox2_TextChanged(ByVal sender As System.Object, ByVal e
As System.Windows.Controls.TextChangedEventArgs) Handles
C1MaskedTextBox2.TextChanged
Me.Label3.Content = "マスク:" & Me.C1MaskedTextBox2.Mask & " 値:" &
Me.C1MaskedTextBox2.Value & " テキスト:" & Me.C1MaskedTextBox2.Text
End Sub
```

C#

```
private void c1MaskedTextBox2_TextChanged(object sender, TextChangedEventArgs e)
{
this.label3.Content = "マスク:" + this.C1MaskedTextBox2.Mask + " 値:" +
this.C1MaskedTextBox2.Value + " テキスト:" + this.C1MaskedTextBox2.Text;
}
```

5. **C1MaskedTextBox3_TextChanged** 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 C1MaskedTextBox3_TextChanged(ByVal sender As System.Object, ByVal e
As System.Windows.Controls.TextChangedEventArgs) Handles
C1MaskedTextBox3.TextChanged
Me.Label4.Content = "マスク:" & Me.C1MaskedTextBox3.Mask & " 値:" &
Me.C1MaskedTextBox3.Value & " テキスト:" & Me.C1MaskedTextBox3.Text
End Sub
```

C#

```
private void c1MaskedTextBox3_TextChanged(object sender, TextChangedEventArgs e)
{
this.label4.Content = "マスク:" + this.C1MaskedTextBox3.Mask + " 値:" +
this.C1MaskedTextBox3.Value + " テキスト:" + this.C1MaskedTextBox3.Text;
}
```

6. **C1MaskedTextBox4_TextChanged** 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 C1MaskedTextBox4_TextChanged(ByVal sender As System.Object, ByVal e
As System.Windows.Controls.TextChangedEventArgs) Handles
C1MaskedTextBox4.TextChanged
Me.Label5.Content = "マスク:" & Me.C1MaskedTextBox4.Mask & " 値:" &
Me.C1MaskedTextBox4.Value & " テキスト:" & Me.C1MaskedTextBox4.Text
End Sub
```

C#

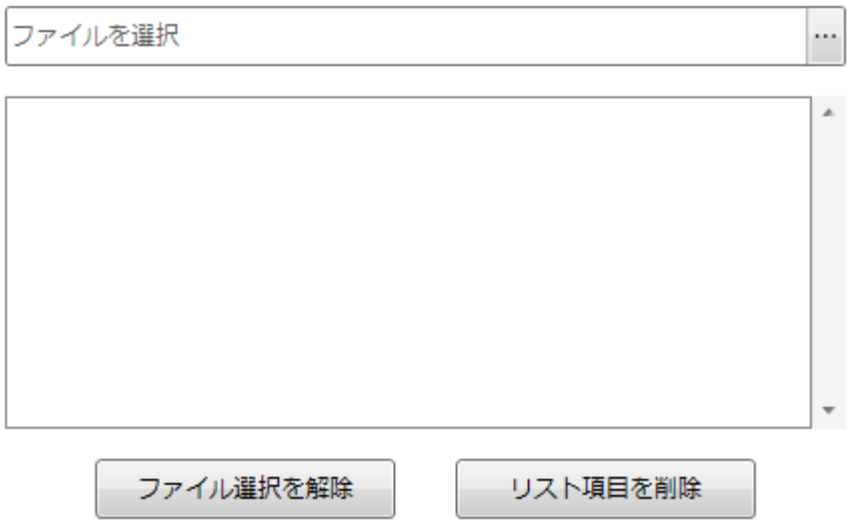
```
private void c1MaskedTextBox4_TextChanged(object sender, TextChangedEventArgs e)
{
this.label5.Content = "マスク:" + this.C1MaskedTextBox4.Mask + " 値:" +
this.C1MaskedTextBox4.Value + " テキスト:" + this.C1MaskedTextBox4.Text;
}
```

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

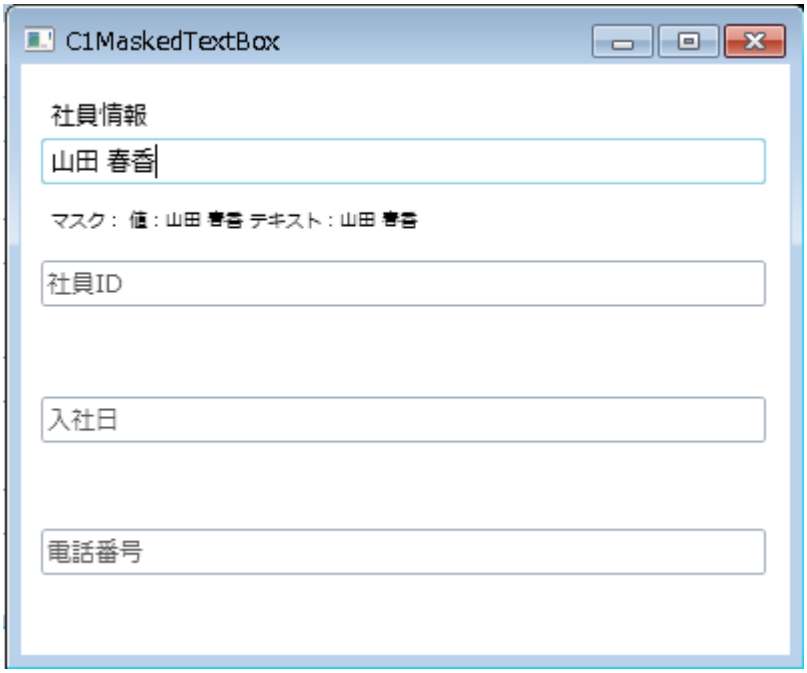
これまでに WPF/Silverlight アプリケーションを作成し、外観と動作をカスタマイズしたので、次にアプリケーションを実行します。アプリケーションを実行して **MaskedTextBox for WPF/Silverlight** の実行時の動作を確認するには、次の手順に従います。

1. [**デバッグ**]メニューから[**デバッグ開始**]を選択し、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。



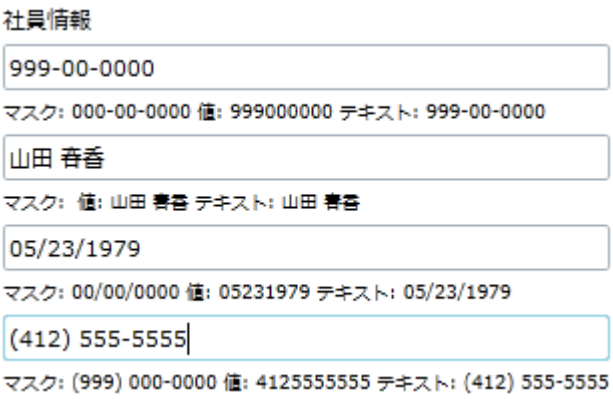
   各 **C1MaskedTextBox** コントロールに表示されるウォーターマークに注目してください。

2. 最初の **C1MaskedTextBox** コントロールに数字を入力します。



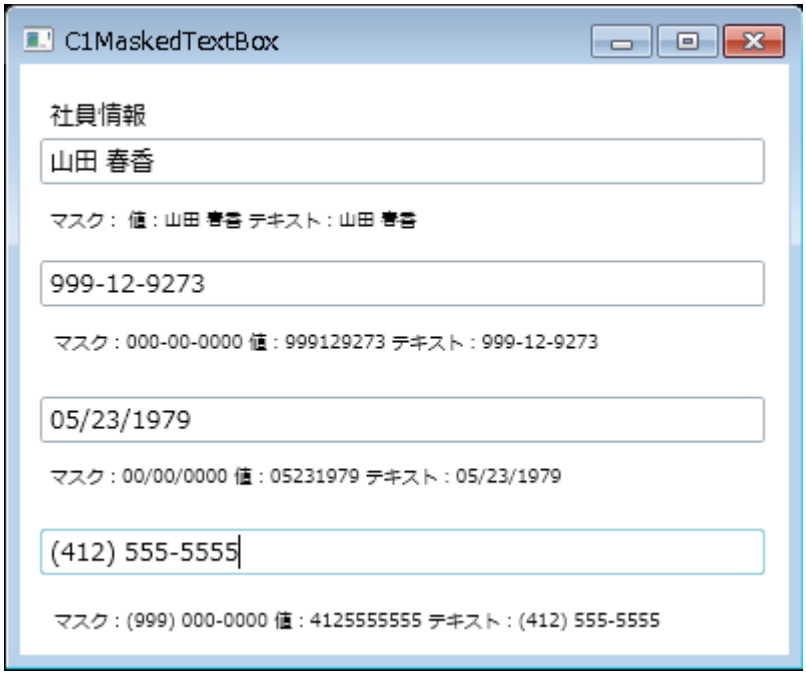
コントロールの下にあるラベルには、マスク、現在の値、および現在のテキストが表示されます。

3. 2番目の **C1MaskedTextBox** コントロールに文字列を入力します。



このコントロールにはマスクが設定されていません。したがって、必要なら、数字も他の文字もコントロールに入力できます。

4. 3番目の **C1MaskedTextBox** コントロールに文字列を入力してみます。入力できないことがわかります。**Mask** プロパティは、数字のみを受け付けるように設定されていました。代わりに数値を入力します。これは問題ありません。
5. 残りのコントロールに数字を入力します。アプリケーションは次の図のように表示されます。



各 **C1MaskedTextBox** コントロールの下に表示される **Value** プロパティには、リテラル文字が含まれません。一方、**Text** プロパティにはリテラル文字が含まれます。

おめでとうございます!

これで、**MaskedTextBox for WPF/Silverlight** クイックスタートは完了です。**MaskedTextBox for WPF/Silverlight** アプリケーションを作成し、コントロールの外観と動作をカスタマイズし、アプリケーションの実行時機能をいくつか確認することができました.

マスク要素

MaskedTextBox for WPF/Silverlight は、Microsoft によって定義される標準の数値書式文字列をサポートします。**Mask** 文字列は、1つ以上のマスク要素で構成される必要があります。マスク要素については、次の表で説明します。

要素	説明
0	必須の数字。この要素は単一の数字（0～9）を表します。
9	数字またはスペース（省略可）。
#	数字またはスペース（省略可）。マスク内のこの位置が空白の場合、**Text** プロパティにはスペースとしてレンダリングされます。プラス記号（+）とマイナス記号（-）を使用できます。
L	英字（省略不可）。入力は ASCII 文字の a ～ z と A ～ Z に制限されます。このマスク要素は、正規表現の [a-zA-Z] と同じです。
?	英字（省略可）。入力は ASCII 文字の a ～ z と A ～ Z に制限されます。このマスク要素は、正規表現の [a-zA-Z]? と同じです。
&	文字（省略不可）。
C	文字（省略可）。任意の非制御文字。

A	英数字（省略）。
a	英数字（省略可）。
.	小数点のプレースホルダ。実際に使用される表示文字は、書式プロバイダに合わせた小数点記号になります。
,	桁区切りのプレースホルダ。実際に使用される表示文字は、書式プロバイダに合わせた桁区切りのプレースホルダになります。
:	時刻の区切り。実際に使用される表示文字は、書式プロバイダに合わせた時刻記号になります。
/	日付の区切り。実際に使用される表示文字は、書式プロバイダに合わせた日付記号になります。
$	通貨記号。実際に表示される文字は、書式プロバイダに合わせた通貨記号になります。
<	下にシフトします。後続のすべての文字を小文字に変換します。
>	上にシフトします。後続のすべての文字を大文字に変換します。
|	前に行われた上または下へのシフトを無効にします。
\	エスケープ。マスク文字をエスケープしてリテラルにします。「\\」はバックスラッシュのエスケープシーケンスです。
All other characters	リテラル。すべての非マスク要素は、**C1MaskedTextBox** 内にそのまま表示されます。リテラルは実行時に常にマスク内の静的位置を占め、ユーザーはリテラルを移動したり削除することはできません。

小数点記号（.）、桁区切り記号（,）、時刻記号（:）、日付記号（/）、通貨記号（$）には、デフォルトでアプリケーションのカルチャで定義された記号が表示されます。

リテラル

「マスクの書式設定」で定義されたマスク要素に加えて、他の文字をマスクに入れることができます。これらの文字をリテラルと呼びます。リテラルは、**C1MaskedTextBox** 内にそのまま表示される非マスク要素です。リテラルは実行時に常にマスク内の静的位置を占め、ユーザーはリテラルを移動したり削除することはできません。

たとえば、**Mask** プロパティを「(999)-000-0000」に設定して電話番号を定義する場合、マスク文字としては「9」要素と「0」要素があります。その他の文字、つまりハイフンと丸かっこはリテラルです。これらの文字は、**C1MaskedTextBox** コントロールにそのまま表示されます。

> リテラルを使用する場合は、**TextMaskFormat** プロパティを IncludeLiterals または IncludePromptAndLiterals に設定する必要があります。リテラルを使用しない場合は、TextMaskFormat を IncludePrompt または ExcludePromptAndLiterals に設定します。

通貨のマスクの追加

Mask プロパティを使用して、通貨値のマスクを簡単に追加できます。デフォルトでは、**C1MaskedTextBox** コントロールは、最初に **Mask** が設定されませんが、設計時、XAML、またはコードでこの値をカスタマイズできます。マスク文字の詳細については、「マスク要素」を参照してください。

XAML の場合

Mask プロパティを設定するには、**Mask="\ 999,990"** を **<c1:C1MaskedTextBox>** タグに追加します。次のようになります。

XAML

```
<c1:C1MaskedTextBox Height="23" HorizontalAlignment="Left" Margin="10,10,0,0"
Name="C1MaskedTextBox1" VerticalAlignment="Top" Width="120" Mask="$999,999.00">
</c1:C1MaskedTextBox>
```

コードの場合

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

Visual Basic

```
C1MaskedTextBox1.Mask = "$999,999.00"
```

C#

```
c1MaskedTextBox1.Mask = "$999,999.00";
```

設計時

設計時に **Mask** プロパティを設定するには、次の手順に従います。

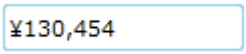
1. **C1MaskedTextBox** コントロールをクリックして選択します。
2. [**プロパティ**]ウィンドウに移動し、**Mask** プロパティの横にあるテキストボックスに「¥999,999.00」と入力します。

これで、**Mask** プロパティは指定された値に設定されます。.

プロジェクトの実行と確認

コントロールにマスクが表示されます。

数値を入力します。マスクが数字に置き換わることがわかります。



MaskedTextBox の機能

プロンプト

C1MaskedTextBox コントロールにプロンプト文字を入れるように選択できます。プロンプト文字は、ユーザーにテキストの入力を求めるためにコントロールに表示されるテキストを定義します。プロンプト文字は、ユーザーにテキストを入力できることを示したり、入力できるテキストの種類を説明するために使用できます。デフォルトでは、下線文字「_」が使用されます。プロンプト文字を使用する場合は、TextMaskFormat プロパティを IncludePrompt または IncludePromptAndLiterals に設定する必要があります。プロンプト文字を使用しない場合は、TextMaskFormat を IncludeLiterals または ExcludePromptAndLiterals に設定します。

プロンプト文字を使用する場合は、**TextMaskFormat** プロパティを **IncludePrompt** または **IncludePromptAndLiterals** に

設定する必要があります。プロンプト文字を使用しない場合は、**TextMaskFormat** を **IncludeLiterals** または **ExcludePromptAndLiterals** に設定します。

In XAML

PromptChar プロパティを設定するには、**Mask="0000" PromptChar="#"** を **<c1:C1MaskedTextBox>** タグに追加します。次のようになります。

XAML

```
<c1:C1MaskedTextBox Height="23" HorizontalAlignment="Left" Margin="10,10,0,0"
Name="C1MaskedTextBox1" VerticalAlignment="Top" Width="120" Mask="0000"
PromptChar="#"></c1:C1MaskedTextBox>
```

コードの場合

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

Visual Basic

```
Dim x As Char = "#"c
C1MaskedTextBox1.Mask = "0000"
C1MaskedTextBox1.PromptChar = x
```

C#

```
char x = '#';
this.c1MaskedTextBox1.Mask = "0000";
this.c1MaskedTextBox1.PromptChar = x;
```

設計時

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

1. **C1MaskedTextBox** 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、**Mask** プロパティの横にあるテキストボックスに「**0000**」と入力して、マスクを設定します。
3. [**プロパティ**]ウィンドウで、**PromptChar** プロパティの横にあるテキストボックスに「**#**」(シャープ文字)を入力します。

プロジェクトの実行と確認

コントロールには、プロンプトとしてシャープ文字が表示されます。次の図では、数値 32 がコントロールに入力されています。

値

値の設定

Value プロパティは、現在表示されているテキストを決定します。デフォルトでは、**C1MaskedTextBox** コントロールは、最初に **Value** が設定されませんが、設計時、XAML、またはコードでこの値をカスタマイズできます。

XAML の場合

たとえば、**Value** プロパティを設定するには、次に示すように **Value="123"** を **<c1:C1MaskedTextBox>** タグに追加します。

XAML

```
<c1:C1MaskedTextBox Height="23" HorizontalAlignment="Left" Margin="10,10,0,0"
Name="C1MaskedTextBox1" VerticalAlignment="Top" Width="120" Value="123">
</c1:C1MaskedTextBox>
```

コードの場合

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

Visual Basic

```
C1MaskedTextBox1.Value = "123"
```

C#

```
c1MaskedTextBox1.Value = "123";
```

設計時

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

1. **C1MaskedTextBox** 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、**Value** プロパティの横にあるテキストボックスに値(たとえば、「**123**」)を入力します。

これで、Value プロパティは指定された値に設定されます。

編集の禁止

デフォルトでは、**C1MaskedTextBox** コントロールの Value プロパティは、実行時にユーザーによって編集可能です。コントロールの編集をロックするには、**IsReadOnly** プロパティを **True** に設定します。

XAML の場合

XAML で **C1MaskedTextBox** コントロールの実行時の編集をロックするには、**IsReadOnly="True"** を **<c1:C1MaskedTextBox>** タグに追加します。次のようになります。

XAML

```
<c1:C1MaskedTextBox Height="23" HorizontalAlignment="Left" Margin="10,10,0,0"
Name="C1MaskedTextBox1" VerticalAlignment="Top" Width="120" IsReadOnly="True">
```

```
</c1:C1MaskedTextBox>
```

コードの場合

C1MaskedTextBox コントロールの実行時の編集をロックするには、プロジェクトに次のコードを追加します。

Visual Basic

```
C1MaskedTextBox1.IsReadOnly = True
```

C#

```
c1MaskedTextBox1.IsReadOnly = true;
```

設計時

C1MaskedTextBox コントロールの実行時の編集をロックするには、次の手順に従います。

1. **C1MaskedTextBox** 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、**IsReadOnly** チェックボックスを「オン」にします。

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

プロジェクトの実行と確認

コントロールの編集がロックされました。コントロール内でカーソルをクリックしてみてください。コントロールにテキストの挿入ポイント（点滅する垂直線）が表示されないことがわかります。

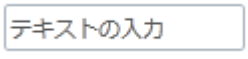
ウォーターマーク

Watermark プロパティを使用して、どのような値を **C1MaskedTextBox** コントロールに入力する必要があるかを説明する簡単なヒントをユーザーに提供できます。ウォーターマークは、テキストが入力されるまでコントロールに表示されます。ウォーターマークを追加するには、任意の **C1MaskedTextBox** コントロールの XAML マークアップで、**Watermark="ウォーターマーク・テキスト"** を **<c1:C1MaskedTextBox>**タグに追加します。

XAML

```
<c1:C1MaskedTextBox Height="23" Margin="21,46,167,0" Name="C1MaskedTextBox1"
VerticalAlignment="Top" Watermark="Enter Text" />
```

実行時に、コントロールは次のように表示されます。



コントロール内をクリックしてテキストを入力すると、ウォーターマークが消えることがわかります。

フォーマッティング

フォントタイプおよびフォントサイズの変更

C1MaskedTextBox の[プロパティ]ウィンドウ、XAML、またはコードでテキストの外観を変更できます。

XAML の場合

たとえば、XAML でコントロールのフォントを 10 ポイントの Arial に変更するには **<c1:C1MaskedTextBox>** タグに **FontFamily="Arial" FontSize="10"** を追加します。次のようなコードになります。

XAML

```
<c1:C1MaskedTextBox Height="23" HorizontalAlignment="Left" Margin="10,10,0,0"
Name="C1MaskedTextBox1" VerticalAlignment="Top" Width="120" FontSize="10"
FontFamily="Arial"></c1:C1MaskedTextBox>
```

コードの場合

たとえば、グリッドのフォントを 10 ポイントの Arial に変更するには、プロジェクトに次のコードを追加します。

Visual Basic

```
C1MaskedTextBox1.FontSize = 10
C1MaskedTextBox1.FontFamily = New System.Windows.Media.FontFamily("Arial")
```

C#

```
C1MaskedTextBox1.FontSize = 10;
C1MaskedTextBox1.FontFamily = new System.Windows.Media.FontFamily("Arial");
```

デザインの場合

設計時に[プロパティ]ウィンドウでグリッド内のフォントを Arial の 10 ポイントに変更するには、次の手順に従います。

1. **C1MaskedTextBox** 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、**FontFamily** プロパティを「**Arial**」に設定します。
3. [**プロパティ**]ウィンドウで、**FontSize** プロパティを「**10**」に設定します。

これで、コントロールのフォントサイズとフォントスタイルが設定されます。

プロジェクトの実行と確認

コントロールのコンテンツが Arial の 10 ポイントのフォントで表示されます。

Menu and ContextMenu (Silverlight のみ)

Menu for Silverlight および **ContextMenu for Silverlight**を使用すると、すべての要素を備えたメニューシステムを Silverlight アプリケーションに追加できます。**C1Menu** コントロールを使ってメニューと **C1ContextMenu** コントロールを作成し、インターフェイスにポップアップメニューをアタッチします。

Menu for Silverlight には次のコントロールが含まれます。

- **C1ContextMenu**

  C1ContextMenu は、選択されたオブジェクトに関連付けられてよく使用されるコマンドを提供するポップアップメニューを提供します

- **C1Menu**

C1Menu は、イベントハンドラに関連付けられた要素の階層構造を表すコントロールです。

Menu for Silverlight の主な特徴は次のようになります。

- **ブラウザ境界の検出**

  メニューが自動的に、常にページ境界内に収まるように配置されます。詳細については、「境界の検出」を参照してください。

- **キーボードによる移動のサポート**

  C1Menu コントロールは、キーボードによる移動をサポートしているため、Silverlight アプリケーションにアクセスしやすくなっています。

- **階層を持つメニュー**

  メニューを任意の深さにネストできます。詳細については、「サブメニューのネスト」を参照してください。

- **メニュー項目のアイコン**

  メニューでは、各メニュー項目に対してラベルのほかにアイコンを使用できます。

- **チェック可能なメニュー項目**

  オン/オフの状態でメニュー項目を示します。詳細については、「チェック可能なメニュー項目」を参照してください。

- **コンテキストメニュー**

  C1ContextMenu コントロールを使用すると、頻繁に使用されるコマンドから選択したオブジェクトへの関連付けを行うポップアップメニューを作成できます。**C1ContextMenuService** クラスは、**ToolTipService** クラスによって提供される **ToolTip** プロパティと同様に、ページの任意の FrameworkElement オブジェクトにアタッチできる拡張プロパティとしてコンテキストメニューを公開します。詳細については、「コンテキストメニューの要素」を参照してください。

- **Silverlight Toolkit テーマのサポート**

  Cosmopolitan、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。詳細については、「テーマ」を参照してください。

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。

Menu と ContextMenu のクイックスタート

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 C1Menu コントロール 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. 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、**C1Menu** 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「C1Menu」と入力します。
   c. **C1Menu** コントロールのアイコンが表示されます。

d. [**C1Menu**]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. [オブジェクトとタイムライン]タブで[C1Menu]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 プロパティを見つけ、そのグリフ(▦)をクリックして、コントロールの幅を Auto に設定します。
   - Height プロパティを見つけ、そのグリフ(▦)をクリックして、コントロールの高さを Auto に設定します。

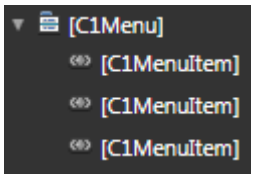
これで、Menu for Silverlight および **ContextMenu for Silverlight** クイックスタートの最初の手順は終了です。この手順では、プロジェクトを作成し、それに **C1Menu** コントロールを追加しました。次の手順では、コントロールに最上位のメニュー項目を追加します。

手順 2:メニュー項目の追加

前の手順では、C1Menu コントロールがアタッチされた Silverlight プロジェクトを作成しました。この手順では、C1Menu コントロールに3つのメニュー項目を追加します。

1. 次の手順に従って、[ツール]パネルに[C1MenuItem]アイコンを追加します。
   a. [ツール]パネルで、[アセット]ボタン(二重山かっこボタン)をクリックします。
   b. 検索バーに、「C1MenuItem」と入力して、[C1MenuItem]アイコンを表示します。
   c. [C1MenuItem]アイコンをダブルクリックして[ツール]パネルに追加します。
2. 次の手順に従って、メニューに項目を追加します。
   a. [オブジェクトとタイムライン]タブで、[C1Menu]を選択します。
   b. [ツール]パネルで、[C1MenuItem]アイコン([アセット]ボタンの下)をダブルクリックして C1Menu コントロールに C1MenuItem を追加します。

3. 手順 2を2回繰り返して、合計3つのメニュー項目を C1Menu 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]タブの階層は次のようになります。

   階層が上の図のように表示されない場合は、ドラッグアンドドロップ操作を使って C1MenuItems を並べ替えることができます。

4. [オブジェクトとタイムライン]タブで最初の[C1MenuItem]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。
   a. Width プロパティを見つけ、そのグリフ(▦)をクリックして、コントロールの幅を Auto に設定します。
   b. Height プロパティを見つけ、そのグリフ(▦)をクリックして、コントロールの高さを Auto に設定します。
   c. Header プロパティを見つけ、これを「ファイル」に設定します。
5. [オブジェクトとタイムライン]タブで2番目の[C1MenuItem]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。
   a. Width プロパティを見つけ、そのグリフ(▦)をクリックして、コントロールの幅を Auto に設定します。
   b. Height プロパティを見つけ、そのグリフ(▦)をクリックして、コントロールの高さを Auto に設定します。
   c. Header プロパティを見つけ、これを「編集」に設定します。
6. [オブジェクトとタイムライン]タブで3番目の[C1MenuItem]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。
   a. Width プロパティを見つけ、そのグリフ(▦)をクリックして、コントロールの幅を Auto に設定します。
   b. Height プロパティを見つけ、そのグリフ(▦)をクリックして、コントロールの高さを Auto に設定します。
   c. Header プロパティを見つけ、これを「追加された項目リスト」に設定します。

この手順では、Expression Blend で C1Menu コントロールに最上位のメニュー項目を追加しました。次の手順では、XAML を使用してこの2つの項目にサブメニューを追加します。

手順 3:サブメニューの追加

前の手順では、Expression Blend で最上位のメニュー項目を追加しました。この手順では、XAML を使って2つのメニュー項目にサブメニューを追加します。

1. XAML ビューに切り替えます。
2. 開始タグと終了タグができるように、最初の2つの **<c1:C1MenuItem>** タグを変更します。次のようになります。

XAML

```
<c1:C1MenuItem Height="Auto" Width="Auto" Header="File"></c1:C1MenuItem>
<c1:C1MenuItem Height="Auto" Width="Auto" Header="Edit"></c1:C1MenuItem>
```

3. [ファイル]メニュー項目にサブメニューを追加するために、**<c1:C1MenuItem Header="ファイル">** タグと **</c1:C1MenuItem>** タグの間に次のマークアップを追加します。

XAML

```
<c1:C1MenuItem Height="Auto" Width="Auto" Header="新規作成">
<c1:C1MenuItem Height="Auto" Width="Auto" Header="ドキュメント"/>
<c1:C1MenuItem Height="Auto" Width="Auto" Header="プロジェクト"/>
</c1:C1MenuItem>
<c1:C1MenuItem Height="Auto" Width="Auto" Header="開く">
<c1:C1MenuItem Height="Auto" Width="Auto" Header="ドキュメント"/>
<c1:C1MenuItem Height="Auto" Width="Auto" Header="プロジェクト"/>
<c1:C1Separator/>
<c1:C1MenuItem Header="最近使用したドキュメント 1" Height="Auto" Width="Auto"
GroupName="CheckedDocuments" IsCheckable="True" IsChecked="True" />
<c1:C1MenuItem Header="最近使用したドキュメント 2" Height="Auto" Width="Auto"
GroupName="CheckedDocuments" IsCheckable="True" />
</c1:C1MenuItem>
<c1:C1Separator/>
<c1:C1MenuItem Height="Auto" Width="Auto" Header="閉じる"/>
<c1:C1MenuItem Height="Auto" Width="Auto" Header="ソリューションを閉じる"/>
<c1:C1Separator/>
<c1:C1MenuItem Height="Auto" Width="Auto" Header="終了"/>
```

4. [Edit]メニュー項目にサブメニューを追加するために、**<c1:C1MenuItem Header="編集">** タグと **</c1:C1MenuItem>** タグの間に次のマークアップを追加します。

XAML

```
<c1:C1MenuItem Height="Auto" Width="Auto" Header="元に戻す"/>
<c1:C1MenuItem Height="Auto" Width="Auto" Header="やり直し"/>
```

この手順では、**C1Menu** コントロールの2つのメニュー項目にサブメニューを追加しました。次の手順では、**C1ContextMenu** コントロールを **C1Menu** コントロールに追加します。

手順 4:コンテキストメニューの追加

前の手順では、**C1Menu** コントロールの2つのメニュー項目にサブメニューを追加しました。この手順では、**C1ContextMenu** コントロールを **C1Menu** コントロールに追加します。このコンテキストメニューには項目が1つあり、クリックすると、「手順 2:メニュー項目の追加」で作成した **C1Menu** コントロールの最上位のメニュー[Added Items]にサブメニュー項目が追加されます。

次の手順に従います。

1. XAML ビューで、**</c1:C1Menu>** タグの直前に次の XAML マークアップを配置します。

XAML

```
<c1:C1ContextMenuService.ContextMenu>
    <c1:C1ContextMenu Width="Auto" Height="Auto">
        <c1:C1MenuItem Height="Auto" Width="Auto" Header="追加された項目リスト"
Click="C1MenuItem_AddOnClick"/>
    </c1:C1ContextMenu>
</c1:C1ContextMenuService.ContextMenu>
```

上のマークアップは、**C1ContextMenuService** ヘルパークラスを使用して、**C1Menu** コントロールに **C1ContextMenu** コントロールを追加します。**C1ContextMenu** コントロールには、**Click** イベントにアタッチされている **C1MenuItem** が1つ、**"C1MenuItem_AddOnClick"** という名前で含まれています。

2. **<c1:C1MenuItem Header="項目の追加"/>** タグに **x:Name="AddedItems"** を追加します。これにより、コードから呼び出せる一意の識別子を項目に渡すことができます。
3. **MainPage.xaml.cs** ページを開き、プロジェクトに次の **Click** イベントハンドラを追加します。

Visual Basic

```
Private Sub C1MenuItem_AddOnClick(ByVal sender As Object, ByVal
e As C1.Silverlight.SourcedEventArgs)
      Dim C1MenuItemAdd As New C1.Silverlight.C1MenuItem()
      C1MenuItemAdd.Header = "追加された項目"
      AddedItems.Items.Add(C1MenuItemAdd)
End Sub
```

C#

```
private void C1MenuItem_AddOnClick(object sender,
C1.Silverlight.SourcedEventArgs e)
    {
         C1.Silverlight.C1MenuItem C1MenuItemAdd = new
C1.Silverlight.C1MenuItem();
         C1MenuItemAdd.Header = "追加された項目";
         AddedItems.Items.Add(C1MenuItemAdd);
    }
```

この手順では、**C1Menu** コントロールに **C1ContextMenu** コントロールを追加します。次の手順では、プロジェクトを実行して **Menu for Silverlight** クイックスタートの結果を確認します。

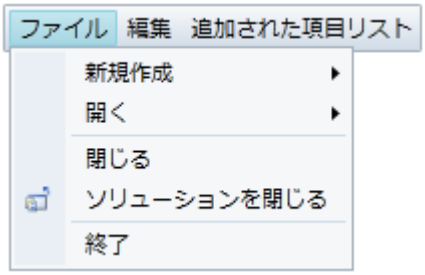
手順 5：アプリケーションの実行

C1Menu コントロールと **C1ContextMenu** コントロールを持つ Silverlight プロジェクトを作成したので、この後はプロジェクトを実行して作業の結果を確認するだけです。

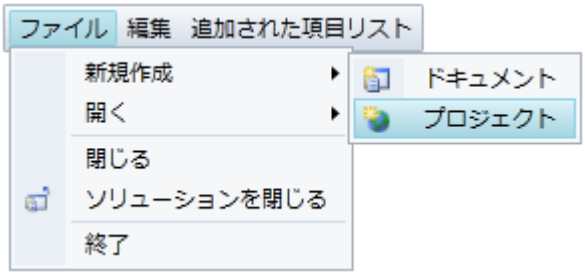
次の手順に従います。

1. ツールバーから、[**デバッグ**]→[**スタートデバッギング**]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ようになります。

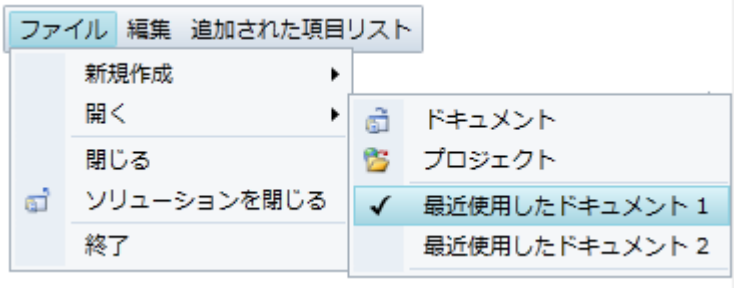
2. [**ファイル**]をクリックして、サブメニューが表示されることを確認します。



3. カーソルを[新規作成]に置き、別のサブ**メニュー**が表示されることを確認します。



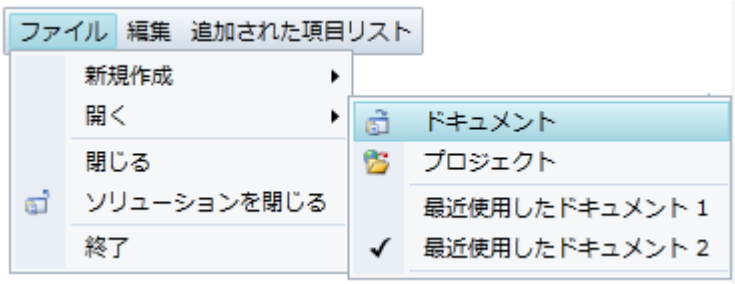
4. カーソルを[**開く**]に置き、別のサブメニューが表示されることを確認します。[**最近使用したドキュメント 1**]にチェックマークが付いています。



5. [**最近使用したドキュメント 2**]をクリックします。

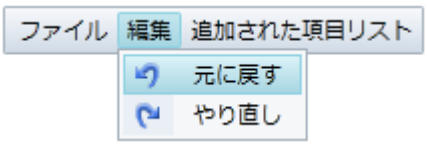
   [開く]サブメニューが閉じます。/p>

6. [**ファイル**]→[**開く**]を選択し、[**最近使用したドキュメント 1**]ではなく[**最近使用したドキュメント 2**]にチェックマークが付いていることを確認します。

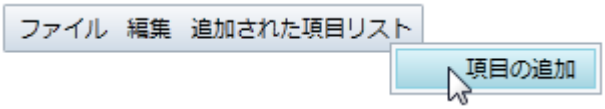


   2つのチェック可能な項目[最近使用したドキュメント 1]と[最近使用したドキュメント 2]が1つのグループにまとめられて、相互に排他的なチェック可能な項目のリストになります。この詳細については、「チェック可能なメニュー項目」を参照してください。

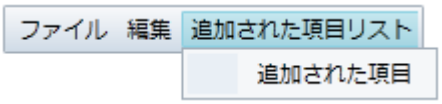
7. [**編集**]をクリックして、**[編集]サブメニュー**が表示されることを確認します。



8. [**追加された項目**]をクリックし、サブメニューがないことを確認します。
9. メニューを右クリックしてコンテキストメニューを呼び出します。
10. コンテキストメニューで、[**項目の追加**]をクリックします。



11. [**追加された項目リスト**]をクリックし、1つの新しい項目で構成される[**追加された項目**]という名前のサブメニューがあることを確認します。



おめでとうございます。

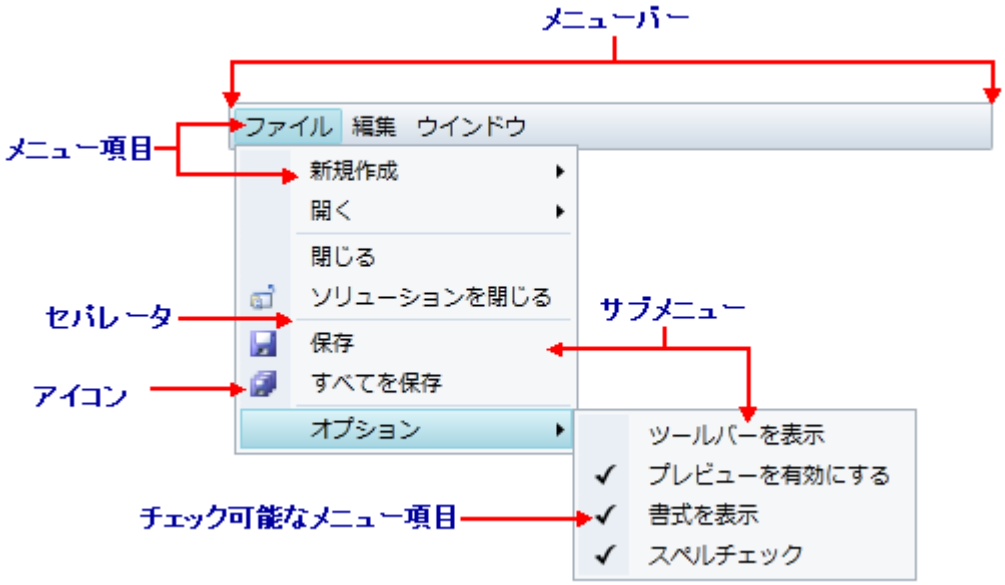
これで、**Menu for Silverlight** クイックスタートは終了です。

Menu と ContextMenuの要素

メニューの要素

C1Menu は、イベントハンドラに関連付けられた要素の階層構造を表すコントロールです。メニューは必要な任意の深さにネストでき、必要な数だけメニューに項目を追加できます。

次の図に、**C1Menu** コントロールの要素を示します。



C1Menu コントロールの要素には、次のものがあります。

- **メニューバー**

  メインメニューは、水平な最上位のメニューです。最上位の **C1MenuItem** で構成され、**C1MenuItem** のすべての機

能を利用できます。

- **サブメニュー**

  サブメニューは、他のメニューからのみアクセスできるメニューです。サブメニューは、**C1MenuItem** を別の **C1MenuItem** 内にネストすることによって作成されます。

- **メニュー項目**

  メニュー項目は、**C1MenuItem** クラスによって表されます。メニュー項目には、ラベルやアイコンを含めることができます。また、チェック可能なメニュー項目として指定することもできます。各メニュー項目は、Click イベントハンドラに関連付けられます。

- **チェック可能なメニュー項目**

  チェック可能なメニュー項目は、**C1MenuItem** クラスのオブジェクトで、ユーザーが選択またはクリアすることができます。チェック可能なメニュー項目は、スタンドアロンで使用することも、他のチェック可能なメニュー項目とグループ化して、相互に排他的なチェックボックスのリストを作成することもできます。チェック可能なメニュー項目は、メニューバーでは使用できません。

- **アイコン**

  アイコンは、Icon プロパティに Image コントロールを追加することによって作成されます。アイコンは、メニューバーでは使用できません。

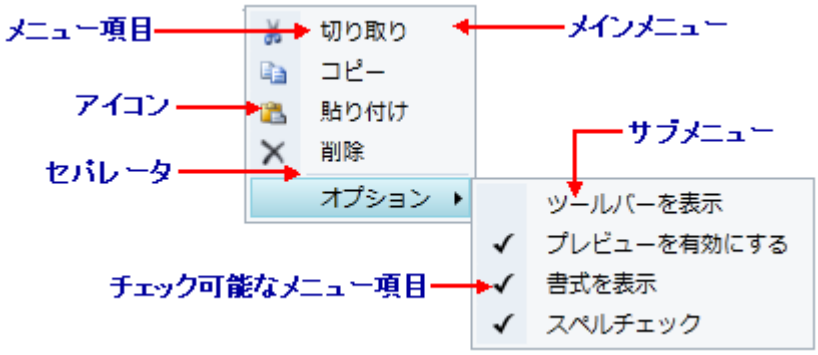
- **セパレータ**

  セパレータは、**C1Separator** クラスを介して作成されます。アイコンは、メニューバーでは使用できません。

コンテキストメニューの要素

C1ContextMenu は、選択されたオブジェクトに関連付けられてよく使用されるコマンドを提供するポップアップメニューを提供します。**C1ContextMenuService** クラスは、**ToolTipService** クラスによって提供される **ToolTip** プロパティと同様に、ページの任意の **FrameworkElement** オブジェクトにアタッチできる拡張プロパティとしてコンテキストメニューを公開します。メニューをオブジェクトにアタッチすると、ユーザーはそのオブジェクトを右クリックしてメニューを開くことができるようになります。次の図に、C1ContextMenu コントロールの要素を示します。

次の図に、**C1ContextMenu** コントロールの要素を示します。



C1Menu コントロールの要素には、次のものがあります。

- **メインメニュー**

  メインメニューは、最上位の **C1MenuItem** で構成される水平のバーです。メニュー項目は、スタンドアロンの項目にすることも、サブメニュー項目のリストを保持することもできます。

- **サブメニュー**

サブメニューは、他のメニューからのみアクセスできるメニューです。サブメニューは、**C1MenuItem** を別の **C1MenuItem** 内にネストすることによって作成されます。

- **メニュー項目**

  メニュー項目は、**C1MenuItem** クラスによって表されます。メニュー項目には、ラベルやアイコンを含めることができます。また、チェック可能なメニュー項目として指定することもできます。各メニュー項目は、Click イベントハンドラに関連付けられます。チェック可能なメニュー項目

- **チェック可能なメニュー項目**

  チェック可能なメニュー項目は、**C1MenuItem** クラスのオブジェクトで、ユーザーが選択またはクリアすることができます。チェック可能なメニュー項目は、スタンドアロンで使用することも、他のチェック可能なメニュー項目とグループ化して、相互に排他的なチェックボックスのリストを作成することもできます。

- **アイコン**

  アイコンは、Icon プロパティに Image コントロールを追加することによって作成されます。

- **セパレータ**

  セパレータは、**C1Separator** クラスを介して作成されます。メニュー項目を視覚的にグループ分けするために使用できます。

メニューを作成する

最上位のメニューを作成する

次の手順に従います。

1. **C1Menu**または**C1ContextMenu**アプリケーションを追加します。
2. **<c1:C1Menu>** タグと **</c1:C1Menu>** タグ、または **<c1:C1ContextMenu>** タグと **</c1:C1ContextMenu>** タグの間に、次の XAML を追加します。

   XAML

   ```
   <c1:C1MenuItem Header="MenuItem1" Height="Auto" Width="Auto"/>
   <c1:C1MenuItem Header="MenuItem2" Height="Auto" Width="Auto"/>
   <c1:C1MenuItem Header="MenuItem3" Height="Auto" Width="Auto"/>
   ```

3. **<c1:C1Menu>** タグまたは **<c1:C1ContextMenu>** タグに、**Height="Auto"** と **Width="Auto"** を追加します。
4. プログラムを実行し、次のいずれかを実行します。

- C1Menu コントロールを使用している場合は、3つの項目を持つメニューバーが表示されることを確認します。

  または

- C1ContextMenu コントロールを使用している場合は、コンテキストメニューがアタッチされている要素を右クリックします。3つの項目を持つメニューが表示されることを確認します。

サブメニューを作成する

このトピックでは、メニュー項目の1つにアタッチされるサブメニューを作成します。このトピックでは、少なくとも1つのメニュー項目を持つ最上位のメニューが既に作成されていることを前提としています（「**最上位のメニューを作成する**」を参照）。

次の手順に従います

1. <c1:C1Menu> タグと </c1:C1Menu> タグ、または <c1:C1ContextMenu> タグと </c1:C1ContextMenu> タグの間に、次の XAML を追加します。

XAML

```
<c1:C1MenuItem Header="ここをクリック" Height="Auto" Width="Auto">
     <c1:C1MenuItem Header="サブメニュー項目" Height="Auto" Width="Auto"/>
     <c1:C1MenuItem Header="サブメニュー項目" Height="Auto" Width="Auto"/>
     <c1:C1MenuItem Header="サブメニュー項目" Height="Auto" Width="Auto"/>
</c1:C1MenuItem>
```

2. <c1:C1Menu> タグまたは <c1:C1ContextMenu> タグに、Height="Auto" と Width="Auto" を追加します。
3. プログラムを実行し、次のいずれかを実行します。

- C1Menu コントロールを使用している場合は、[ここをクリック]をクリックします。3つの項目を持つサブメニューが表示されることを確認します。

  または

- C1ContextMenu コントロールを使用している場合は、コンテキストメニューを開くためにアタッチした要素を右クリックします。[ここをクリック]をクリックし、3つの項目を持つサブメニューが表示されることを確認します。

コンテキストメニューを作成する

このトピックでは、C1ContextMenuService クラスを使って TextBox コントロールにコンテキストメニューをアタッチします。

次の手順に従います。

1. Silverlight プロジェクトに、C1.Silverlight アセンブリへの参照を追加します。
2. ソースビューを開きます（Blend を使用している場合は XAML ビュー）。
3. タグに次のマークアップを追加して、アセンブリの名前空間をインポートします。

XAML

```
xmlns:c1="clr-namespace:C1.Silverlight;assembly=C1.Silverlight"
```

4. <Grid> タグを次のマークアップで置き換えて、プロジェクトに TextBox コントロールを追加します。

XAML

```
<Grid x:Name="LayoutRoot" Background="White">
    <TextBox Text="コンテキストメニューを開くために右クリック" Width="170" Height="25"
Background="#FFC4D8D4">
    </TextBox>
</Grid>
```

5. **<TextBox>** タグと **</TextBox>** タグの間に次のマークアップを追加して、TextBox に **C1ContextMenuService** を追加します。

XAML

```
<c1:C1ContextMenuService.ContextMenu>
</c1:C1ContextMenuService.ContextMenu>
```

C1ContextMenuService クラスは、ページの **FrameworkElement** オブジェクトにアタッチできる拡張プロパティとして、コンテキストメニューを公開します。

6. **<c1:C1ContextMenuService.ContextMenu>** タグと **</c1:C1ContextMenuService.ContextMenu>** タグの間に次の XAML を追加して、コンテキストメニューと数個のコンテキストメニュー項目を作成します。

XAML
```
<c1:C1ContextMenu Width="Auto" Height="Auto">
     <c1:C1MenuItem Header="MenuItem1"></c1:C1MenuItem>
     <c1:C1MenuItem Header="MenuItem2"></c1:C1MenuItem>
     <c1:C1MenuItem Header="MenuItem3"></c1:C1MenuItem>
</c1:C1ContextMenu>
```

7. プロジェクトを実行し、テキストボックスコントロールを右クリックします。3つの項目を持つコンテキストメニューが表示されることを確認します。

メニュー項目へアイコンを追加する

XAML とコードで C1MenuItem にアイコンを追加する方法について説明します。

XAML の場合

次の手順に従います。

1. Silverlight プロジェクトにアイコン画像を追加します。12x12 ピクセルの画像が最適です。
2. <c1:C1MenuItem> タグと </c1:C1MenuItem> タグの間に、次の XAML マークアップを追加し、Source プロパティを画像の名前に置き換えます。

XAML
```
<c1:C1MenuItem.Icon>
     <Image Source="YourImage.png" Height="12" Width="12" Margin="5,0,0,0"/>
</c1:C1MenuItem.Icon>
```

3. プロジェクトを実行します。

コードの場合

次の手順に従います

1. Silverlight プロジェクトにアイコン画像を追加します。12x12 ピクセルの画像が最適です。
2. アイコンを追加する項目に **x:Name="C1MenuItem1"** を追加します。
3. 次の名前空間をインポートします。

Visual Basic
```
Imports System.Windows.Media.Imaging
```

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

4. 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

```
画像を作成し、ソース、マージン、幅を割り当てます
Dim ItemIcon As New Image()
ItemIcon.Source = New BitmapImage(New Uri("02.png", UriKind.RelativeOrAbsolute))
ItemIcon.Margin = New Thickness(5, 0, 0, 0)
ItemIcon.Height = 12
ItemIcon.Width = 12
'新しい画像に C1MenuItem のアイコンプロパティを設定します
C1MenuItem.Icon = ItemIcon
```

C#

```
//画像を作成し、ソース、マージン、幅を割り当てます
Image ItemIcon = new Image();
ItemIcon.Source = new BitmapImage(new Uri("02.png",
UriKind.RelativeOrAbsolute));
ItemIcon.Margin = new Thickness(5,0,0,0);
ItemIcon.Height = 12;
ItemIcon.Width = 12;
//新しい画像に C1MenuItem のアイコンプロパティを設定します
C1MenuItem1.Icon = ItemIcon;
```

5. プロジェクトを実行します。

次の図は、12x12 ピクセルのアイコン付きの **C1MenuItem** です。

セパレータバーを追加する

C1ContextMenu コントロールにセパレータを追加するには、2つの **<c1:C1MenuItem>** タグの間に **<c1:C1Separator />** を配置します。結果は次のようになります。

XAML

```
<c1:C1MenuItem Header="ファイル" />
    <c1:C1Separator/>
<c1:C1MenuItem Header="オプション" />
```

Menu と ContextMenu の機能

オートクローズ

デフォルトでは、**C1ContextMenu** コントロール、そのサブメニュー、および**C1Menu** コントロールのサブメニューは、ユーザーがメニュー以外の場所をクリックしても開いたままです。メニューを閉じるには、**C1Menu** コントロールまたは **C1ContextMenu** コントロールをクリックする必要があります。ただし、オートクローズ機能を有効にすると、この動作を変更できます。これにより、ユーザーはメニューコントロールの境界の外側をクリックすることでメニューを閉じられるようになります。オートクローズをオンにするには、**AutoClose** プロパティを **"True"** に設定します。

XAML の場合

次の手順に従います。

1. **<c1:C1Menu>** タグまたは **<c1:C1ContextMenu>** タグに、**AutoClose="True"** を追加します。
2. プログラムを実行します。

コードの場合

次の手順に従います。

1. ソースビュー（Blend を使用している場合は XAML ビュー）で、チェック可能にする項目の **<c1:C1MenuItem>** タグを探し、**x:Name="C1Menu1"** を追加します。これにより、コードから使用できる一意の識別子を項目に渡すことができます。
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。

Visual Basic

```
C1Menu1.AutoClose = True
```

C#

```
C1Menu1.AutoClose = true;
```

3. プログラムを実行します。

デザインの場合

次の手順に従います。

1. **C1ContextMenu** コントロールまたは **C1Menu** コントロールを選択します。

   [**プロパティ**]ウィンドウの下にコントロールのプロパティが表示されます。

2. [**AutoClose**]チェックボックスをオンにします。
3. プログラムを実行します。

サブメニューのネスト

C1Menu コントロールと **C1ContextMenu** コントロールは、サブメニューを持つことができます。これらのサブメニューは、**C1MenuItems** が他の **C1MenuItems** タグ内にネストしていると作成されます。たとえば、次の XAML マークアップがあります。

XAML

```
<c1:C1MenuItem Header="1 番目">
    <c1:C1MenuItem Header="2 番目">
        <c1:C1MenuItem Header="3 番目">
            <c1:C1MenuItem Header="4 番目">
                <c1:C1MenuItem Header="5 番目">
</c1:C1MenuItem>
```

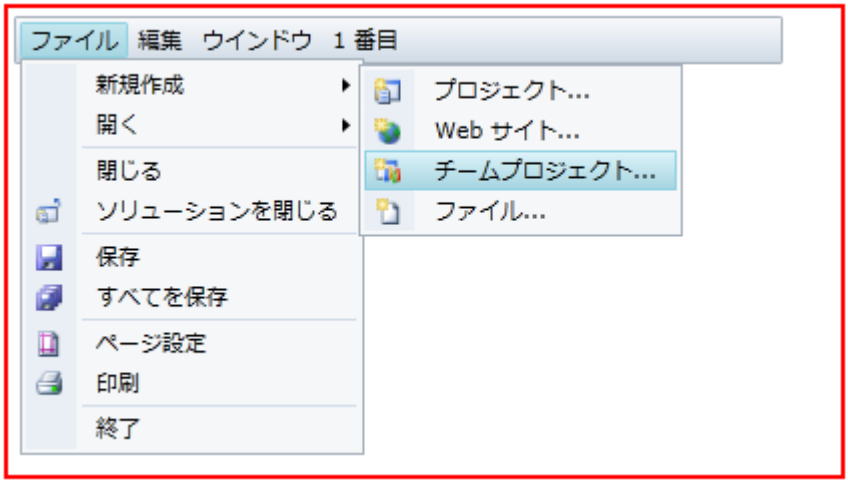
これを **C1Menu** の開始タグと終了タグの間に配置すると、次のようなに作成されます。

必要な数だけメニューをネストできますが、使いやすさの観点から、1つの階層にサブメニューを2、3個以上は置かない方がよいでしょう。

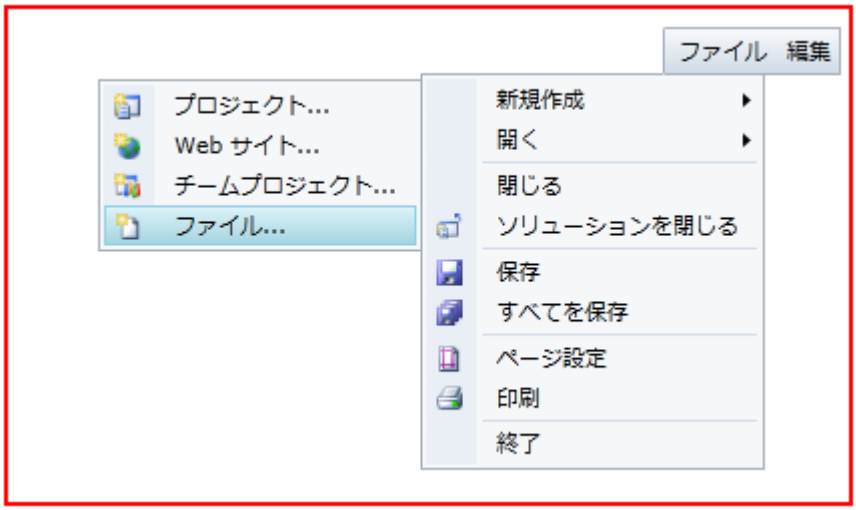
境界の検出

境界の検出を使用すると、ユーザーが開いたメニューは常にページ境界内部に収まります。この機能は、メニュー内に複数のネストしたサブメニューがある場合に、メニューをプロジェクトページの範囲いっぱいまで広げることができるという点で便利です。

次の図では、メニューの境界に2つのサブメニューを広げられる十分な広さがあります。



次の図は、メニューと境界は同じで、今回は境界の検出を有効にしただけです。サブメニューが左側に移動することで、近い位置にあるページの右側の境界で切られることを避けています。



デフォルトでは、境界の検出は無効ですが、**DetectBoundaries** プロパティを **"True"** に設定することによって有効にできます。

境界の検出を有効にすることに関するタスク別ヘルプについては、「境界の検出を有効にする」を参照してください。

XAML の場合

次の手順に従います。

1. **<c1:C1Menu>** タグまたは **<c1:C1ContextMenu>** タグに、**DetectBoundaries="True"** を追加します。
2. プログラムを実行します。

コードの場合

次の手順に従います。

1. ソースビューで、チェック可能にする項目の**<c1:C1MenuItem>** タグを探し、**x:Name="C1Menu1"** を追加します。これにより、コードから使用できる一意の識別子を項目に渡すことができます。
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。

Visual Basic

```
C1Menu1.DetectBoundaries = True
```

C#

```
C1Menu1.DetectBoundaries = true;
```

3. プログラムを実行します。

設計時間

次の手順に従います。

1. **C1ContextMenu** コントロールまたは **C1Menu** コントロールを選択します。

   [**プロパティ**]ウィンドウの下にコントロールのプロパティが表示されます。

2. [**DetectBoundaries**]チェックボックスをオンにします。
3. プログラムを実行します。

チェック可能なメニュー項目を作成する

IsCheckable プロパティを "True" に設定すると、**C1MenuItem** をチェック可能なメニュー項目にすることができます。**IsChecked** プロパティの値は、メニュー項目がオンになっているかどうかを決定します。デフォルトでは、**IsChecked** プロパティは False です。項目をクリックすると、**IsChecked** プロパティは True になります。

グループに追加する各項目の **GroupName** プロパティを設定すると、相互に排他的なチェック可能な項目のグループを作成できます。たとえば、下の XAML は、相互に排他的な3つのチェック可能な項目のグループを作成します。

XAML

```
<c1:C1MenuItem Header="MenuItem1" IsCheckable="True"
GroupName="MutuallyExclusiveGroup"/>
<c1:C1MenuItem Header="MenuItem2" IsCheckable="True"
GroupName="MutuallyExclusiveGroup"/>
<c1:C1MenuItem Header="MenuItem3" IsCheckable="True"
GroupName="MutuallyExclusiveGroup"/>
```

チェック可能なメニュー項目を作成するには、次の方法のいずれかを使用できます。

XAML の場合

次の手順に従います。

1. チェック可能にするメニュー項目の **<c1:C1MenuItem>** タグを探し、タグに **IsCheckable="True"** を追加します。XAML は次のようになります。

   XAML

   ```
   <c1:C1MenuItem Header="C1MenuItem" IsCheckable="True"/>
   ```

   オプション：実行時にチェック済みの状態にするために、タグに **IsChecked="False"** を追加することもできます。

2. プロジェクトを実行します。

コードの場合

次の手順に従います

1. ソースビューで、チェック可能にする項目の **<c1:C1MenuItem>** タグを探し、**x:Name="CheckableMenuItem"** を追加します。これにより、コードから使用できる一意の識別子を項目に渡すことができます。
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。

   Visual Basic

   ```
   CheckableMenuItem.IsCheckable = True
   ```

   C#

   ```
   CheckableMenuItem.IsCheckable = true;
   ```

   オプション：実行時にチェック済みの状態にするために、プロジェクトに次のコードを追加します。

Visual Basic

```
CheckableMenuItem.IsChecked = True
```

C#

```
CheckableMenuItem.IsChecked = true;
```

3. プログラムを実行します。

設計時

次の手順に従います。

1. ソースビューで、チェック可能にしたい C1MenuItem を選択します。
2. [**プロパティ**]ウィンドウで、[**IsCheckable**]チェックボックスをオンにして **IsCheckable** プロパティを **"True"** に設定します。これにより、実行時に項目がチェック可能になります。

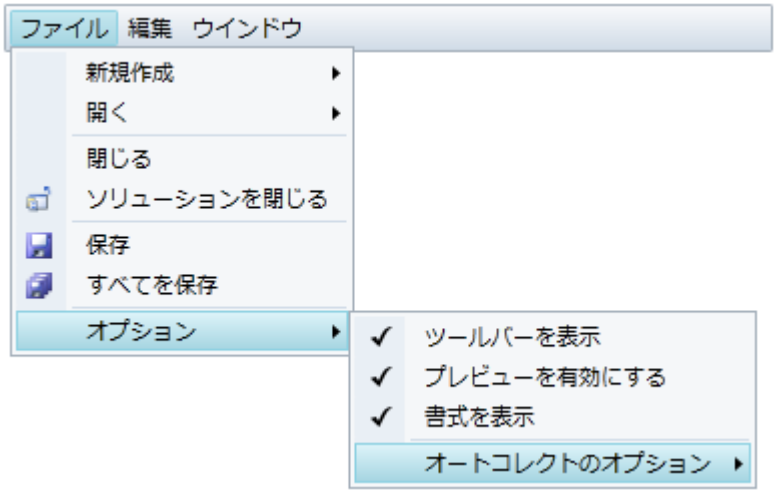
   オプション：実行時に項目をオンにするには、[**IsChecked**]チェックボックスをオンにしますが、これは項目をチェック可能にするために必要な作業ではありません。
3. プロジェクトを実行します。

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。

> **メモ**：このトピックでは、**C1Menu** コントロールのテーマについてのみ説明します。ただし、**C1ContextMenu**コントロールは、**C1Menu** コントロールのサブメニューのテーマと同じです。

プロジェクトに **C1Menu** コントロールを追加すると、コントロールはデフォルトの青いテーマで表示されます。



C1Menu コントロールには、付属している Silverlight の6つのテーマ（BureauBlack、Cosmopolitan、ExpressionDark、ExpressionLight、RainierOrange、ShinyBlue、WhistlerBlue）の中から1つテーマを設定できます。

メニューコントロールに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。

メニューのテーマを使用する

次の手順に従います。

1. Visual Studio で、.xaml ページを開きます。
2. **<Grid></Grid>** 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。

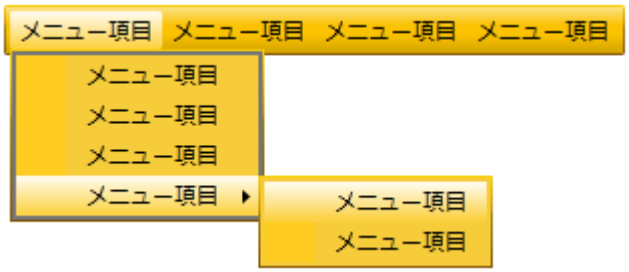
```
XAML
<my:C1ThemeRainierOrange></my:C1ThemeRainierOrange>
```

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、[**C1Menu**]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 **<my:C1ThemeRainierOrange>** 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:C1Menu></c1:C1Menu>
</my:C1ThemeRainierOrange>
```

6. プロジェクトを実行します。

次の図は、**C1ThemeRainierOrange** テーマが使用された **C1Menu** コントロールです。



コンテキストメニューのテーマを使用する

C1ContextMenu コントロールは、サービスを使用することによって必ず別のコントロールにアタッチされるので、**C1ContextMenu** コントロールへのテーマの追加は、他の Silverlight コントロールへのテーマの追加とわずかながら異なります。グリッドまたはパネルにテーマを適用し、暗黙のスタイルマネージャを使用して、テーマが **C1ContextMenu** コントロールに渡されるようにします。

次の手順に従います。

1. Visual Studio で、.xaml ページを開きます。
2. **<Grid></Grid>** 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。

```
XAML
<my:C1ThemeRainierOrange></my:C1ThemeRainierOrange>
```

4. **<my:C1ThemeRainierOrange>** タグと **</my:C1ThemeRainierOrange>** タグの間にカーソルを置き、[Enter] キーを押します。[ツール]パネルで、[Grid]アイコンをダブルクリックします。次のように、グリッドタグがテーマタグの間に表示されます。

5. [**ツール**]パネルで、[**Grid**]アイコンをダブルクリックします。次のように、グリッドタグがテーマタグの間に表示されます。

```
XAML
<my:C1ThemeRainierOrange>
    <Grid></Grid>
</my:C1ThemeRainierOrange>
```

6. **<Grid>** タグと **</Grid>** タグの間に **C1ContextMenu** コントロールを追加します

7. 次の名前空間を **<UserControl>** タグに追加します。

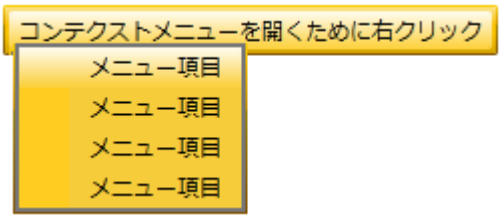
```
XAML
xmlns:c1Theming="clr-
namespace:C1.Silverlight.Theming;assembly=C1.Silverlight.Theming"
```

8. **c1Theming:ImplicitStyleManager.ApplyMode="Auto"** を **<c1:C1Accordion>** タグに追加して、**ApplyMode** プロパティを設定します。マークアップは次のようになります。

```
XAML
<c1:C1ContextMenu c1Theming:ImplicitStyleManager.ApplyMode="Auto">
```

9. プロジェクトを実行します。

次の図は、**C1ThemeRainierOrange** テーマが使用された **C1ContextMenu** コントロールです。



C1ScrollViewer の操作

C1ScrollViewerは、標準の ScrollViewer コントロールの代わりに **C1Menu** で使用されます。**C1ScrollViewer** は、水平と垂直の両方向へのスクロールをサポートしています。**C1ScrollViewer** には、標準の ScrollViewer コントロールに勝る次のようなメリットがあります。

- **ClearStyle のサポート**

  ComponentOne ClearStyle を使用すると、いくつかのプロパティを変更するだけで、コントロールの外観を簡単にカスタマイズすることができます。

- **統一された独自のスタイル**

  C1ScrollViewer コントロールは、標準のコントロールとは異なる独自のスタイルを持ちます。さらに、**C1ScrollViewer** コントロールは、**ComponentOne for Silverlight** と **ComponentOne for WPF** の両方で同じスタイルを使用します。標準の ScrollViewer は、この2つのプラットフォームで異なるスタイルを使用しています。

- **マウスオーバー時のスクロール**

  C1ScrollViewer は、実行時にマウスオーバーでスクロールすることができます。この機能は、**ScrollMode** プロパティを使用してカスタマイズできます。

C1ScrollViewer 要素は、1つのコンテンツ要素と最大2つの ScrollBar コントロールをカプセル化します。その範囲には、**C1ScrollViewer** のすべてのコンテンツが含まれます。コンテンツの可視領域がビューポートです。

HorizontalScrollBarVisibility プロパティと **VerticalScrollBarVisibility** プロパティは、垂直と水平の ScrollBar コントロールが表示される条件を制御します。

NumericBox

NumericBox for WPF/Silverlight アプリケーションは、数値テキストボックスコントロールとして **NumericBox for WPF/Silverlight** が用意されています。このコントロールは、標準の Windows Forms **NumericUpDown** コントロールに似ており、初期設定のまま数値入力/編集機能を提供します。

C1NumericBox コントロールは1つの数値を保持し、コントロールのアップ/ダウンボタンをクリックすることで、この数値をインクリメント/デクリメントできます。**IsReadOnly** プロパティが **True** に設定されていない場合は、ユーザーが値を入力することもできます。

NumericBox for WPF/Silverlight の主な特徴は次のようになります。

- **柔軟な書式設定**

  Format プロパティを使用すると、使い慣れた .NET 書式文字列でデータを思いどおりに表示できます。

- **数値範囲のサポート**

  ユーザーが入力できる最大値と最小値を簡単に変更できます。

- **アップ/ダウンボタン**

  C1NumericBox コントロールには、値をインクリメントとデクリメントするためのアップ/ダウンボタンがあります。

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

NumericBoxクイックスタート

手順 1:コントロールの追加

この手順では、最初に Visual Studio で **NumericBox for WPF/Silverlight** を使用するアプリケーションを作成します。**C1NumericBox** コントロールをアプリケーションに追加するだけで、完全な機能を備えた数値エディタとして使用できます。さらに、そのコントロールをアプリケーションに合わせてカスタマイズできます。

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. ツールボックスに移動し、[**C1NumericBox**]アイコンをダブルクリックして、MainWindow または Main page にコントロールを追加します。
3. **C1NumericBox1** コントロールをクリックして選択し、[**プロパティ**]ウィンドウに移動します。
4. **[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 = 40

- Minimum = 0
- Maximum = 9

Width プロパティは、コントロールのサイズを変更します。**Minimum** プロパティと **Maximum** プロパティは、コントロールに入力できる最小値と最大値を設定します。データ検証機能が組み込まれているため、ユーザーはこの範囲外の値を入力できなくなります。

5. [デザイン]ビューで、C1NumericBox1 コントロールを右クリックし、[コピー]を選択します。
6. ウィンドウを右クリックし、[貼り付け]を選択して同じ設定で C1NumericBox2 コントロールを作成します。
7. 手順5と6をさらに3回繰り返して、合計5つの C1NumericBox コントロールを作成します。
8. [**デザイン**]ビューで、各コントロールが **C1NumericBox1** から **C1NumericBox5** の番号順に左から右に並んで表示されるように再配置します。

   アプリケーションは次のように表示されます。

これで、WPF/Silverlight アプリケーションを作成し、アプリケーションに **C1NumericBox** コントロールを追加し、これらのコントロールをカスタマイズできました。次の手順では、アプリケーションの設定を完成させます。

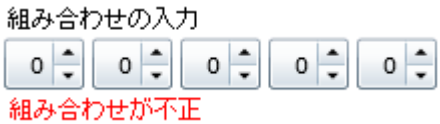
手順 2：アプリケーションのカスタマイズ

前の手順では、新しい WPF/Silverlight プロジェクトを作成し、アプリケーションに5つの **C1NumericBox** コントロールを追加しました。この手順では、引き続き、コントロールを追加してアプリケーションをカスタマイズします。

次の手順に従います。

1. Visual Studio のツールボックスに移動し、標準の **Label** コントロールを2回ダブルクリックして、**Label1** と **Label2** をプロジェクトに追加します。
2. Visual Studio のツールボックスで、標準の **Button** コントロールをダブルクリックして **Button1** をプロジェクトに追加します。
3. **Label1** を1回クリックして選択し、[**プロパティ**]ウィンドウでその **Content** プロパティを「**組み合わせを入力してください。**」に設定します。
4. **Label2** を1回クリックして選択し、**[プロパティ]**ウィンドウでその **Content** プロパティを「**不正な組み合わせ！**」に設定し、**Foreground** プロパティを **Red** に設定します。
5. **Button1** を1回クリックして選択し、[**プロパティ**]ウィンドウでその **Content** プロパティを「**継続**」に設定し、**Visibility** プロパティを **Hidden** に設定します。

アプリケーションは次のように表示されます。



これで、アプリケーションのユーザーインターフェイスを設定できました。次の手順では、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、**C1NumberBox**、**Label**、**Button** の各コント

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. **Button1** をダブルクリックしてコードビューに切り替え、**Button1_Click** イベントハンドラを作成します。
2. 次の imports 文をページの先頭に追加します。

Visual Basic

```
Imports C1.WPF
Imports System.Windows.Media
Imports System.Diagnostics
```

C#

```
using C1.WPF;
using System.Windows.Media;
using System.Diagnostics;
```

3. **MainWindow** or **Page** クラスのすぐ内側にある次のグローバル変数を初期化します。

Visual Basic

```
Dim nb1 As Integer = 5
Dim nb2 As Integer = 2
Dim nb3 As Integer = 3
Dim nb4 As Integer = 7
Dim nb5 As Integer = 9
```

C#

```
int nb1 = 5;
int nb2 = 2;
int nb3 = 3;
int nb4 = 7;
int nb5 = 9;
```

これらの数字は、正しい「暗証番号」としてアプリケーションで使用されます。ユーザーが実行時に正しい番号の組み合わせを入力すると、ボタンが表示されます。

4. **Button1_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 btn1_Click(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles btn1.Click
    HtmlPage.PopupWindow(New Uri("http://www.componentone.com"), "new", Nothing)
End Sub
```

C#

```
private void btn1_Click(object sender, RoutedEventArgs e)
{
    HtmlPage.PopupWindow(New Uri("http://www.componentone.com"), "new", null);
}
```

5. 次に、以下のカスタム **NBValidation** イベントをコードに追加します。

Visual Basic

```
Private Sub NBValidation()
    If Me.c1nb1.Value = nb1 And Me.c1nb2.Value = nb2 And Me.c1nb3.Value = nb3
And Me.c1nb4.Value = nb4 And Me.c1nb5.Value = nb5 Then
        Me.lbl2.Foreground = New SolidColorBrush(Colors.Green)
        Me.lbl2.Content = "Combination Valid"
        Me.c1nb1.IsReadOnly = True
        Me.c1nb2.IsReadOnly = True
        Me.c1nb3.IsReadOnly = True
        Me.c1nb4.IsReadOnly = True
        Me.c1nb5.IsReadOnly = True
        Me.btn1.Visibility = Windows.Visibility.Visible
    End If
End Sub
```

C#

```
private void NBValidation()
{
    if (this.c1nb1.Value == nb1 & this.c1nb2.Value == nb2 & this.c1nb3.Value ==
nb3 & this.c1nb4.Value == nb4 & this.c1nb5.Value == nb5)
    {
        this.lbl2.Foreground = new SolidColorBrush(Colors.Green);
        this.lbl2.Content = "Combination Valid";
        this.c1nb1.IsReadOnly = true;
        this.c1nb2.IsReadOnly = true;
        this.c1nb3.IsReadOnly = true;
        this.c1nb4.IsReadOnly = true;
        this.c1nb5.IsReadOnly = true;
        this.btn1.Visibility = Windows.Visibility.Visible;
    }
}
```

ユーザーが（上の手順 3で示した）正しい番号を入力すると、**C1NumericBox** コントロールは読み取り専用に設定されて編集できなくなります。コントロールの下にあるラベルのテキストは、正しいコードが入力されていることを示すように変更されます。また、ComponentOne Web サイトに移動するためのボタンが表示されます。

6. **C1NumericBox1_ValueChanged** イベントハンドラにコードを入力して、**NBValidation** を初期化します。次のようになります。

Visual Basic

```
Private Sub c1nb1_ValueChanged(ByVal sender As System.Object, ByVal e As
C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
c1nb1.ValueChanged
    NBValidation()
End Sub
Private Sub c1nb2_ValueChanged(ByVal sender As System.Object, ByVal e As
C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
c1nb2.ValueChanged
    NBValidation()
End Sub
Private Sub c1nb3_ValueChanged(ByVal sender As System.Object, ByVal e As
C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
```

```
c1nb3.ValueChanged
    NBValidation()
End Sub
Private Sub c1nb4_ValueChanged(ByVal sender As System.Object, ByVal e As
C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
c1nb4.ValueChanged
    NBValidation()
End Sub
Private Sub c1nb5_ValueChanged(ByVal sender As System.Object, ByVal e As
C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
c1nb5.ValueChanged
    NBValidation()
End Sub
```

C#

```
private void c1nb1_ValueChanged(object sender, PropertyChangedEventArgs<double>
e)
{
    NBValidation();
}
private void c1nb2_ValueChanged(object sender, PropertyChangedEventArgs<double>
e)
{
    NBValidation();
}
private void c1nb3_ValueChanged(object sender, PropertyChangedEventArgs<double>
e)
{
    NBValidation();
}
private void c1nb4_ValueChanged(object sender, PropertyChangedEventArgs<double>
e)
{
    NBValidation();
}
private void c1nb5_ValueChanged(object sender, PropertyChangedEventArgs<double>
e)
{
    NBValidation();
}
```

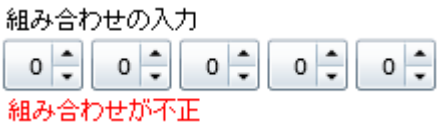
この手順では、アプリケーションにコードを追加しました。次の手順では、アプリケーションを実行し、実行時の操作を確認します。

手順 4:アプリケーションの実行

これまでに WPF/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、**NumericBox for WPF/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、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. 最初（左端）の **C1NumericBox** コントロールに**5**が表示されるまで、このコントロールの**アップ**ボタンをクリックします。クリックするたびに数字が1だけインクリメントされます。これは、**Increment** プロパティがデフォルトで1に設定されているためです。

3. 2番目の **C1NumericBox** の内部をクリックすると、「0」値が強調表示されます。「2」を入力して数字を書き換えます。

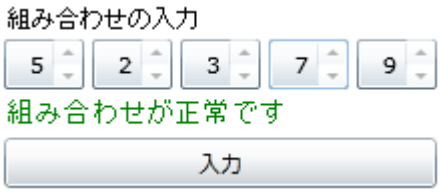
4. 3番目の **C1NumericBox** コントロールで**ダウン**ボタンをクリックしてみると、数字が変わらないことがわかります。これは、**Minimum** プロパティが0に設定されているためです。コントロールは0未満の値を受け付けません。3が表示されるまで**アップ**ボタンをクリックします。

5. 4番目の **C1NumericBox** コントロールで、0の前にカーソルを置き、クリックします。「5」を入力すると、「50」と表示されます。

6. 最後の **C1NumericBox** コントロールの内部をクリックします。4番目の **C1NumericBox** の 50 が9にリセットされることがわかります。これは、**Maximum** プロパティが9に設定されており、コントロールが9より大きい値を受け付けないためです。

7. 最後の **C1NumericBox** コントロールに9を入力します。

8. 4番目の **C1NumericBox** コントロールの**ダウン**ボタンを2回クリックすると、7が表示されます。2番目の **Label** のテキストが変化し、ボタンが表示されます。



9. **C1NumericBox** コントロールへの入力を試みたり、**アップ**ボタンや**ダウン**ボタンをクリックしても、入力は受け付けられません。これは、すべて正しい暗証番号を入力したときに **IsReadOnly** プロパティが **True** に設定され、コントロールの編集がロックされたためです。

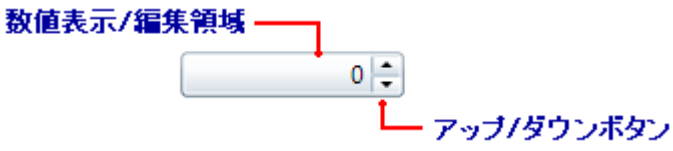
10. 表示された[**Enter**]ボタンをクリックすると、ComponentOne Web サイトに移動します。

おめでとうございます!

これで、**NumericBox for WPF/Silverlight** クイックスタートは完了です。**NumericBox for WPF/Silverlight** アプリケーションを作成し、コントロールの外観と動作をカスタマイズし、アプリケーションの実行時機能をいくつか確認することができました。

NumericBox要素

NumericBox for WPF/Silverlight には **C1NumericBox** コントロールがあります。これは、数値の入力と編集を提供するシンプルなコントロールです。XAML ウィンドウに追加された **C1NumericBox** コントロールは、完全な機能を備えた数値エディタになります。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。



コントロールは次の要素で構成されます。

- **アップ/ダウンボタン**

ユーザーは、**アップ**ボタンと**ダウン**ボタンを使用して、コントロールに表示される値を変更できます。ボタンをクリックするたびに、**Increment** プロパティで指示された値（デフォルトでは1）だけ **Value** が変化します。デフォルトでは、**アップ**ボタンと**ダウン**ボタンが表示されます。これらのボタンを非表示にするには、**ShowButtons** プロパティを **False** に設定します。

- **数値表示/編集領域**

  Value プロパティの値は、数値表示/編集領域に表示されます。ユーザーは、ボックスに入力して、**Value** プロパティを変更できます。デフォルトでは、ユーザーがこの数値を編集できます。コントロールの編集をロックするには、**IsReadOnly** を **True** に設定します。

数値書式設定

C1NumericBox コントロールに表示される数値の表示方法を変更できます。それには、**Format** プロパティを設定します。**NumericBox for WPF/Silverlight** は、Microsoft によって定義される標準の数値書式文字列をサポートします。詳細については、「MSDN」を参照してください。

Format 文字列は、書式を定義する英字または英字と数字の組み合わせで構成されます。デフォルトでは、**Format** プロパティは「**F0**」に設定されています。英字は書式タイプを指示します。この「F」は固定小数点を表し、数字は小数点以下の桁数を表します（この場合はなし）。

次の書式を使用できます。

書式指定子	名前	説明
C または c	Currency	数値は、金額を表す文字列に変換されます。変換は、現在の NumberFormatInfo オブジェクトの通貨書式情報によって制御されます。 精度指定子は、小数点以下の桁数を指示します。精度指定子を省略すると、現在の NumberFormatInfo オブジェクトで指定されるデフォルトの通貨精度が使用されます。
D または d	Decimal	この書式は、整数型でのみサポートされます。数値は 10 進数字（0～9）から成る文字列に変換されます。数字が負の場合は、先頭にマイナス記号が付きます。 精度指定子は、結果の文字列に必要な最小桁数を指示します。精度指定子によって指定された桁数になるように、必要に応じて、数字の左側がゼロで埋められます。 次の例は、Decimal 書式指定子で Int32 値を書式設定しています。
E または e	Scientific（指数）	数値は、「-d.ddd…E+ddd」または「-d.ddd…e+ddd」形式の文字列に変換されます。「d」はそれぞれ数字（0～9）を示します。数値が負である場合は、文字列の先頭にマイナス記号（-）が付加されます。小数点の前には、常に1桁の数字が置かれます。 精度指定子は、小数点以下の桁数を指示します。精度指定子を省略すると、デフォルトで小数点以下6桁が使用されます。 書式指定子の大文字と小文字は、指数部分の前に「E」と「e」のどちらを付けるかを指示します。指数は、常にプラス記号（+）またはマイナス記号（-）と3桁以上の数字で表されます。この形式でない場合は、必要に応じて指数に0が付加されます。
F または f	固定小数点	数値は、「-ddd.ddd…」形式の文字列に変換されます。「d」はそれぞれ数字（0～9）を示します。数値が負である場合は、文字列の先頭にマイナス記号（-）が付加されます。 精度指定子は、小数点以下の桁数を指示します。精度指定子を省略すると、現在の NumberFormatInfo オブジェクトの NumberDecimalDigits プロパティによってデフォルトの数値精度が指定されます。
G また	General	数値の型と精度指定子の有無に応じて、固定小数点または指数表記のいずれか簡単な方の形

は g		式に数値が変換されます。精度指定子を省略するかOにした場合は、次のリストに示すように、数値の型によってデフォルトの精度が決定されます。 • Byte または SByte: 3 • Int16 または UInt16: 5 • Int32 または UInt32: 10 • Int64: 19 • UInt64: 20 • Single: 7 • Double: 15 • Decimal: 29 数値を指数表記で表現する場合の指数部が -5 より大きく、精度指定子より小さい場合は、固定小数点表記が使用されます。それ以外の場合は、指数表記が使用されます。必要に応じて、結果には小数点が含まれ、末尾のOは省略されます。精度指定子があり、結果の有効桁数が指定された精度を超えている場合は、数値が丸められて末尾の余分な桁が除かれます。 上の規則の例外として、数値が Decimal で、精度指定子が省略されている場合があります。その場合は、常に固定小数点表記が使用され、末尾のOは保持されます。 指数表記を使用する場合は、書式指定子が「G」なら結果の指数部の前に「E」が付き、書式指定子が「g」なら「e」が付きます。指数部は少なくとも2桁あります。これは、「E」または「e」書式指定子によって生成される指数表記の書式（指数部は3桁以上）とは異なります。
N または n	Number	数値は、「-d,ddd,ddd.ddd…」形式の文字列に変換されます。「-」は負数の記号（必要に応じて）、「d」は数字（0～9）、「,」は数値の区切り記号、「.」は小数点記号を示します。実際の負数パターン、桁区切りの間隔、区切り記号、および小数点はそれぞれ、現在の NumberFormatInfo オブジェクトの NumberNegativePattern、NumberGroupSizes、NumberGroupSeparator、および NumberDecimalSeparator プロパティによって指定されます。 精度指定子は、小数点以下の桁数を指示します。精度指定子を省略すると、現在の NumberFormatInfo オブジェクトの NumberDecimalDigits プロパティによってデフォルトの数値精度が指定されます。
P または p	Percent	数値は、NumberFormatInfo.PercentNegativePattern プロパティ（数値が負の場合）または NumberFormatInfo.PercentPositivePattern プロパティ（数値が正の場合）で定義されたとおりに、パーセントを表す文字列に変換されます。変換された数値は 100 倍されて、パーセンテージで示されます。 精度指定子は、小数点以下の桁数を指示します。精度指定子を省略すると、現在の NumberFormatInfo オブジェクトで指定されるデフォルトの数値精度が使用されます。
R または r	ラウンドトリップ	この書式は、Single 型と Double 型でのみサポートされます。ラウンドトリップ指定子を指定すると、文字列に変換された数値が再度同じ数値に解析されます。この指定子を使って書式設定された数値は、まず、Double 型の場合は 15 スペースの精度、Single 型の場合は7スペースの精度の汎用の書式でテストされます。値が再度正しく同じ数値に解析された場合は、汎用の書式指定子を使って書式設定されます。ただし、値が再度同じ数値に解析されなかった場合は、Double 型の場合は 17 桁の精度、Single 型の場合は9桁の精度で書式設定されます。 精度指定子は指定してもかまいませんが、無視されます。この指定子を使用する場合は、精度よりラウンドトリップが優先されます。
X または	Hexadecimal	この書式は、整数型でのみサポートされます。数値が 16 進数字の文字列に変換されます。書式

x		指定子の大文字と小文字は、9より大きい 16 進数字に大文字を使用するか小文字を使用するかを指示します。たとえば、「X」を使用すると「ABCDEF」になり、「x」を使用すると「abcdef」になります。 精度指定子は、結果の文字列に必要な最小桁数を指示します。精度指定子によって指定された桁数になるように、必要に応じて、数字の左側がゼロで埋められます。
その他の文字	(不明な指定子)	(不明な指定子は実行時に FormatException を生成します)

入力検証

Minimum プロパティと **Maximum** プロパティを使用して、実行時の制限になる数値範囲を設定できます。**Minimum** プロパティや **Maximum** プロパティを設定すると、ユーザーは **Maximum** より大きい値や **Minimum** より小さい値を選択できなくなります。

Minimum および **Maximum** プロパティを設定する場合は、**Minimum** を **Maximum** より小さくする必要があります。また、**Value** プロパティには、**Minimum** と **Maximum** の範囲内の値を設定してください。

RangeValidationMode プロパティを使って範囲検証モードを選択することもできます。この**プロパティは**、入力された数字をいつ検証するかを制御します。**RangeValidationMode** には、次のオプションの1つを設定できます。

オプション	説明
Always	このモードでは、ユーザーは範囲外の値を入力できません。
AlwaysTruncate	このモードでは、ユーザーは範囲外の値を入力できません。制限を超えた値は切り捨てられます。
OnLostFocus	このモードでは、コントロールがフォーカスを失ったときに値が切り捨てられます。

値の設定

開始値

Value プロパティは、現在選択されている値を決定します。デフォルトでは、**C1NumericBox** コントロールは、最初に **Value** が 0 に設定されますが、設計時、XAML、またはコードでこの値をカスタマイズできます。

XAML の場合

たとえば、**Value** プロパティを設定するには、次のように示すように **Value="123"** を **<c1:C1NumericBox>** タグに追加します。次のようになります。

XAML

```
<c1:C1NumericBox Name="C1NumericBox1" Value="123" />
```

コードの場合

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

Visual Basic

```
C1NumericBox1.Value = 123
```

C#

```
c1NumericBox1.Value = 123;
```

設計時

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

1. **C1NumericBox** 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、**Value** プロパティの横にあるテキストボックスに値(たとえば、**123**)を入力します。

   これで、Value プロパティは指定された値に設定されます。

インクリメント値の設定

Increment プロパティは、実行時に**アップ**ボタンまたは**ダウン**ボタンを1回クリックしたときの **Value** プロパティの変化量を決定します。デフォルトでは、**C1NumericBox** コントロールは、最初に **Increment** が 1 に設定されますが、設計時、XAML、またはコードでこの値をカスタマイズできます。

XAML の場合

たとえば、**Increment** プロパティを 20 に設定するには、**<c1:C1NumericBox>** タグに **Increment="20"** を追加します。次のようになります。

XAML

```
<c1:C1NumericBox Name="C1NumericBox1" Increment="20" />
```

コードの場合

たとえば、Increment プロパティを 20 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

```
C1NumericBox1.Increment = 20
```

C#

```
c1NumericBox1.Increment = 20;
```

設計時

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

1. **C1NumericBox** 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、**Increment** プロパティの横にあるテキストボックスに値(たとえば、20)を入力します。

   これで、**Increment** プロパティは指定された値に設定されます。

最小値および最大値の設定

Minimum プロパティと **Maximum** プロパティを使用して、実行時の制限になる数値範囲を設定できます。**Minimum** プロパティや **Maximum** プロパティを設定すると、ユーザーは **Maximum** より大きい値や **Minimum** より小さい値を選択できなくなります。

XAML の場合

XAML で **Minimum** と **Maximum** を設定するには、**Maximum="500" Minimum="-500"** を **<c1:C1NumericBox>** タグに追加します。次のようになります。

XAML

```
<c1:C1NumericBox Name="C1NumericBox1" Maximum="500" Minimum="-500" />
```

コードの場合

Minimum と **Maximum** を設定するには、プロジェクトに次のコードを追加します。

Visual Basic

```
C1NumericBox1.Minimum = -500
C1NumericBox1.Maximum = 500
```

C#

```
c1NumericBox1.Minimum = -500;
c1NumericBox1.Maximum = 500;
```

設計時

設計時に **Minimum** と **Maximum** を設定するには、次の手順に従います。

1. **C1NumericBox** 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、**Maximum** プロパティの横に値(たとえば、**500**)を入力します。
3. **[プロパティ]**ウィンドウで、**Minimum** の横に値(たとえば、**-500**)を入力します。

これで、**Minimum** と **Maximum** の値が設定されます。

フォントおよびフォントサイズの変更

グリッド内のテキストの外観を変更するには、**C1NumericBox** の[プロパティ]ウィンドウ、XAML、またはコードでテキストプロパティを使用します。

XAML の場合

たとえば、XAML でコントロールのフォントを 10 ポイントの Arial に変更するには **<c1:C1NumericBox>** タグに **FontFamily="Arial" FontSize="10"** を追加します。次のようなコードになります。

XAML

```
<c1:C1NumericBox Name="C1NumericBox1" FontFamily="Arial" FontSize="10" />
```

コードの場合

たとえば、グリッドのフォントを 10 ポイントの Arial に変更するには、プロジェクトに次のコードを追加します。

Visual Basic

```
C1NumericBox1.FontSize = 10
C1NumericBox1.FontFamily = New System.Windows.Media.FontFamily("Arial")
```

C#

```
c1NumericBox1.FontSize = 10;
c1NumericBox1.FontFamily = new System.Windows.Media.FontFamily("Arial");
```

設計時

設計時に[**プロパティ**]ウィンドウでグリッド内のフォントを Arial の 10 ポイントに変更するには、次の手順に従います。

1. **C1NumericBox** 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、**FontFamily** プロパティを「**Arial**」に設定します。
3. [**プロパティ**]ウィンドウで、**FontSize** プロパティを「**10**」に設定します。

これで、コントロールのフォントサイズとフォントスタイルが設定されます。

プロジェクトの実行と確認

コントロールのコンテンツが Arial の 10 ポイントのフォントで表示されます。

アップ／ダウンボタンの非表示

デフォルトでは、ユーザーがボックス内の値をインクリメントおよびデクリメントできるように、**C1NumericBox** コントロールにはボタンが表示されます。**C1NumericBox** コントロールの**アップ**ボタンと**ダウン**ボタンを実行時に非表示にすることもできます。**アップ**ボタンと**ダウン**ボタンを非表示にするには、**ShowButtons** プロパティを **False** に設定します。

XAML の場合

たとえば、XAML で**アップ**ボタンと**ダウン**ボタンを非表示にするには、**ShowButtons="False"** を **<c1:C1NumericBox>** タグに追加します。次のようになります。

XAML

```
<c1:C1NumericBox Name="C1NumericBox1" ShowButtons="False" />
```

コードの場合

たとえば、**アップ**ボタンと**ダウン**ボタンを非表示にするには、プロジェクトに次のコードを追加します。

Visual Basic

```
C1NumericBox1.ShowButtons = False
```

C#

```
c1NumericBox1.ShowButtons = false;
```

設計時

設計時に**アップ**ボタンと**ダウン**ボタンを非表示に設定するには、次の手順に従います。

1. **C1NumericBox** 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、**ShowButtons** チェックボックスを「オフ」にします。

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

プロジェクトの実行と確認

アップボタンと**ダウン**ボタンは非表示になります。

編集の禁止

デフォルトでは、**C1NumericBox** コントロールの **Value** プロパティは、実行時にユーザーによって編集可能です。コントロールの編集をロックするには、**IsReadOnly** プロパティを True に設定します。

XAML の場合

XAMLで**C1NumericBox**コントロールをロックするに**IsReadOnlyの= "true"**を追加します**<C1:C1NumericBox>**タグは、次のように表示されるように:

XAML

```
<c1:C1NumericBox Name="C1NumericBox1" IsReadOnly="True" />
```

コードの場合

コード内で編集して、実行時から**C1NumericBox**コントロールをロックするには、プロジェクトに次の行を追加します。

Visual Basic

```
C1NumericBox1.IsReadOnly = True
```

C#

```
c1NumericBox1.IsReadOnly = true;
```

設計時

C1NumericBox コントロールの実行時の編集をロックするには、次の手順に従います。

1. **C1NumericBox** 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、**IsReadOnly** チェックボックスを「オン」にします。

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

プロジェクトの実行と確認

コントロールの編集がロックされます。**アップ**ボタンと**ダウン**ボタンは淡色表示され、非アクティブになります。

ProgressBar

ProgressBar for WPF/Silverlight を使用して、時間のかかる操作の進歩状況を視覚的に表示できます。**C1ProgressBar** ドットパターンの繰り返しをアニメーションで表示して、不確定の処理が進行中であることを示します。

主な特長

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

- **高パフォーマンス**

  C1ProgressBar は、UI スレッドの実行をブロックしないため、バックグラウンド UI が処理を行う間、進捗状況インジケータを表示することができます。

- **ユーザーフレンドリなエクスペリエンス**

  C1ProgressBar コントロールは、Windows 8 で使用されているネイティブの進捗状況インジケータに似ており、親しみやすいユーザーエクスペリエンスを提供します。これを使用して、データの読み込みや長い計算の進捗状況を直感的な方法でユーザーに示すことができます。

- **ClearStyleを使用して簡単に色を変更する**

  C1ProgressBar は、テンプレートを上書きしなくてもコントロールのブラシを簡単に変更できる ComponentOne の ClearStyle 技術をサポートします。Visual Studio でブラシのプロパティをいくつか設定するだけで、コントロール全体を簡単に設定できます。例えば、不確定前景・背景ブラッシュで、テンプレートをカスタマイズせずにコントロール全体をスタイルできます。

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

クイックスタート

このクイックスタートは、**ProgressBar 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。このクリックスタートでは、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 **ProgressBar for WPF/Silverlight** コントロールを追加して、実行時機能を確認します。

手順1:アプリケーションの作成

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

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で、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**Width="400" Height="300"**を**Width="Auto" Height="Auto"**に変更して、**UserControl**のサイズを変更します。次のようになります。

   マークアップ

   ```
   <UserControl x:Class="C1RangeSlider.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="Auto"
   Height="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、**<Grid>**タグの直下に次の XAML マークアップを追加します。
6. ツールボックスに移動し、**[C1ProgressBar]**アイコンをダブルクリックして、グリッドにコントロールを追加します。これで、**C1.WPF/C1.Silverlight** 名前空間と**<c1:C1ProgressBar></c1:C1ProgressBar>** タグがプロジェクトに追加されていることを確認してください。

これで、C1ProgressBarコントロールを含む WPF/Silverlight アプリケーションが作成されました。次の手順では、コントロールをカスタマイズします。

手順2:コントロールのカスタマイズ

前の手順では、Visual Studio で、アプリケーションのユーザーインターフェースを作成して **C1ProgressBar** を追加しました。この手順では、CheckBox を追加してコントロールをカスタマイズします。

1. ページ上に **C1ProgressBar** が選択されている状態で、メニューから**[表示]**→**[プロパティ]**を選択します。
2. **[プロパティ]**ウィンドウで、次のプロパティを設定します。
   - **Margin**プロパティを "3" に設定します。
   - **Height**プロパティを "200" に設定します。
   - **Width**プロパティを "450" に設定します。
3. XAML マークアップは次のようになります。

マークアップ

```
<Grid x:Name="LayoutRoot">
        <c1:C1ProgressBar HorizontalAlignment="Left" Margin="3"
Name="C1ProgressBar1" VerticalAlignment="Top" Height="200" Width="450" />
</Grid>
```

4. コードビューで、チェックボックスを追加するマークアップを追加します。

マークアップ

```
<Grid x:Name="LayoutRoot">
  <c1:C1ProgressBar HorizontalAlignment="Left" Margin="3" Name="C1ProgressBar1"
VerticalAlignment="Top" Height="200" Width="450" />
  <CheckBox Content="Show C1ProgressBar1" IsChecked="{Binding IsIndeterminate,
ElementName=C1ProgressBar1, Mode=TwoWay}"/>
</Grid>
```

この手順では、コントロールを正常に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、実行時の操作を確認します。

手順3:アプリケーションの実行

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。
2. **「C1ProgressBar1を表示する」**チェックボックスを選択すると、プログレスバーが表示されます。結果的に、ProgressBar コントロールが赤い一連のドットとして画面上に移動している動作を確認できます。

おめでとうございます。これで、**ProgressBar for WPF/Silverlight** クイックスタートは終了です。

C1ProgressBarについて

C1ProgressBar はドットパターンの繰り返しをアニメーションで表示して、不確定の処理が進行中であることを示します。

C1ProgressBar コントロールの2つの主要なプロパティは、**ActualIsIndeterminate** と **IsIndeterminate** です。

- **ActualIsIndeterminate**

  このプロパティは、実際の不確定プロパティが特定の値を反映しているかどうかを示します。

- **IsIndeterminate**

  このプロパティは、**C1ProgressBar** に継続的に不確定のフィードバックを表示するかどうかを示します。

Progress Indicator

ProgressIndicator for WPF/Silverlight は、時間のかかる操作における進捗を視覚的に示します。**C1ProgressIndicator** コントロールは、不確定な作業が進行中であることを示すため、クラシックな読み込みリングを表示します。

ある操作が数秒以上 UI を中断するようなときに C1ProgressIndicator を使用できます。これは、ユーザーに対して処理が機能していることを示し、それによって、処理が長時間にわたるとしても辛抱強く待ってくれるでしょう。

C1ProgressIndicator コントロールを使用して、以下の内容を述語的に、視覚的に、あるいは両方で示すことができます:

- プロセス内で何が行われているか
- プロセスがどれだけ完了しているか
- 残り時間はどのくらいか
- プロセスをキャンセルする方法

以下に示す主要な機能をうまく活用して、C1ProgressIndicator を最大限に活用してください:

- **高いパフォーマンス**

  C1ProgressIndicator は実行中に UI スレッドをブロックしません。そのためバックグラウンド UI がいくつかの操作を実行している間に進捗インジケータを表示することができます。

- **ユーザーフレンドリ体験**

  C1ProgressIndicator コントロールは身近でモダンな外観を持つ、Windows 8 で使用されているネイティブな進捗インジケータをモデルにして作成されています。それを使用して、データの読み込みやいくつかの長い計算などを直感的な方法で示すことができます。

- **ClearStyle を使用して簡単に色を変更する**

  テンプレートを上書きしなくてもコントロールのブラシを簡単に変更できる **ComponentOne ClearStyle** 技術をサポートします。Visual Studio でブラシのプロパティをいくつか設定するだけで、コントロールの各部のスタイルを簡単に設定できます。

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

RadialMenu

RadialMenu for WPF/Silverlight を使用すると、すばやくアクセスできる状況依存型デスクトップアプリケーションを作成できます。**C1RadialMenu** コントロールは、よく使用されているマイクロソフトアプリケーションをモデルにして、従来のコンテキストメニューに代わるユニークなタッチ操作向きのメニューを提供します。

RadialMenu for WPF/Silverlight の主な特徴は次のようになります。

- **メニューのネスト**

  ラジアルメニューは必要な任意の深さにネストでき、必要な数だけラジアルメニューに項目を追加できます。**C1RadialMenu** コントロールは、含まれている項目の数に基づいて、自動的にセクターを作成します。

- **柔軟な位置決め**

  C1RadialMenu コントロール内の各項目の正確な位置を指定でき、項目の開始角度を指定することもできます。

- **自動選択**

  C1RadialMenu コントロールでは、各メニュー項目に任意の数のサブメニュー項目を入れることができ、各サブメニューには選択された項目が1つ表示されます。選択されたサブメニュー項目を指定することも、それまでのユーザーのアクションに基づいてコントロールに自動的に項目の選択を任せることもできます。このように、デフォルトではないがよく使用されるメニュー項目がメインメニューに表示されるため、これまでよりすばやい選択が可能です。

- **自動折りたたみ**

  デフォルトでは、ユーザーがラジアルメニューの外部をクリックしても、**C1RadialMenu** コントロールはサブメニューと共に開いたままです。ただし、自動折りたたみ機能を有効にして、この動作を変更できます。その場合は、コントロールの境界の外側をクリックして、ラジアルメニューを閉じることができます。

- **チェック可能なメニュー項目**

  IsCheckable プロパティを "**True**" に設定すると、**C1RadialMenuItem** をチェック可能なメニュー項目にすることができます。**C1RadialMenu** のチェックされた項目は、標準のチェックマーク付き項目ではなく、強調表示された項目のように表示されます。

- **ClearStyle を使用して簡単に色を変更する**

  C1RadialMenu コントロールは、テンプレートを上書きしなくてもコントロールのブラシを簡単に変更できる **ComponentOne の ClearStyle** 技術をサポートします。Visual Studio でブラシのプロパティをいくつか設定するだけで、コントロールの各部のスタイルを簡単に設定できます。

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

RadialMenuクイックスタート

手順1:C1RadialMenu アプリケーションの作成

この手順では、最初に Visual Studio で C1RadialMenuコントロール を使用する WPF/Silverlight アプリケーションを作成します。

XAMLの場合

1. Visual Studio で、[**ファイル**]→[**新規作成**]を選択します。
2. ツールボックスに移動し、**C1RadialMenu**コントロールを見つけます。
3. **C1RadialMenu**アイコンをダブルクリックして、MainWindowにコントロールを追加します。
4. MainPage.xaml または MainWindow .xaml をまだ開いていない場合は開き、**<Window>** タグと **</Window>** タグの間に次のマークアップを追加します。このマークアップは、メニュー項目コンテンツのレイアウトを定義するスタイルリソースを追加します。

XAML copyCode

```
<Window.Resources>
    <Style TargetType="TextBlock" x:Key="TextIconStyle">
        <Setter Property="Margin" Value="-10" />
        <Setter Property="FontSize" Value="20" />
        <Setter Property="FontFamily" Value="Segoe UI Symbol" />
        <Setter Property="FontWeight" Value="Normal" />
        <Setter Property="Foreground" Value="#333333" />
```

```
        <Setter Property="HorizontalAlignment" Value="Center" />
        <Setter Property="VerticalAlignment" Value="Center" />
    </Style>
    <Style TargetType="Image" >
        <Setter Property="Width" Value="25"/>
        <Setter Property="Height" Value="25"/>
        <Setter Property="Margin" Value="0"/>
    </Style>
</Window.Resources>
```

5. アプリケーションに境界線の定義を追加する場合、**<Grid>** タグと **</Grid>** タグの間に次のマークアップを追加します。

XAML

```
<Border Background="LemonChiffon" MinHeight="40" BorderBrush="#969696"
BorderThickness="1" Padding="5" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch"></Border>
```

このマークアップは、します。

これで、**RadialMenu for WPF/Silverlight** クイックスタートの最初の手順は終了です。この手順では、WPF/Silverlight プロジェクトを作成しました。次の手順では、**RadialMenu** コントロールとメニュー項目を追加します。

手順2:コントロールへの RadialMenu 項目の追加

前の手順では、WPF/Silverlight プロジェクトを作成して**C1RadialMenu** コントロールを追加しました。この手順では、前の手順で追加した**C1RadialMenu** コントロールのプロパティを設定します。また、メインラジアルメニューにサブメニューとしていくつかの**C1RadialMenuItem** コントロールを追加します。マークアップに追加された各メニュー項目では、画像とTextBlockのラベルが含まれます。

1. MainWindow.xaml で**<Grid></Grid>** と**<Border></Border>**タグの間に**C1ContextMenuService**のマークアップを追加します。

XAML CS copyCode

```
<c1:C1ContextMenuService.ContextMenu>
    <c1:C1RadialMenu x:Name="contextMenu" Offset="-133,0" Opened="contextMenu_Opened"
AccentBrush="ForestGreen"
                     ItemClick="contextMenu_ItemClick" ItemOpened="contextMenu_ItemOpened"
BorderBrush="#FFC6DEC4">
        <c1:C1RadialMenuItem Header="UndoRedo" SelectedIndex="0" ShowSelectedItem="True"
                             Command="{Binding UndoCommand, ElementName=page}">
            <c1:C1RadialMenuItem.Icon>
                <Image Source="Menu/e10e-Undo.48.png" />
            </c1:C1RadialMenuItem.Icon>
        </c1:C1RadialMenuItem>
         <c1:C1RadialMenuItem Header="追加" SectorCount="6">
            <c1:C1RadialMenuItem.Icon>
                <Image Source="Menu/e109-Add.48.png"/>
            </c1:C1RadialMenuItem.Icon>
            <c1:C1RadialMenuItem Header="新しい" ToolTip="新しい項目">
                <c1:C1RadialMenuItem.Icon>
                    <Image Source="Menu/e1da-Folder.48.png"/>
                </c1:C1RadialMenuItem.Icon>
```

```
            </c1:C1RadialMenuItem>
            <c1:C1RadialMenuItem Header="既存" ToolTip="既存項目">
                <c1:C1RadialMenuItem.Icon>
                    <Image Source="Menu/e132-File.48.png"/>
                </c1:C1RadialMenuItem.Icon>
            </c1:C1RadialMenuItem>
            <c1:C1RadialMenuItem Header="フォルダ">
                <c1:C1RadialMenuItem.Icon>
                    <Image Source="Menu/e188-Folder.48.png"/>
                </c1:C1RadialMenuItem.Icon>
            </c1:C1RadialMenuItem>
            <c1:C1RadialMenuItem Header="クラス">
                <c1:C1RadialMenuItem.Icon>
                    <Image Source="Menu/1f4c4-Document.48.png"/>
                </c1:C1RadialMenuItem.Icon>
            </c1:C1RadialMenuItem>
        </c1:C1RadialMenuItem>
        <c1:C1RadialMenuItem Header="ファイル" SectorCount="6">
            <c1:C1RadialMenuItem Header="除外" ToolTip="プロジェクトから除外" DisplayIndex="2">
                <c1:C1RadialMenuItem.Icon>
                    <Image Source="Menu/e10a-Cancel.48.png"/>
                </c1:C1RadialMenuItem.Icon>
            </c1:C1RadialMenuItem>
            <c1:C1RadialMenuItem Header="削除">
                <c1:C1RadialMenuItem.Icon>
                    <Image Source="Menu/e107-Delete.48.png"/>
                </c1:C1RadialMenuItem.Icon>
            </c1:C1RadialMenuItem>
            <c1:C1RadialMenuItem Header="名前を変更します">
                <c1:C1RadialMenuItem.Icon>
                    <Image Source="Menu/e104-Edit.48.png"/>
                </c1:C1RadialMenuItem.Icon>
            </c1:C1RadialMenuItem>
        </c1:C1RadialMenuItem>
        <c1:C1RadialMenuItem Header="プロパティ">
            <c1:C1RadialMenuItem.Icon>
                <Image Source="Menu/e115-Settings.48.png"/>
            </c1:C1RadialMenuItem.Icon>
        </c1:C1RadialMenuItem>
    </c1:C1RadialMenu>
</c1:C1ContextMenuService.ContextMenu>
```

C1ContextMenuService は、**C1RadialMenu** が任意のコントロールのコンテキストメニューとして動作できるようにします。コンテキストメニューは、親コントロールを右クリックまたは右タップすると自動的に表示されます。この場合、親コントロールは Grid です。次に、このコントロール内に **C1RadialMenu** を追加します。上のマークアップは、RadialMenu にメニュー項目を追加します。**C1RadialMenu** コントロールをプログラムによって表示する場合は、**C1ContextMenuService** を省略し、コードで単に **Show** メソッドを呼び出すことができます。

2. **</c1radial:C1ContextMenuService.ContextMenu>** タグのすぐ下に次のマークアップを追加します。

XAML

```
<Grid HorizontalAlignment="Stretch" VerticalAlignment="Top"  Height="75" >
 <TextBlock Foreground="Sienna" TextAlignment="Center" FontSize="14" x:Name="text" Text="ここはコンテクストメニューのボタンを押してください (Windowsの場合右クリックする)。" TextWrapping="Wrap" />
</Grid>
```

これは、**</Border>** タグの前後に1つずつ、合計2つの汎用 **TextBlock** コントロールをアプリケーションに追加します。最初の **TextBlock** には、**C1RadialMenu** の表示方法についての指示が入ります。2番目の **TextBlock** には、ユーザーがタップ、クリック、または開いた **C1RadialMenuItem** が表示されます。

3.

XAML

```
<TextBlock x:Name="txt" Foreground="Red" Text="" FontSize="16" VerticalAlignment="Bottom"
HorizontalAlignment="Center" Margin="10" />
```

4. **MainPage.xaml.cs** ページを開き、プロジェクトに次の Click イベントハンドラを追加します。

   - **Visual Basic**

```
Private Sub contextMenu_ItemClick(sender As Object, e As SourcedEventArgs)
    txt.Text = "Item Clicked: " + DirectCast(e.Source, C1RadialMenuItem).Header.ToString()
    Dim str As String = "Item Clicked: " + DirectCast(e.Source, C1RadialMenuItem).Header.ToString()
End Sub
Private Sub contextMenu_ItemOpened(sender As Object, e As SourcedEventArgs)
    Dim item As C1RadialMenuItem = TryCast(e.Source, C1RadialMenuItem)
    txt.Text = "Item Opened: " + (If(item.Header, item.Name)).ToString()
    Dim str As String = "Item Opened: " + (If(item.Header, item.Name)).ToString()
End Sub
Private Sub contextMenu_Opened(sender As Object, e As EventArgs)
    contextMenu.Expand()
End Sub
```

   - **C#**

```
private void contextMenu_ItemClick(object sender, SourcedEventArgs e)
{
    txt.Text = "Item Clicked: " + ((C1RadialMenuItem)e.Source).Header.ToString();
    string str = "Item Clicked: " + ((C1RadialMenuItem)e.Source).Header.ToString();
}
private void contextMenu_ItemOpened(object sender, SourcedEventArgs e)
{
    C1RadialMenuItem item = e.Source as C1RadialMenuItem;
    txt.Text = "Item Opened: " + (item.Header ?? item.Name).ToString();
    string str = "Item Opened: " + (item.Header ?? item.Name).ToString();
}
private void contextMenu_Opened(object sender, EventArgs e)
{
    contextMenu.Expand();
}
```

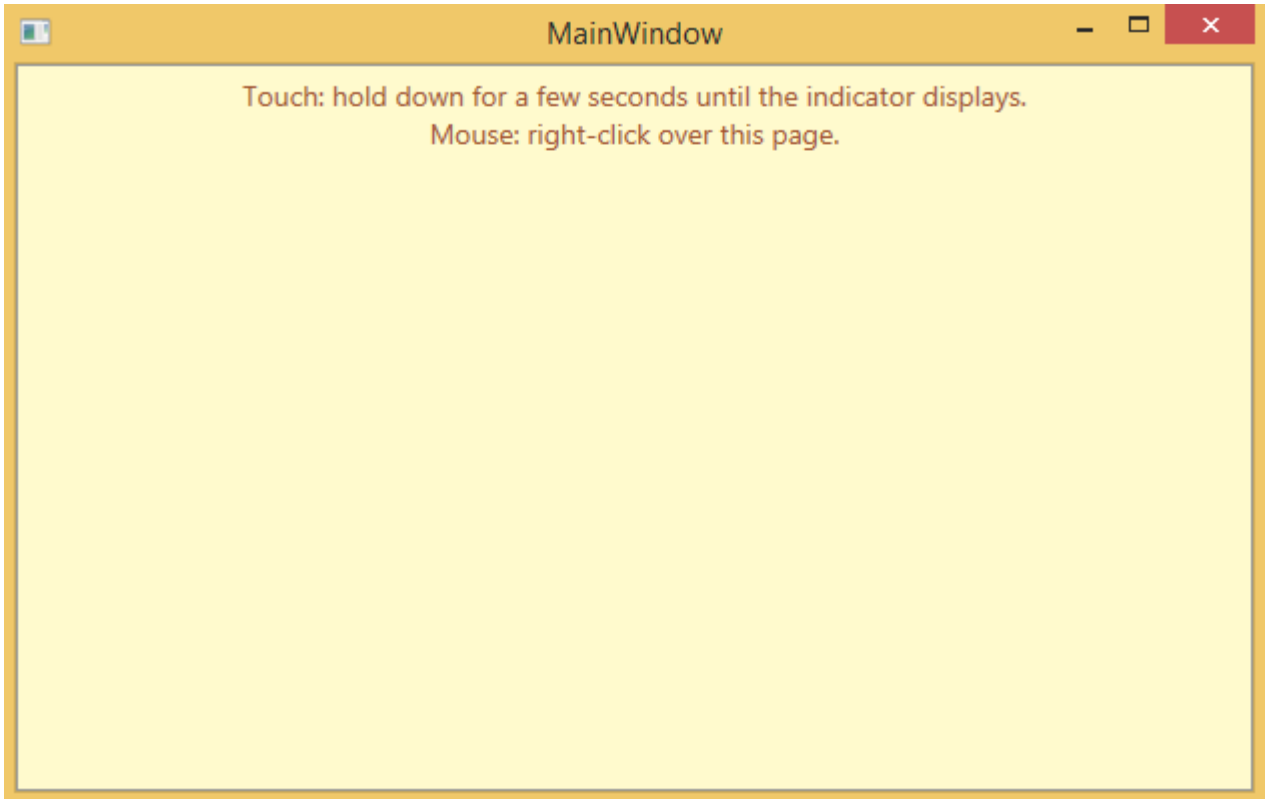
この手順では、**C1RadialMenu** コントロールを追加しました。次の手順では、プロジェクトを実行して **RadialMenu for WPF/Silverlight** クイックスタートの結果を確認します。

手順3：プロジェクトの実行

C1RadialMenu コントロールを持つ WPF/Silverlight ストアプロジェクトを作成したので、この後はプロジェクトを実行して作業の結果を確認するだけです。

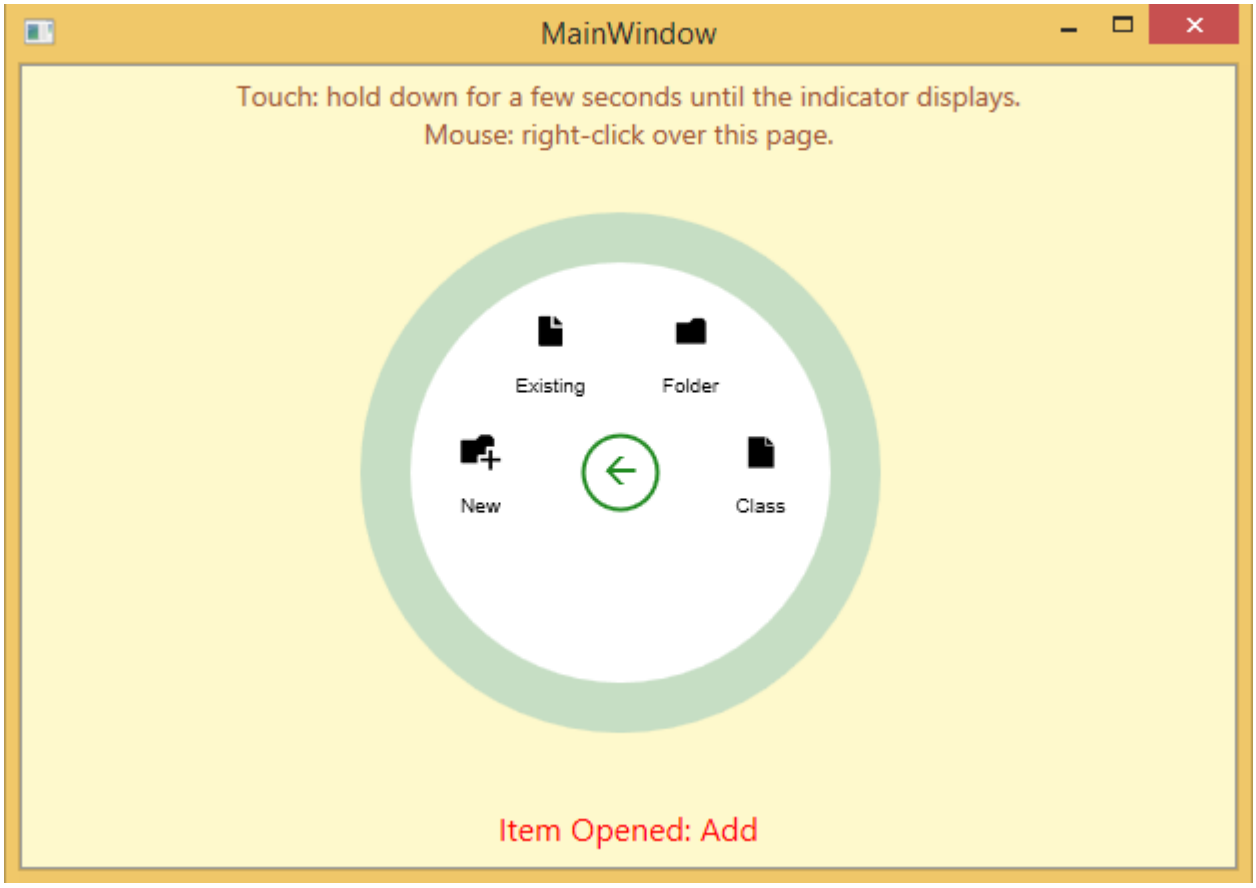
次の手順を実行します。

1. [**デバッグ**]→[**デバッグ開始**]を選択して、プロジェクトを実行します。ページを右タップまたは右クリックすると、次の図のように**ナビゲーションボタン**が表示されます。

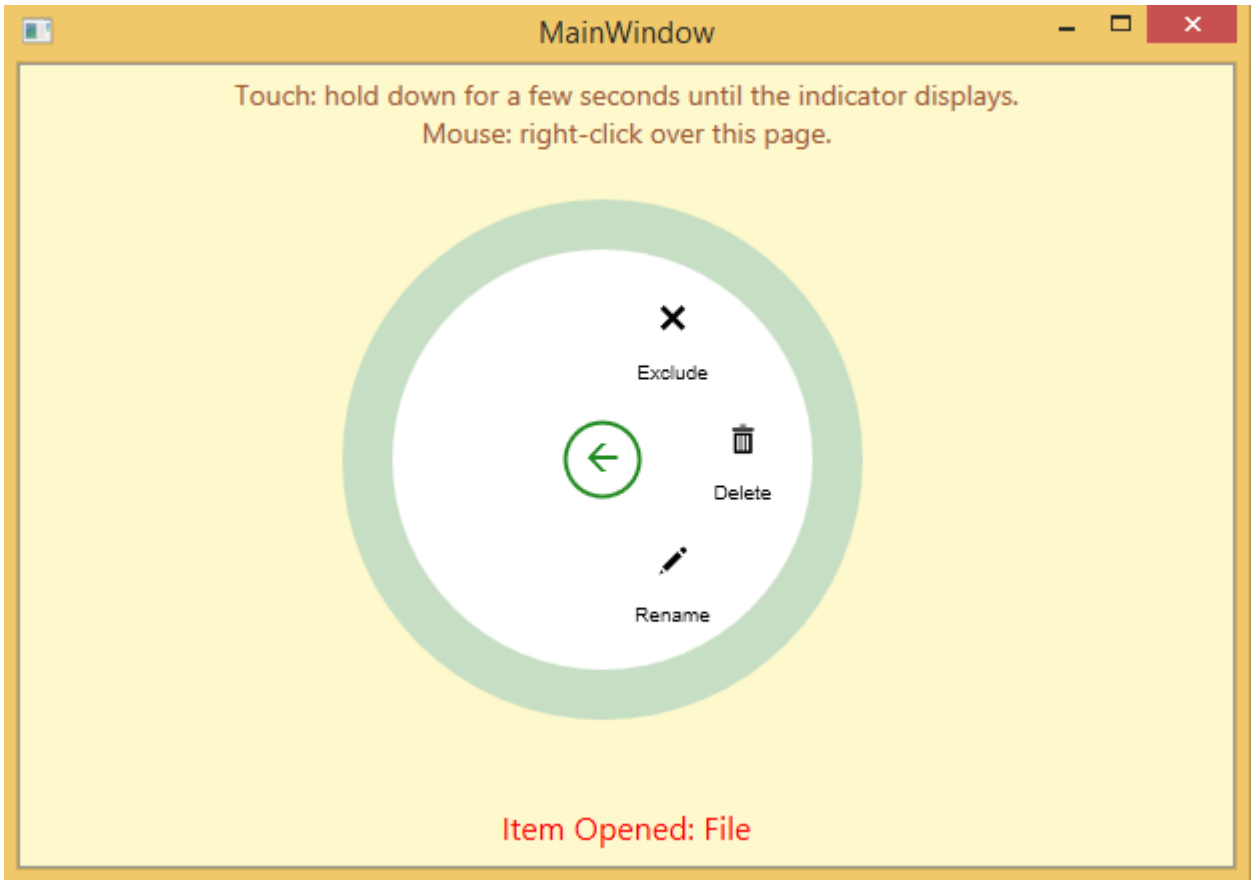


2. ラジアルメニューを表示するには、**ナビゲーションボタン**をタップまたはクリックします。

3. [**Cut**]**C1RadialMenuItem** の上の[**ExpandArea**]をタップし、[**Clipboard**]メニューが表示されることを確認します。



4. メインラジアルメニューに戻るには、**C1RadialMenu** コントロールの中心にある紫色の矢印をタップします。



おめでとうございます!

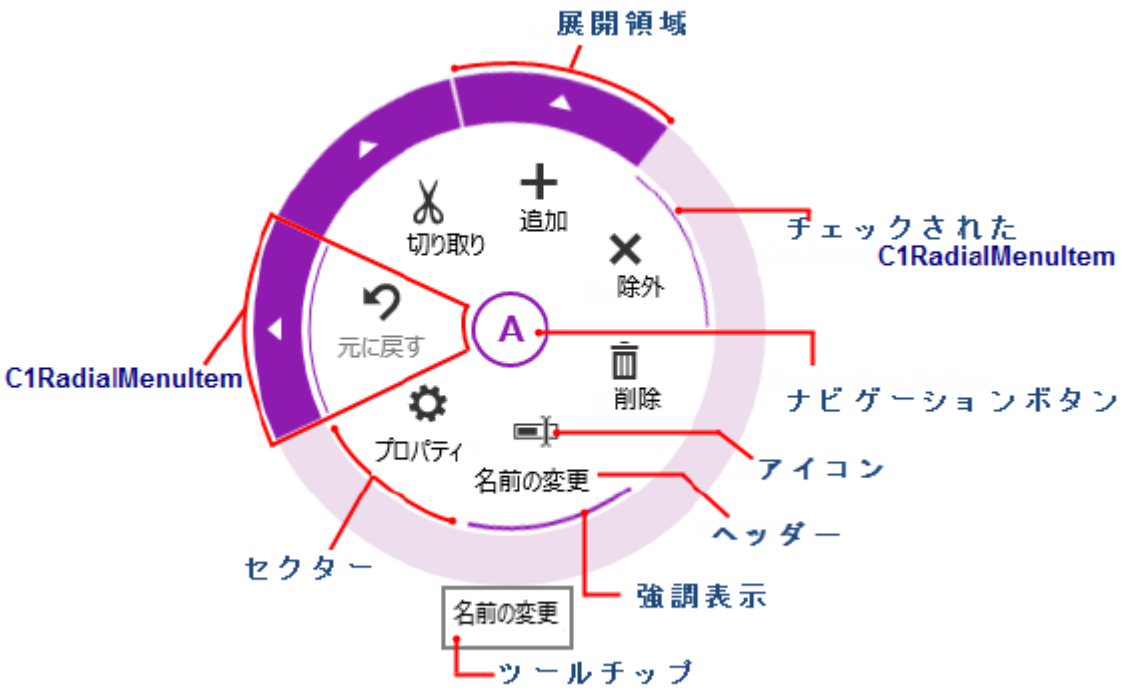
これで、**RadialMenu for WPF/Silverlight** クイックスタートは終了です。

C1RadialMenu の要素

C1RadialMenu の要素

C1RadialMenu は、要素を放射状に並べることができるコントロールです。ラジアルメニューは必要な任意の深さにネストでき、必要な数だけラジアルメニューに項目を追加できます。

次の図に、**C1RadialMenu** コントロールの要素を示します。



C1RadialMenu コントロールの要素には、次のものがあります。

- **C1RadialMenu**

  メインラジアルメニューは、円形の最上位のコンテキストメニューコントロールです。これは、最上位にあるいくつかの **C1RadialMenuItem** で構成され、**C1RadialMenuItem** のすべての機能を利用できます。

- **C1RadialMenuItem**

  ラジアルメニュー項目は、**C1RadialMenuItem** クラスによって表されます。**C1RadialMenuItem** は、**ヘッダー**と**アイコン**を持つことができ、また**チェック**可能に設定することもできます。各ラジアルメニュー項目は**コマンド**に関連付けることができ、関連付けられたコマンドは、ボタンが押されると呼び出されます。子項目を含む **C1RadialMenuItem** には、**展開領域**もあります。

- **セクター**

  C1RadialMenu は複数のセクターに分割されます。**C1RadialMenu** コントロールは、コントロールに含まれる **C1RadialMenuItem** の数に基づいて自動的にセクターを作成しますが、**SectorCount** プロパティを使用して、セク

ター数をカスタマイズすることもできます。

- **ナビゲーションボタン**

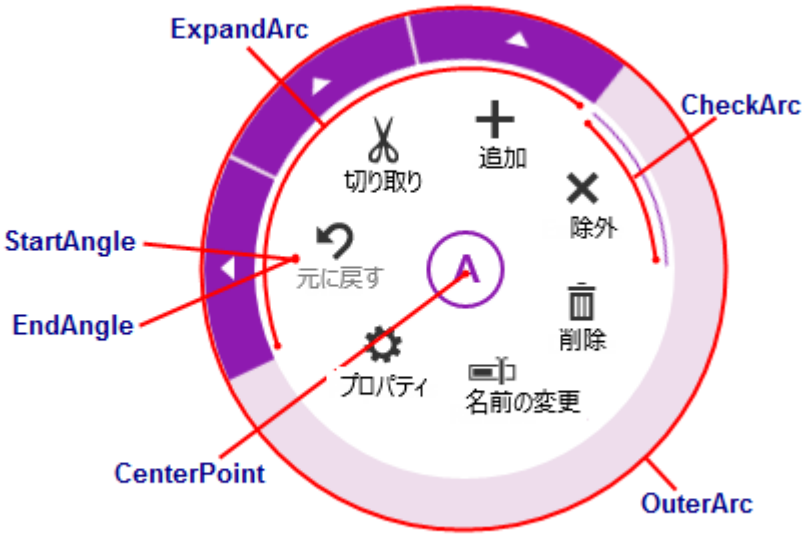
  ユーザーがアプリケーションを右タップすると、**ナビゲーションボタン**が表示されます。**ナビゲーションボタン**をタップして、**C1RadialMenu** を表示または非表示にします。ユーザーがサブメニューに移動すると、このボタンは戻る矢印に変わります。

- **アイコン**

  Icon プロパティを使用して、**C1RadialMenu** に表示されるアイコンをカスタマイズできます。

C1RadialMenuItem 要素

C1RadialMenuItem 要素は、デフォルトでは **C1RadialPanel** によって設定されます。これらの要素は、その C1RadialMenuItem コントロール内に形状パスを描画します。デフォルトコントロールテンプレートはこれらのパスを使用し、ユーザーもカスタムコントロールテンプレートでこれらのパスを使用できます。



- **CenterPoint**

  CenterPoint プロパティは円の中心の座標を取得します。これを使用して、XAML で、現在の **C1RadialMenuItem** を表す扇形を描画できます。

- **CheckArc**

  CheckArc プロパティはチェック円弧の定義を取得します。

- **ExpandArc**

  ExpandArc プロパティは、展開領域円弧の定義を取得します。

- **OuterArc**

  OuterArc プロパティは外側円弧の定義を取得します。これを使用して、XAML で、現在の **C1RadialMenuItem** を表す扇形を描画できます。

- **StartAngle**

StartAngle は、最初の **C1RadialMenuItem** を配置する場所を定義します。上のサンプルの -180 は時計の文字盤の9時に対応します。これがデフォルトの**StartAngle** です。

- **EndAngle**

  EndAngle は、最後の **C1RadialMenuItem** を配置する場所を定義します。上のサンプルの 180 は時計の文字盤の9時に対応します。これは、デフォルトの **EndAngle** です。

  角度の開始点と終了点を同じ位置にすると、**C1RadialMenu** の周囲に適切にメニュー項目が配置されます。

ラジアルメニューの作成

以下のように、**C1RadialMenu** を使用して最上位のメニューもサブメニューも作成できます。

最上位のメニューの作成

次の手順を実行します。

1. **<Grid>** タグと **</Grid>** タグの間に次の XAML を配置します。

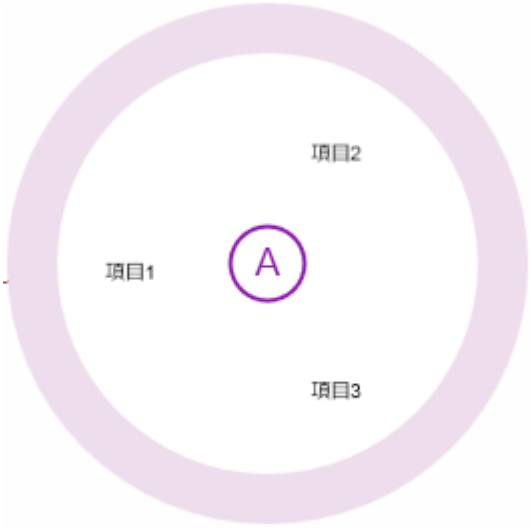
```
XAML
<Xaml:C1ContextMenuService.ContextMenu>
     <Xaml:C1RadialMenu >
    </Xaml:C1RadialMenu>
</Xaml:C1ContextMenuService.ContextMenu>
```

2. 次の XAML を **<Xaml:C1RadialMenu>** タグと **</Xaml:C1RadialMenu>** タグの間に配置します。

```
XAML
<Xaml:C1RadialMenuItem Header="RadialMenuItem1" />
<Xaml:C1RadialMenuItem Header="RadialMenuItem2" />
<Xaml:C1RadialMenuItem Header="RadialMenuItem3" />
```

3. プログラムを実行し、次の点を確認します。

- ページを右タップまたは右クリックします。この場合は、これが C1RadialMenu がアタッチされている要素です。ナビゲーションボタンが表示されることを確認します。
- ラジアルメニューを表示するには、ナビゲーションボタンをタップまたはクリックします。C1RadialMenu に3つの C1RadialMenuItem が含まれていることを確認します。



サブメニュー

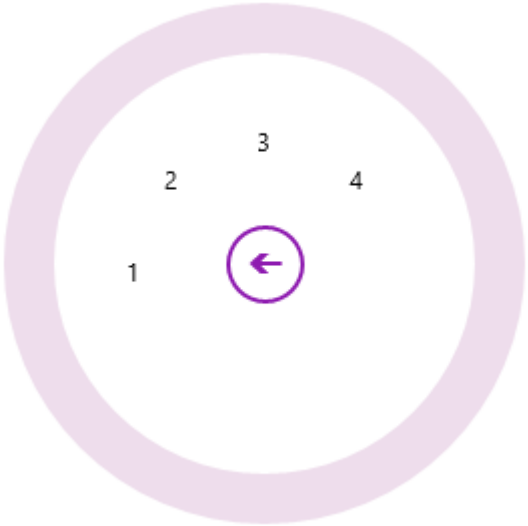
このトピックでは、C1RadialMenu の項目の1つにアタッチされるサブメニューを作成します。このトピックでは、少なくとも1つの C1RadialMenuItem を持つ最上位のラジアルメニューが既に作成されていることを前提としています。

1. 次の XAML を **<c1radial:C1RadialMenu>** タグと **</c1radial:C1RadialMenu>** タグの間に配置します。

```
XAML
<c1radial:C1RadialMenuItem Header="ここでタップ" >
  <c1radial:C1RadialMenuItem Header="項目1" />
  <c1radial:C1RadialMenuItem Header="項目2" />
  <c1radial:C1RadialMenuItem Header="項目3" />
</c1radial:C1RadialMenuItem>
```

2. プログラムを実行します。
   - ページを右タップまたは右クリックして、ナビゲーションボタンを表示します。ナビゲーションボタンをタップまたはクリックして、メイン **C1RadialMenu** を表示します。
   - [Tap Here]項目をタップし、作成したサブメニューが表示されることを確認します。

次は、[Tap Here]項目をタップした後の **C1RadialMenu** です。



カラーピッカーメニューの作成

C1RadialColorItem を使用して、**C1RadialMenu** コントロールを使用するカラーピッカーを作成できます。このトピックでは、C1RadialMenu アプリケーションを作成し、C1RadialMenu コントロールにいくつかの C1RadialColorItem を追加します。アプリケーションの作成には、XAML マークアップとコードの両方を使用します。

次の手順を実行します。

1. ページ内の **<Grid> </Grid>** タグを見つけ、このタグを次のマークアップに置き換えて、アプリケーションのフレームワークを作成します。

```
XAML
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Border Background="LemonChiffon" MinHeight="40" BorderBrush="#969696"
```

```
                BorderThickness="1" Padding="5" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch">
            <c1radial:C1ContextMenuService.ContextMenu>

            </c1radial:C1ContextMenuService.ContextMenu>

           <c1radial:C1ListViewer x:Name="text" Foreground="Sienna"
HorizontalAlignment="Stretch" VerticalAlignment="Center" Height="75"
ZoomMode="Disabled" >
                <TextBlock Text=""タッチ： インジケータが表示されるまでに数秒間押したままにします。
&#10;キーボード： このテキスト上にあるコンテキストメニューボタンを押してください。&#10;マウス： このテキスト上で
右クリックします。"
                           FontSize="16" TextWrapping="Wrap" />
            </c1radial:C1ListViewer>
        </Border>
        <TextBlock x:Name="txt" Foreground="Red" Text="" FontSize="16"
VerticalAlignment="Bottom" HorizontalAlignment="Center" Margin="10" />
    </Grid>
```

2. 追加したマークアップにある **<Xaml:C1ContextMenuService.ContextMenu> </Xaml:C1ContextMenuService.ContextMenu>** タグを探します。このタグの間にカーソルを置きます。

3. Visual Studio ツールボックスで **C1RadialMenu** コントロールを見つけてダブルクリックし、このコントロールをアプリケーションに追加します。

4. 次のように、開始タグ **<C1RadialMenu>** を編集します。

XAML

```
<Xaml:C1RadialMenu x:Name="contextMenu" Offset="-130,0"
ItemClick="contextMenu_ItemClick" >
```

5. 次のマークアップを **<C1RadialMenu> </C1RadialMenu>** タグの間に追加して、3つの **C1RadialColorItem** をアプリケーションに追加します。

XAML

```
<Xaml:C1RadialColorItem x:Name="rainbowItem" ShowSelectedItem="True"
AutoSelect="True" IsSelectable="False" >
</Xaml:C1RadialColorItem>
<Xaml:C1RadialColorItem x:Name="greenItem" SelectedIndex="5" ShowSelectedItem="True"
AutoSelect="True" IsSelectable="False" >
</Xaml:C1RadialColorItem>
<Xaml:C1RadialColorItem x:Name="blueItem" SelectedIndex="0" ShowSelectedItem="True"
AutoSelect="True" IsSelectable="False" >
</Xaml:C1RadialColorItem>
```

6. 「**rainbowItem**」という名前の **C1RadialColorItem** を選択し、次のマークアップを開始タグと終了タグの間に挿入します。これで、**SolidColorBrush** のサブメニュー項目が追加されます。

XAML

```
<SolidColorBrush Color="Red"/>
<SolidColorBrush Color="Orange"/>
<SolidColorBrush Color="Yellow"/>
<SolidColorBrush Color="Green"/>
<SolidColorBrush Color="Blue"/>
<SolidColorBrush Color="Indigo"/>
```

```
<SolidColorBrush Color="Violet"/>
```

7. 「**greenItem**」という名前の **C1RadialColorItem** を選択し、次のマークアップを開始タグと終了タグの間に挿入します。これで、いくつかの階調の緑色に対応する **C1RadialColorItem** サブメニュー項目が追加されます。

XAML

```
<Xaml:C1RadialColorItem x:Name="greenItem" SelectedIndex="5" ShowSelectedItem="True"
AutoSelect="True" IsSelectable="False">
    <Xaml:C1RadialColorItem ToolTip="Lime" Brush="#FF92D050" GroupName="Green"/>
    <Xaml:C1RadialColorItem ToolTip="Light Green" Brush="#FFC6EFCE"
GroupName="Green"/>
    <Xaml:C1RadialColorItem ToolTip="Green" Brush="#FF00FF00" GroupName="Green"/>
    <Xaml:C1RadialColorItem ToolTip="Dark Green" Brush="#FF1D421E"
GroupName="Green"/>
    <Xaml:C1RadialColorItem ToolTip="Dark Green" Brush="#FF1D5A2D"
GroupName="Green"/>
    <Xaml:C1RadialColorItem ToolTip="Dark Green" Brush="Green" GroupName="Green"/>
    <Xaml:C1RadialColorItem ToolTip="Dark Green" Brush="#FF008000"
GroupName="Green"/>
    <Xaml:C1RadialColorItem ToolTip="Dark Green" Brush="#FF00B050"
GroupName="Green"/>
</Xaml:C1RadialColorItem>
```

8. 「**blueItem**」という名前の **C1RadialColorItem** を選択し、次のマークアップを開始タグと終了タグの間に挿入します。これで、いくつかの階調の青色に対応する **C1RadialColorItem** サブメニュー項目が追加されます。

XAML

```
<Xaml:C1RadialColorItem x:Name="blueItem" SelectedIndex="0" ShowSelectedItem="True"
AutoSelect="True" IsSelectable="False" >
    <Xaml:C1RadialColorItem ToolTip="Blue" Brush="Blue" GroupName="Blue"/>
    <Xaml:C1RadialColorItem ToolTip="Slate Blue" Brush="MediumSlateBlue"
GroupName="Blue"/>
    <Xaml:C1RadialColorItem ToolTip="Turquoise" Brush="Turquoise" GroupName="Blue"/>
    <Xaml:C1RadialColorItem ToolTip="Aqua" Brush="Aqua" GroupName="Blue"/>
    <Xaml:C1RadialColorItem ToolTip="Sky Blue" Brush="SkyBlue" GroupName="Blue"/>
    <Xaml:C1RadialColorItem ToolTip="Purple" Brush="#FFAC38AC"
AccentBrush="#FF801C80" GroupName="Blue"/>
    <Xaml:C1RadialColorItem ToolTip="Dark Purple" Brush="Purple" GroupName="Blue"/>
    <Xaml:C1RadialColorItem ToolTip="Dark Blue" Brush="DarkBlue" GroupName="Blue"/>
</Xaml:C1RadialColorItem>
```

9. ページを右クリックし、リストから[コードの表示]を選択します。

10. 次の using 文をページの先頭に追加します

C#

```
using C1.Xaml;
```

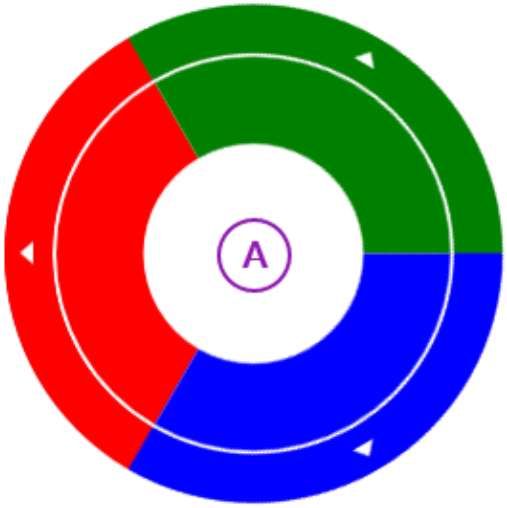
11. 次の **ItemClick** イベントをページに追加します。これで、**C1RadialMenu** から色を選択した際に、メインページに追加したテキストの色を変化させることができます。

C#

```
private void contextMenu_ItemClick(object sender, SourcedEventArgs e)
        {
```

```
            C1RadialMenuItem item = e.Source as C1RadialMenuItem;
            if (item is C1RadialColorItem)
            {
                this.text.Foreground = ((C1RadialColorItem)item).Brush;
                txt.Text = "選択されました: " +
((C1RadialColorItem)item).Color.ToString();
            }
            else
            {
                txt.Text = "選択されました: " + (item.Header ?? item.Name).ToString();
            }
        }
```

12. [**F5**]キーを押すか、デバッグを開始して、アプリケーションを実行します。アプリケーションを右クリックまたは右タップして **C1RadialMenu** を開くと、次の図のように表示されます。



数値ラジアルメニューの作成

C1RadialMenu コントロールを使用して、円形の数値スライダメニューを作成できます。このメニューは、特に、ユーザーにフォントサイズを選択してもらうアプリケーションで役立ちます。アプリケーションの作成には、XAML マークアップとコードの両方を使用します。

次の手順を実行します。

1. ページ内の **<Grid> </Grid>** タグを見つけ、このタグを次のマークアップに置き換えて、アプリケーションのフレームワークを作成します。

XAML

```
Page.Resources>
        <Style TargetType="TextBlock" x:Key="TextIconStyle">
            <Setter Property="Margin" Value="-10" />
            <Setter Property="FontSize" Value="20" />
            <Setter Property="FontFamily" Value="Segoe UI Symbol" />
            <Setter Property="FontWeight" Value="Normal" />
            <Setter Property="Foreground" Value="#333333" />
            <Setter Property="HorizontalAlignment" Value="Center" />
            <Setter Property="VerticalAlignment" Value="Center" />
```

```
        </Style>
        <Style TargetType="Image" x:Key="MenuIcon">
            <Setter Property="Width" Value="16"/>
            <Setter Property="Height" Value="16"/>
            <Setter Property="Margin" Value="0"/>
        </Style>
    </Page.Resources>
    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <c1radial:C1ContextMenuService.ContextMenu>

        </c1radial:C1ContextMenuService.ContextMenu>

        <c1radial:C1ListViewer x:Name="text" Foreground="Sienna"
HorizontalAlignment="Stretch" VerticalAlignment="Center" Height="75"
ZoomMode="Disabled" FontSize="16" >
            <TextBlock Text="タッチ： インジケータが表示されるまでに数秒間押したままにします。
&#10;キーボード： このテキスト上にあるコンテキストメニューボタンを押してください。&#10;マウス： このテキスト
上で右クリックします。"
                       TextWrapping="Wrap" />
        </c1radial:C1ListViewer>

        <TextBlock x:Name="txt" Foreground="Red" Text="" FontSize="16"
VerticalAlignment="Bottom" HorizontalAlignment="Center" Margin="10" />

    </Grid>
```

2. 追加したマークアップにある **<Xaml:C1ContextMenuService.ContextMenu> </Xaml:C1ContextMenuService.ContextMenu>** タグを探します。このタグの間にカーソルを置きます。
3. Visual Studio ツールボックスで **C1RadialMenu** コントロールを見つけてダブルクリックし、このコントロールをアプリケーションに追加します。
4. 次のように、開始タグ **<C1RadialMenu>** を編集します。

XAML

```
<Xaml:C1RadialMenu x:Name="contextMenu" Offset="-130,0"
Opened="contextMenu_Opened" AccentBrush="ForestGreen"
ItemClick="contextMenu_ItemClick" ItemOpened="contextMenu_ItemOpened"
BorderBrush="#FFC6DEC4">
```

5. 次のマークアップを **<C1RadialMenu> </C1RadialMenu>** タグの間に追加して、1つの **C1RadialNumericItem** をアプリケーションに追加します。同時に、いくつかのサブ項目と1つのカスタム項目アイコンも追加します。

XAML

```
<Xaml:C1RadialNumericItem Header="Font Size" Minimum="9" Maximum="72"
MarkStartAngle="-128" MarkEndAngle="231" x:Name="fontSize" Value="11">
    <Xaml:C1RadialNumericItem.Icon>
      <TextBlock Style="{StaticResource TextIconStyle}" FontFamily="Segoe UI"
FontSize="18">
         <Run Text="A"/>
         <Run Text="{Binding Value, ElementName=fontSize}"
Typography.Variants="Superscript"/>
      </TextBlock>
```

```
    </Xaml:C1RadialNumericItem.Icon>
    <x:Double>9</x:Double>
    <x:Double>11</x:Double>
    <x:Double>13</x:Double>
    <x:Double>16</x:Double>
    <x:Double>20</x:Double>
    <x:Double>36</x:Double>
    <x:Double>72</x:Double>
</Xaml:C1RadialNumericItem>
```

6. ページを右クリックし、リストから[コードの表示]を選択します。
7. 次の using 文をページの先頭に追加します。

C#

```
using C1.WPF;
OR
using C1.Silverlight
```

8. 次の **ItemClick** イベントをページに追加します。これで、選択した項目に合わせて、**TextBox** コントロール内のテキストのサイズを変更できるようになります。

XAML

```
private void contextMenu_ItemClick(object sender, SourcedEventArgs e)
        {
            C1RadialMenuItem item = e.Source as C1RadialMenuItem;

if (item is C1RadialNumericItem)
            {
                txt.FontSize = ((C1RadialNumericItem)item).Value;
                txt.Text = "選択されました: " +
((C1RadialNumericItem)item).Value.ToString();
            }
            else
            {
                txt.Text = "選択されました: " + (item.Header ??
item.Name).ToString();
            }
        }
```

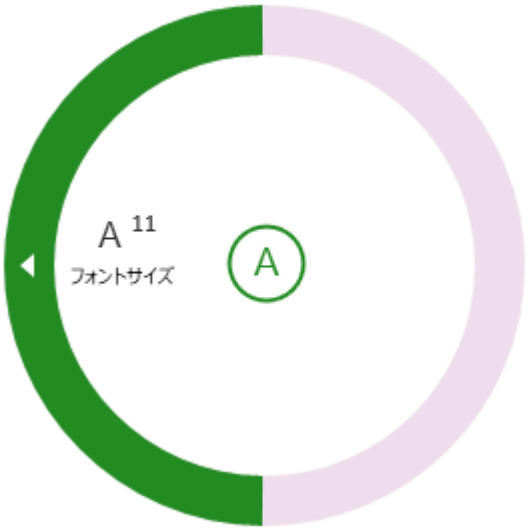
9. さらに、2つのイベントを追加します。このコードは、**ItemOpened** イベントと **Opened** イベントを制御します。

XAML

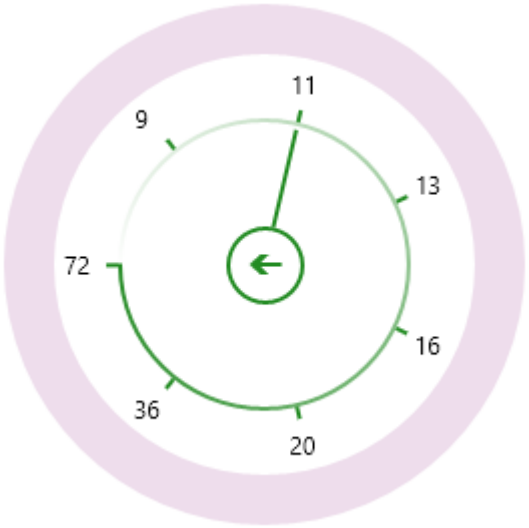
```
private void contextMenu_ItemOpened(object sender, SourcedEventArgs e)
        {
            C1RadialMenuItem item = e.Source as C1RadialMenuItem;
            txt.Text = "開かれました: " + (item.Header ?? item.Name).ToString();
        }
        private void contextMenu_Opened(object sender, EventArgs e)
        {
            // このサンプルでは、編集可能な基底のコントロールがないため、即座にメニューを展開します。
            contextMenu.Expand();
```

```
        }
```

10. [**F5**]キーを押すか、デバッグを開始して、アプリケーションを実行します。**C1RadialMenu** コントロールは次の図のようになります。



11. [**Font Size**]オプションをクリックすると、**C1NumericRadialItem** が表示されます。



RadialMenu の機能

自動メニュー折りたたみを有効にする

自動メニュー折りたたみ機能は、開いた状態の**C1RadialMenu** コントロールを自動的に折りたたむオプションを提供します。ラジアルメニューを閉じるための使用可能な唯一の方法は、ナビゲーションボタンをクリックすることです。しかし、メニュー領域以外の場所をクリックするだけでもRadialMenuが閉じるようになります。

C1RadialMenu クラスの**AutoCollapse**プロパティを使用すると、RadialMenuコントロールの自動折りたたみが有効になります。これを無効にするため、**AutoCollapse**プロパティを**False** に設定します。デフォルトでは、**AutoCollapse**プロパティが**True**に設定されます。

XAMLの場合

1. <Xaml:C1RadialMenu> タグに AutoCollapse="True" を追加します。

2. プログラムを実行します。

コードの場合

1. ソースビューで、チェック可能にする項目の <c1radial:C1RadialMenuItem> タグを探し、それに Name="C1RadialMenu1" を追加します。これにより、コードから使用できる一意の識別子を項目に渡すことができます。
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。

Visual Basic
```
C1RadialMenu1.AutoCollapse = True
```

C#
```
C1RadialMenu1.AutoCollapse = true;
```

3. プログラムを実行します。

プロジェクトを実行したら、ページを右クリックして**C1RadialMenu**コントロールを開きます。開いた状態でメニュー外の場所にクリックすると、C1RadialMenuコントロールが折りたたみます。

子RadialMenuItemsのクリックで折りたたむ

C1RadialMenuでは、空の子ラジアルメニュー項目のクリックでメニューが折りたたむことが可能です。本機能を有効にするには、**CollapseOnClick** プロパティを**True**に設定します。

XAMLの場合

1. <C1RadialMenuItem> タグに ClickOnCollapse="True"を追加します。
2. プログラムを実行します。

コードの場合

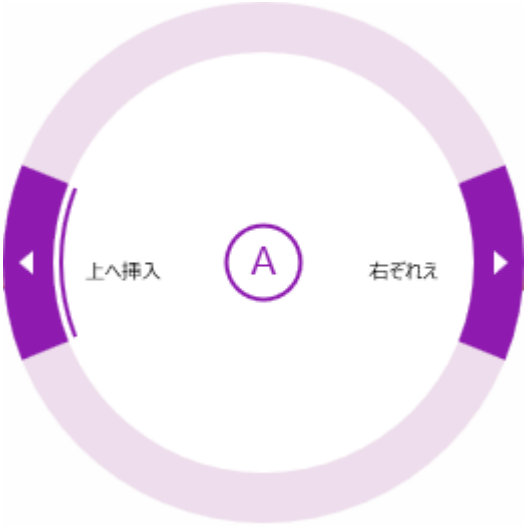
1. コードビューに切り替えて、InitializeComponent()にて次のコードを追加します:

Visual Basic
```
C1RadialMenuItem1.ClickOnCollapse = True
```

C#
```
C1RadialMenuItem1.ClickOnCollapse = true;
```

チェック可能な C1RadialMenuItem の作成

このトピックでは、ユーザーがオンまたはオフにすることができるチェック可能な C1RadialMenuItem を作成します。このトピックを完了するには、1つ以上の項目を持つ C1RadialMenu コントロールか、1つ以上のサブメニューを持つ C1RadialMenu コントロールが必要です。



XAML の場合

1. チェック可能にする **C1RadialMenuItem** の **<Xaml:C1RadialMenuItem>** タグを探し、タグに **IsCheckable="True"** を追加します。XAML は次のようになります。

```
XAML
<Xaml:C1RadialMenuItem Header="C1RadialMenuItem" IsCheckable="True"/>
```

2. プロジェクトを実行します。

コードの場合

1. ソースビューで、チェック可能にする項目の **<Xaml:C1RadialMenuItem>** タグを探し、それに **Name="CheckableRadialMenuItem"** を追加します。これにより、コードから使用できる一意の識別子を項目に渡すことができます。
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。

```
Visual Basic
CheckableRadialMenuItem.IsCheckable = True
C#
CheckableRadialMenuItem.IsCheckable = true;
```

3. プログラムを実行します。

プログラムを実行したら、**C1RadialMenu** を開きます。チェックマークを付けるには、**C1RadialMenuItem** をクリックします。

相互に排他的なチェック可能なアイテム

グループに追加する各項目の **GroupName** プロパティを設定すると、相互に排他的なチェック可能な項目のグループを作成できます。たとえば、下の XAML は、相互に排他的な4つのチェック可能な項目のグループを作成します。

```
XAML
<Xaml:C1RadialMenu SectorCount="8" >
```

```
  <Xaml:C1RadialMenuItem Header="挿入" SectorCount="8" AutoSelect="True"
ShowSelectedItem="True" IsCheckable="True" >
    <Xaml:C1RadialMenuItem Header="左へ挿入" IsCheckable="True"
GroupName="MutuallyExclusiveGroup"/>
    <Xaml:C1RadialMenuItem Header="上へ挿入" DisplayIndex="2" IsCheckable="True"
GroupName="MutuallyExclusiveGroup"/>
    <Xaml:C1RadialMenuItem Header="右へ挿入" DisplayIndex="4" IsCheckable="True"
GroupName="MutuallyExclusiveGroup"/>
    <Xaml:C1RadialMenuItem Header="下へ挿入" DisplayIndex="6" IsCheckable="True"
GroupName="MutuallyExclusiveGroup" />
  <Xaml:C1RadialMenuItem>
</Xaml1:C1RadialMenu>
```

相互に排他的なグループ内では、一度に1つの **C1RadialMenuItem** しかチェックできません.次の手順を実行します。

XAML の場合

1. 相互に排他的なチェック可能な項目のグループに追加する **C1RadialMenuItem** ごとに、**<Xaml:C1RadialMenuItem>** タグに **IsCheckable="True"** と **GroupName="CheckableGroup"** を追加します。
2. プログラムを実行してグループの1番目の項目をクリックします。**C1RadialMenuItem** が強調表示されていることを確認します。
3. ここで、グループの2番目の項目をクリックします。1番目の項目から強調表示が削除され、2番目の項目に追加されることを確認します。

コードの場合

1. 相互に排他的なチェック可能な項目のグループに追加する各 **C1RadialMenuItem** 項目の Name プロパティを設定します。
2. MainPage.xaml.cs ページを開きます。
3. 各 **C1RadialMenuItem** の **IsCheckable** プロパティと **GroupName** プロパティを設定し、**"ItemName"** を **C1RadialMenuItem** の Name プロパティの値に置き換えます。

Visual Basic
`ItemName.IsCheckable = True`
C#
`ItemName.IsCheckable = true;`

4. プログラムを実行してグループの1番目の項目をタップし、次の点を確認します。

- 強調表示の線を細くしたようなチェックが **C1RadialMenuItem** に追加されることを確認します。
- ここで、グループの2番目の項目をタップします。1番目の項目からチェックが削除され、2番目の項目に追加されることを確認します。

ナビゲーションボタン

C1RadialMenu の**ナビゲーションボタン**は、ユーザーがアプリケーションを右タップしたときに最初に表示されるオブジェクトです。ナビゲーションボタンは、**C1RadialMenu** の **Icon** プロパティと **NavigationButtonRelativeSize** プロパティを使用してカ

スタマイズできます。**NavigationButtonRelativeSize** プロパティを使用すると、**C1RadialMenu** のサイズを基準にしてナビゲーションボタンのサイズを設定することができます。デフォルトでは、このプロパティは 0.15 に設定されています。

たとえば、次は **NavigationButtonRelativeSize** を 0.15 に設定したデフォルトのナビゲーションボタンです。

NavigationButtonRelativeSize プロパティを 0.45 に設定すると、ナビゲーションボタンは次のようになります。

Icon プロパティを変更することもできます。デフォルトでは、**Icon** プロパティは "A" に設定されています。

デフォルトナビゲーションボタンの **Icon** プロパティを変更し、**NavigationButtonRelativeSize** プロパティを 0.25 に設定すると、次のようになります。

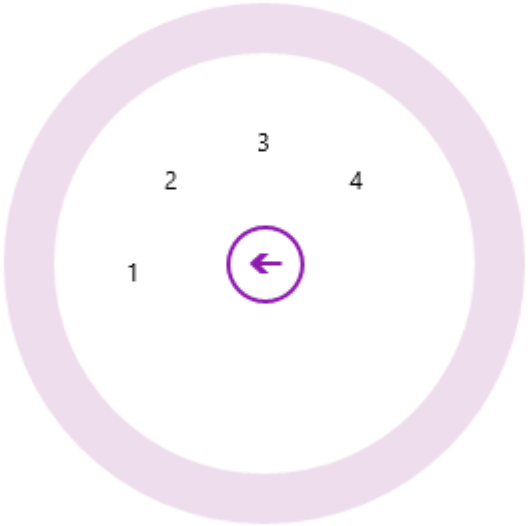
サブメニューのネスト

C1RadialMenu コントロールは、サブメニューを保持することができます。これらのサブメニューは、**C1RadialMenuItem** が他の **C1RadialMenuItem** タグ内にネストしていると作成されます。次に例を示します。

XAML

```
c1radial:C1RadialMenuItem Header="一番">
     <c1radial:C1RadialMenuItem Header="二番">
          <c1radial:C1RadialMenuItem Header="三番">
               <c1radial:C1RadialMenuItem Header="四番">
                    <c1radial:C1RadialMenuItem Header="五番"/>
</c1:C1RadialMenuItem>
```

この XAML マークアップを別の **C1RadialMenu** の開始タグと終了タグの間に配置すると、次のようなサブメニューが作成されます。

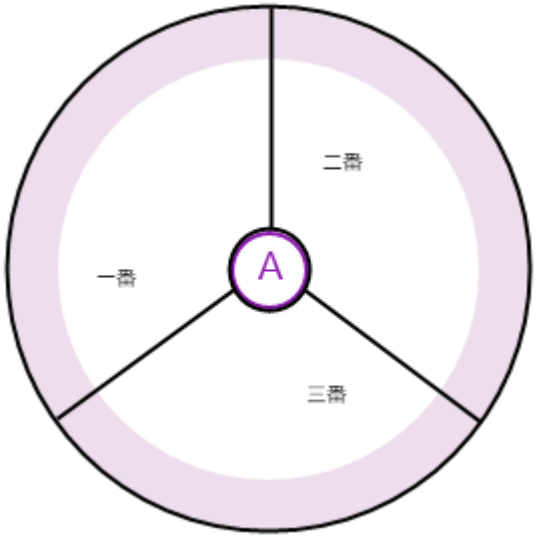


必要な数だけラジアルメニューをネストできますが、使いやすさの観点から、1つの階層にサブメニューを2、3個以上は置かない方がよいでしょう。

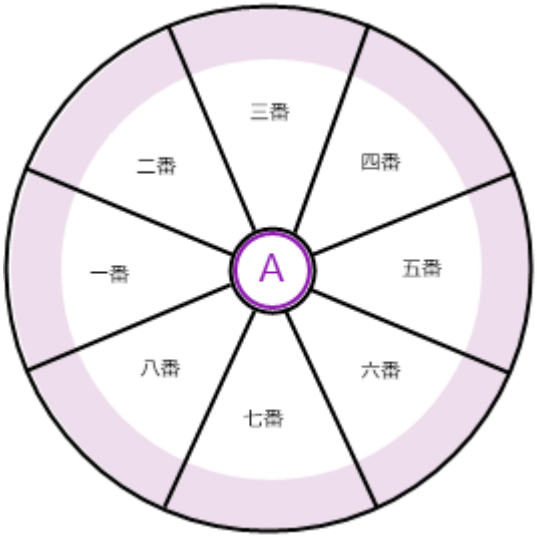
項目の配置

デフォルトでは、**C1RadialMenuItem** は 360 度のメニュー内に均等に配置されます。

C1RadialMenu に3つの項目がある場合は、各項目がそれぞれ **C1RadialMenu** の3分の1を占めるように配置されます。



また、**C1RadialMenu** に8つの項目がある場合は、次の図のようになります。

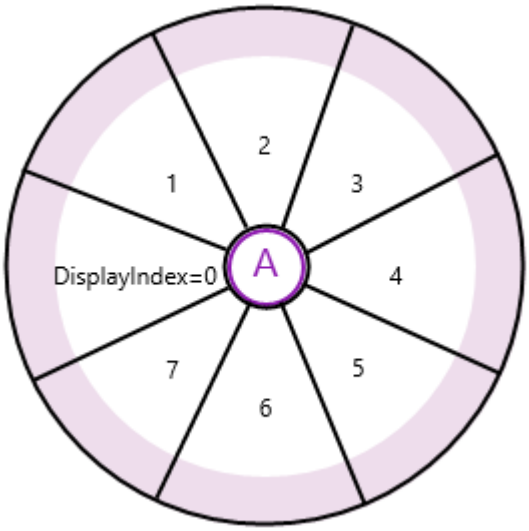


各項目は、それぞれラジアルメニューの8分の1を占めています。各 C1RadialMenuItem は、1つのセクター内に配置されます。セクターの数は、**SectorCount** プロパティを使用してカスタマイズできます。このプロパティは、メインの **C1RadialMenu** と子項目を含む任意の **C1RadialMenuItem** の両方に設定できます。

SectorCount と DisplayIndex プロパティを使用する

C1RadialMenuItem.DisplayIndexProperty プロパティは、0から始まるインデックスを使用して、**C1RadialMenuItem** の表示方法を定義します。

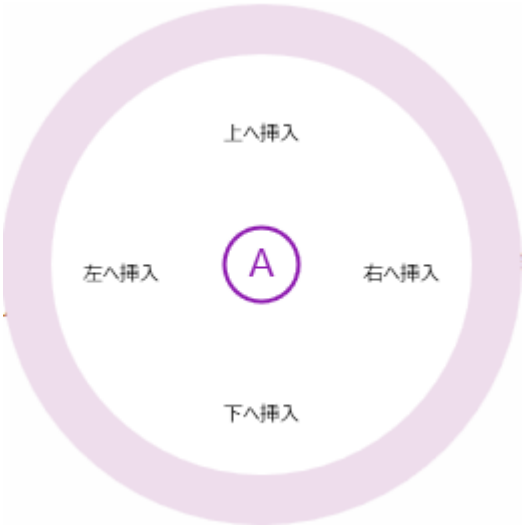
たとえば、**SectorCount** を "8" に設定した **C1RadialMenu** の表示インデックスは、次のようになります。インデックスは **C1RadialMenu** の中心の左側からメニューに沿って時計回りに大きくなります。



XAML

```
<Xaml:C1RadialMenu SectorCount="8">
    <Xaml:C1RadialMenuItem Header="左へ挿入"/>
    <Xaml:C1RadialMenuItem Header="上へ挿入" DisplayIndex="2" />
    <Xaml:C1RadialMenuItem Header="右へ挿入" DisplayIndex="4" />
    <Xaml:C1RadialMenuItem Header="下へ挿入" DisplayIndex="6" />
 </Xaml:C1RadialMenu>
```

上のマークアップは、次の図のような **C1RadialMenu** コントロールを作成します。



空の C1RadialMenuItems を使用する

他のメニュー項目の間に空の**C1RadialMenuItem** を挿入することにより、C1RadialMenuItemsの位置をカスタマイズすることができます。XAMLマークアップは次のようになります。

XAML

```
<Xaml:C1RadialMenuSectorCount="8">
   <Xaml:C1RadialMenuItemHeader="Item 1"/>
   <Xaml: C1RadialMenuItem/>
   <Xaml:C1RadialMenuItemHeader="Item 2" />
```

```
   <Xaml: C1RadialMenuItem/>
   <Xaml:C1RadialMenuItemHeader="Item 3" />
</Xaml:C1RadialMenu>
```

項目の選択

C1RadialMenuItem は、任意の数の子項目を持つことができます。**SelectedIndex** プロパティを設定して、選択された項目として表示する項目を制御できますが、デフォルトでは、**C1RadialMenuItem** はコレクションの最初の子項目を選択された項目として表示します。**AutoSelect** プロパティを設定しても、同じ設定になります。

C1RadialMenu コントロールは、**ShowSelectedItem** プロパティによって選択項目の記憶もサポートします。このプロパティを使用すると、設定済みの項目ではなく、最後に選択された項目を表示することができます。**ShowSelectedItem** を使用する場合は、**AutoSelect** プロパティを True に設定する必要があります。

したがって、右揃えテキストをよく使用するユーザーがいると思われる場合は、左揃えオプションや中央揃えオプションをデフォルトとして使用するのではなく、最後に選択された項目を表示することができます。**ShowSelectedItem** プロパティを "True" に設定すると、設定済みの項目や最初に選択された項目ではなく、ユーザーが最後に選択した項目が常に表示されます。

次のマークアップは、2つのサブメニューを含む簡単な **C1RadialMenu** を作成します。

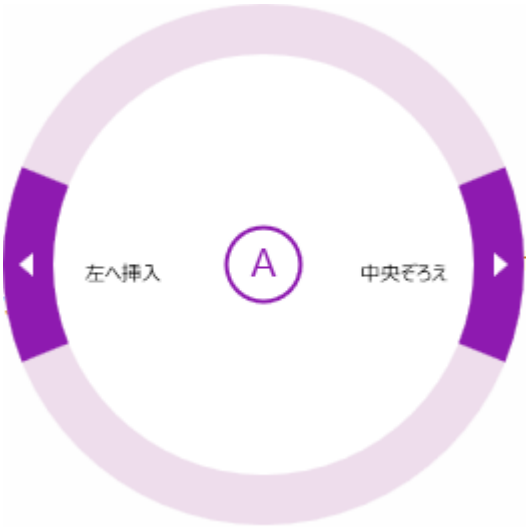
XAML

```
<Xaml:C1RadialMenu SectorCount="8" >
  <Xaml:C1RadialMenuItem Header="挿入" SectorCount="8" AutoSelect="True"
ShowSelectedItem="True" >
    <Xaml:C1RadialMenuItem Header="左へ挿入" />
    <Xaml:C1RadialMenuItem Header="上へ挿入" DisplayIndex="2" />
    <Xaml:C1RadialMenuItem Header="右へ挿入" DisplayIndex="4" />
    <Xaml:C1RadialMenuItem Header="下へ挿入" DisplayIndex="6" />
  </Xaml:C1RadialMenuItem>
  <Xaml:C1RadialMenuItem Header="ぞれえ" AutoSelect="True" SectorCount="8"
DisplayIndex="4" ShowSelectedItem="True">
    <Xaml:C1RadialMenuItem Header="右ぞれえ" />
    <Xaml:C1RadialMenuItem Header="左ぞれえ" />
    <Xaml:C1RadialMenuItem Header="中央ぞれえ" />
  </Xaml:C1RadialMenuItem>
</Xaml:C1RadialMenu>
```

アプリケーションを実行すると、次の図のようになります。

[**Align Right**]オプションの上の**展開**領域をタップすると、次のメニューが表示されます。

[**Align Center**]をタップしてから、**戻る**矢印をタップします。メインメニューは、次の図のようになります。



[**Align Right**]が最後に選択された[**Align Center**]に置き換えられたことに注目してください。

C1RadialMenu の外観

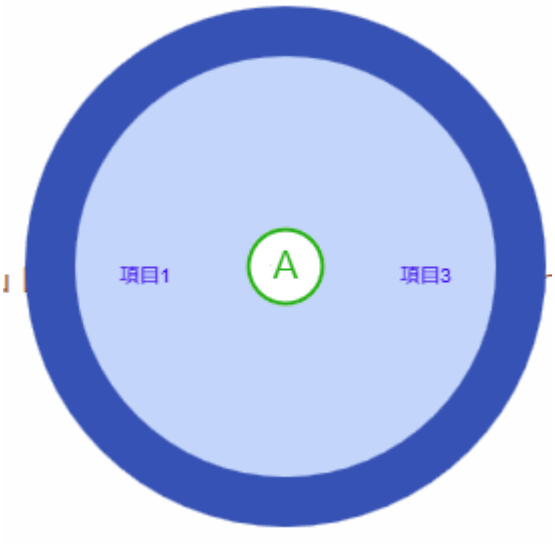
C1RadialMenu の外観のカスタマイズ

いくつかのプロパティを使用するだけで、**C1RadialMenu** の外観をすばやく簡単にカスタマイズできます。

次の手順を実行します。

1. 次のプロパティを **<Xaml: C1RadialMenu>** タグに追加します。
    - AccentBrush="#FF28B01A"
    - Background="#FFC3D5FB"
    - BorderBrush="#FF3652B4"
    - Foreground="#FF4210EE"
2. 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すると、次の図のようになります。



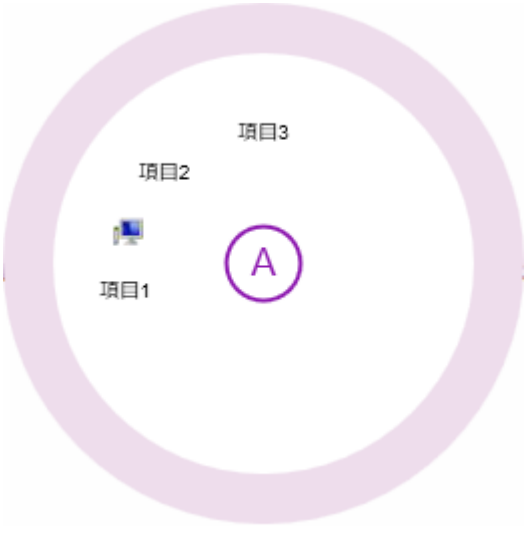
RadialMenu 項目へのアイコンの追加

次の手順を実行します。

1. プロジェクトにアイコン画像を追加します。12x12 ピクセルの画像が最適です。
2. **<Xaml:C1RadialMenuItem>** タグと **<Xaml:C1RadialMenuItem>** タグの間に次の XAML マークアップを追加し、**Source** プロパティの値を画像の名前に置き換えます。

```
XAML
<Xaml:C1RadialMenuItem.Icon>
     <Image Source="YourImage.png" Height="12" Width="12" Margin="5,0,0,0"/>
</Xaml:C1RadialMenuItem.Icon>
```

3. プロジェクトを実行します。次の図は、12x12 ピクセルのアイコン付きの **C1RadialMenuItem** です。



RangeSlider

WPF/Silverlight アプリケーションにスムーズな数値データ選択機能を追加します。**ComponentOne RangeSlider for WPF/Silverlight** は基本的なスライダコントロールを拡張して、1つではなく2つのサムを提供するため、ユーザーは単一の値ではなく範囲を選択できます。

以下に示す主要な機能をうまく活用して、**RangeSlider for WPF/Silverlight** を最大限に使用してください。

- **水平方向または垂直方向**

  1つのシンプルなプロパティで方向を変更します。水平方向および垂直方向の範囲スライダを作成します。

- **最小値と最大値の設定**

  範囲スライダの最小値と最大値を制御します。

- **カスタマイズ可能なスクロールボックス**

  C1RangeSlider のスクロールボックスをカスタマイズし、カスタムズームコントロールを作成します。

- **ClearStyle を使用して簡単に色を変更する**

  RangeSlider は、テンプレートを上書きしなくてもコントロールのブラシを簡単に変更できる **ComponentOne の ClearStyle** 技術をサポートします。Visual Studio でブラシのプロパティをいくつか設定するだけで、コントロールの各部のスタイルを簡単に設定できます。詳細については、「ComponentOne ClearStyle 技術」を参照してください。

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。

Range Sliderクイックスタート

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 **RangeSlider 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。

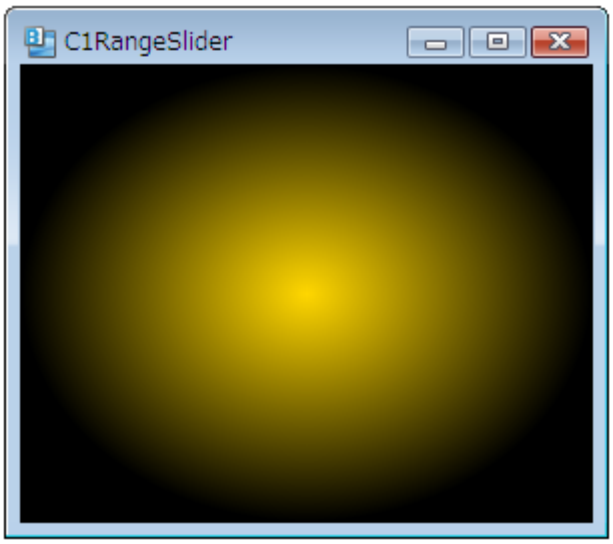
1. Visual Studio で、[**ファイル**]→[**新しいプロジェクト**]を選択します。
2. [新しいプロジェクト]ダイアログボックスの左ペインから言語を選択し、テンプレートリストから[アプリケーション]を選択します。
3. プロジェクトの名前(たとえば、「QuickStart」)を入力し、[OK]をクリックします。[新しい WPF/Silverlight アプリケーション]ダイアログボックスが表示されます。
4. [OK]をクリックしてデフォルト設定を受け入れ、[新しい WPF/Silverlight アプリケーション]ダイアログボックスを閉じると、プロジェクトが作成されます。
5. 参照の追加ダイアログボックスで、以下のアセンブリを見つけて選択し、[OK]をクリックしてプロジェクトに参照を追加します。
   - **WPF**: C1.WPF.4.dll
   - **Silverlight**: C1.Silverlight.5.dll
6. デザインビューでは、グリッドにクリックします。
7. 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。
8. [**デザイン**]ペインで、**Rectangle1** のサイズを変更して**グリッド**全体に拡大します。
9. [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>
        <RadialGradientBrush x:Name="colors">
            <GradientStop x:Name="goldcol" Color="Gold" Offset="0" />
            <GradientStop x:Name="blackcol" Color="Black" Offset="1" />
        </RadialGradientBrush>
    </Rectangle.Fill>
</Rectangle>
```

これで、黒と金色の放射状グラデーションが四角形に追加されます。

10. ここでアプリケーションを実行し、コントロールが次の図のように表示されていることを確認します。



これで、WPF/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、**C1RangeSlider** 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 WPF/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 のグラデーションを制御する **C1RangeSlider** コントロールを追加します。

次の手順に従います。

1. プロジェクトの XAML ウィンドウで、カーソルを **</Rectangle>** タグと **</Grid>** タグの間に置き、1回クリックします。
2. ツールボックスに移動し、[**C1RangeSlider**]アイコンをダブルクリックして、アプリケーション内の Rectangle の上にコントロールを追加します。
3. **x:Name="c1rs1"** を `<c1:C1RangeSlider>` タグに追加して、コントロールに名前を付けます。次のようになります。

XAML

```
<c1:C1RangeSlider x:Name="c1rs1">
```

それに一意の識別子を付けると、コードでそのコントロールにアクセスできるようになります。

4. <c1:C1RangeSlider> タグに **Margin="50"** を追加してマージンを追加します。次のようになります。

```
XAML
<c1:C1RangeSlider x:Name="c1rs1" Margin="50">
```

これで、各辺がグリッドの境界から同じ距離に設定されます。

5. <c1:C1RangeSlider> タグに **Orientation="Vertical"** を追加して、**Orientation** プロパティを Vertical に設定します。次のようになります。

```
Example Title
<c1:C1RangeSlider x:Name="c1rs1" Margin="50" Orientation="Vertical">
```

デフォルトでは、**Orientation** が Horizontal に設定されているため、コントロールが水平方向に表示されます。

6. <c1:C1RangeSlider> タグに **UpperValue="1"** を追加して、**UpperValue** プロパティを1に設定します。次のようになります。

```
XAML
<c1:C1RangeSlider x:Name="c1rs1" Margin="50" Orientation="Vertical"
UpperValue="1">
```

これで、上のサムは1から始まります。

7. <c1:C1RangeSlider> タグに **Maximum="1"** を追加して、**Maximum** プロパティを1に設定します。次のようになります。

```
XAML
<c1:C1RangeSlider x:Name="c1rs1" Margin="50" Orientation="Vertical"
UpperValue="1" Maximum="1">
```

これで、ユーザーは1より大きい値を選択できなくなります。

8. <c1:C1RangeSlider> タグに **ValueChange="0.1"** を追加して、**ValueChange** プロパティを 0.1 に設定します。次のようになります。

```
XAML
<c1:C1RangeSlider x:Name="c1rs1" Margin="50" Orientation="Vertical"
UpperValue="1" Maximum="1" ValueChange="0.1">
```

これで、実行時にスライダをクリックすると、サムが 0.1 単位で移動します。

9. <c1:C1RangeSlider> タグに **Opacity="0.8"** を追加して、**Opacity** プロパティを「0.8」に設定します。次のようになります。:

```
Example Title
<c1:C1RangeSlider x:Name="c1rs1" Margin="50" Orientation="Vertical"
UpperValue="1" Maximum="1" ValueChange="0.1" Opacity="0.8">
```

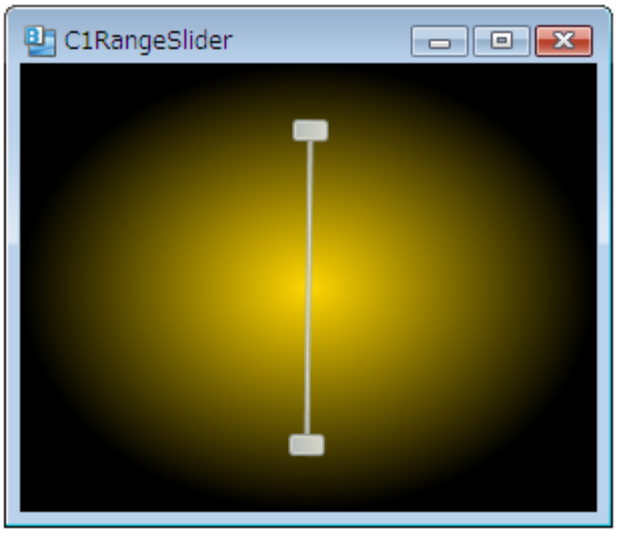
デフォルトで、このプロパティは1に設定されているため、コントロールは完全に不透明になります。これを小さい数値に変更すると、コントロールは少し透明になります。

10. <c1:C1RangeSlider> タグに **LowerValueChanged="c1rs1_LowerValueChanged"** **UpperValueChanged="c1rs1_UpperValueChanged"** を追加してイベントハンドラを指定します。次のようになります。

XAML
`<c1:C1RangeSlider x:Name="c1rs1" Margin="50" Orientation="Vertical" UpperValue="1" Maximum="1" ValueChange="0.1" Opacity="0.8" LowerValueChanged="c1rs1_LowerValueChanged" UpperValueChanged="c1rs1_UpperValueChanged">`

これらのイベントハンドラのコードは、この後の手順で追加します。

11. ここでアプリケーションを実行し、コントロールが次の図のように表示されていることを確認します。



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ましたが、今はスライダを移動しても何も行われません。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 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。

WPF

1. **Window1** をダブルクリックして、**コード**ビューに切り替え、**Window1_Loaded** イベントハンドラを作成します。
2. **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`
C#
`using C1.WPF;`

3. WPFアプリケーション, **Window1_Loaded** 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 Window1_Loaded(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles MyBase.Loaded
    UpdateGradient()
End Sub
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    UpdateGradient();
}
```

4. **Window1_Loaded** イベントハンドラ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 IsLoaded Then
        Me.goldcol.Offset = Me.c1rs1.LowerValue
        Me.blackcol.Offset = Me.c1rs1.UpperValue
    End If
End Sub
```

C#

```
UpdateGradient()
{
    if (IsLoaded)
    {
        this.goldcol.Offset = this.c1rs1.LowerValue;
        this.blackcol.Offset = this.c1rs1.UpperValue;
    }
}
```

5. [**デザイン**]ビューに戻ります。
6. **C1RangeSlider** コントロールをクリックして選択し、[プロパティ]ウィンドウに移動します。
7. [プロパティ]ウィンドウの上部にある稲妻の[**イベント**]アイコンをクリックして、イベントを表示します。
8. **LowerValueChanged** イベントをダブルクリックして、**[コード]**ビューに切り替え、**c1rs1_LowerValueChanged** イベントハンドラを作成します。**[デザイン]**ビューに戻り、**UpperValueChanged** イベントでこの手順を繰り返して、**C1rs1_UpperValueChanged** イベントハンドラを作成します。
9. **c1rs1_LowerValueChanged** 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 c1rs1_LowerValueChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles c1rs1.LowerValueChanged
    UpdateGradient()
End Sub
```

C#

```
private void c1rs1_LowerValueChanged(object sender, EventArgs e)
```

```
{
    UpdateGradient();
}
```

10. **C1rs1_UpperValueChanged** 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 c1rs1_UpperValueChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles c1rs1.UpperValueChanged
    UpdateGradient()
End Sub
```

C#

```
c1rs1_UpperValueChanged(object sender, EventArgs 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。

Silverlight

1. [**ビュー**]→[**コード**]を選択してコードビューに切り替えます。
2. **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
```

C#

```
using C1.Silverlight;
```

3. 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 c1rs1 IsNot Nothing Then
        Me.goldcol.Offset = Me.c1rs1.LowerValue
        Me.blackcol.Offset = Me.c1rs1.UpperValue
    End If
End Sub
```

C#

```
UpdateGradient()
{
    if (c1rs1 != null)
    {
        this.goldcol.Offset = this.c1rs1.LowerValue;
        this.blackcol.Offset = this.c1rs1.UpperValue;
    }
```

```
}
```

4. **デザイン**ビューに戻ります。
5. **C1RangeSlider** コントロールをクリックして選択し、[プロパティ]ウィンドウに移動します。
6. [プロパティ]ウィンドウの上部にある稲妻の[**イベント**]アイコンをクリックして、イベントを表示します。
7. **LowerValueChanged** イベントをダブルクリックして、**コード**ビューに切り替え、**C1rs1_LowerValueChanged** イベントハンドラを作成します。デザインビューに戻り、**UpperValueChanged** イベントでこの手順を繰り返して、**C1rs1_UpperValueChanged** イベントハンドラを作成します。
8. **c1rs1_LowerValueChanged** 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 c1rs1_LowerValueChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles c1rs1.LowerValueChanged
    UpdateGradient()
End Sub
```

C#

```
private void c1rs1_LowerValueChanged(object sender, EventArgs e)
{
    UpdateGradient();
}
```

9. **c1rs1_UpperValueChanged** 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 c1rs1_UpperValueChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles c1rs1.UpperValueChanged
    UpdateGradient()
End Sub
```

C#

```
c1rs1_UpperValueChanged(object sender, EventArgs 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。

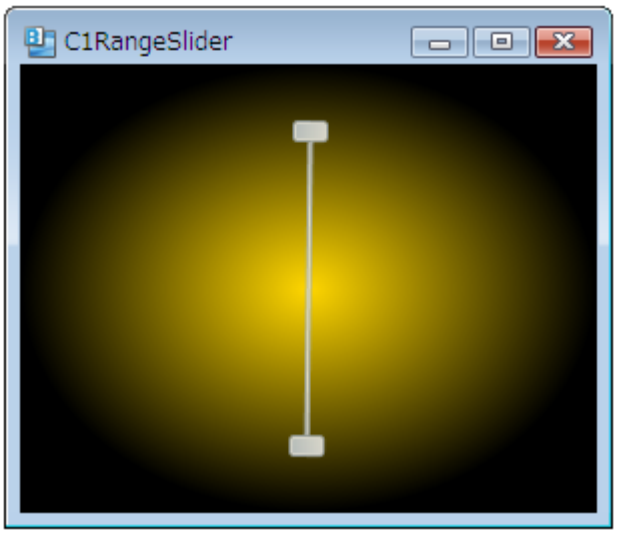
手順 4:アプリケーションの実行

これまでに WPF/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。

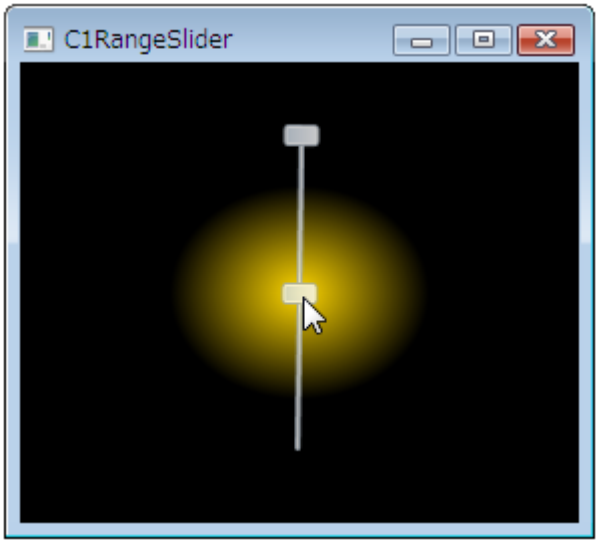
アプリケーションを実行し、**RangeSlider for WPF/Silverlight** の実行時の動作を確認するには、次の手順に従います。

1. [プロジェクト]メニューから[ソリューションのテスト]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。

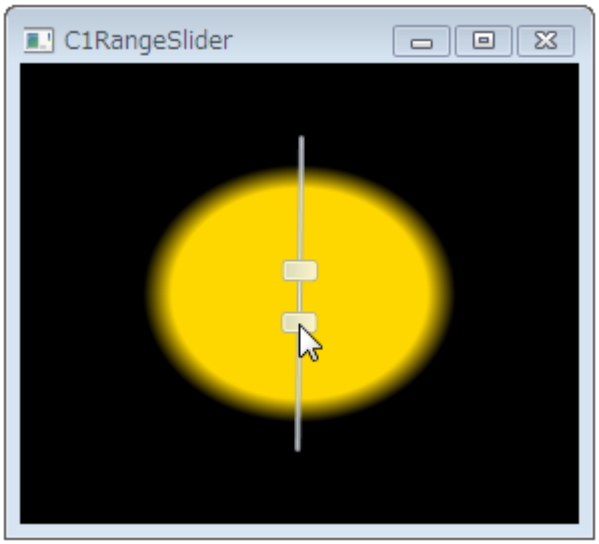
   アプリケーションは次の図のように表示されます。



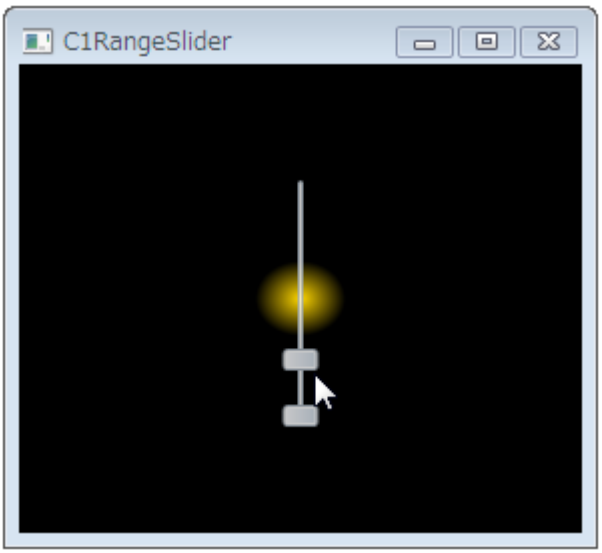
2. 上のサムを下に移動します。グラデーションの外観が変化します。



3. 下のサムを上に移動すると、グラデーション効果の幅が小さくなります。



4. 1つのサムの間をクリックし、スライダトラックに沿ってカーソルを下にドラッグします。両方のサムが一緒に移動します。

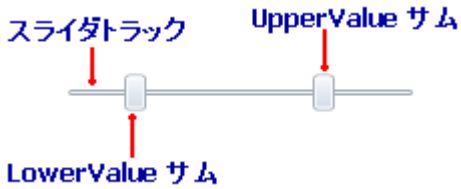


おめでとうございます!

これで、**RangeSlider for WPF/Silverlight** クイックスタートは完了です。**RangeSlider for WPF/Silverlight**アプリケーションを作成し、コントロールの外観と動作をカスタマイズし、アプリケーションの実行時機能をいくつか確認することができました。

RangeSlider要素

RangeSlider for WPF/Silverlight に含まれる **C1RangeSlider** コントロールは、通常のスライダの機能を超えて、値の範囲の選択に使用する2つのサムが付いた簡単な入力コントロールです。[XAML]ウィンドウに追加された **C1RangeSlider** コントロールは、完全な機能を備えたスライダコントロールになり、それをさらにカスタマイズできます。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。



RangeSlider特長

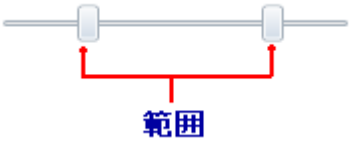
最小と最大

Minimum プロパティと**Maximum** プロパティは、**LowerValue** のサムは **Minimum** より小さい値に設定できず、**UpperValue** のサムは **Maximum** より大きい値に設定できません。

デフォルトでは、**Minimum** プロパティは **0** に、**Maximum** プロパティは **100** に設定されています。

サム値の設定

UpperValue と**LowerValue** のサムは、スライダトラックに沿って移動します。デフォルトでは、**UpperValue** プロパティは 100 に、**LowerValue** プロパティは 0 に設定されています。値の範囲は、**UpperValue** と LowerValue の差によって決定されます。



サム値を設定する

C1RangeSlider コントロールには、値の範囲の選択に使用する2つのサムがあります。**UpperValue** と **LowerValue** のサムは、スライダトラックに沿って移動します。デフォルトでは、**UpperValue** プロパティは 100 に、**LowerValue** プロパティは 0 に設定されています。

XAML の場合

たとえば、**UpperValue** プロパティを「**90**」、**LowerValue** プロパティを「**10**」に設定するには、**UpperValue="90" LowerValue="10"** を `<c1:C1RangeSlider>` タグに追加します。次のようになります。

XAML

```
<c1:C1RangeSlider Name="C1RangeSlider1" Width="26" Height="18" UpperValue="90"
LowerValue="10" />
```

コードの場合

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

Visual Basic

```
Me.C1RangeSlider1.LowerValue = 10
Me.C1RangeSlider1.UpperValue = 90
```

C#

```
this.c1RangeSlider1.LowerValue = 10;
this.c1RangeSlider1.UpperValue = 90;
```

設計時

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

1. **C1RangeSlider** 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、**LowerValue** プロパティの横にあるテキストボックスに値（たとえば、**10**）を入力します。
3. [プロパティ]ウィンドウで、**UpperValue** プロパティの横にあるテキストボックスに値（たとえば、**90**）を入力します。

   これで、**UpperValue** プロパティと **LowerValue** プロパティは指定された値に設定されます。

値の変更単位を設定する

XAML の場合

ValueChange プロパティを5に設定するには、次のように示すように **ValueChange="5"** を `<c1:C1RangeSlider>` タグに追加します。次のようになります。

XAML

```
<c1:C1RangeSlider Name="C1RangeSlider1" Height="18" Width="26" ValueChange="5" />
```

コードの場合

ValueChange プロパティを設定するには、プロジェクトに次のコードを追加します。

Visual Basic

```
Me.C1RangeSlider1.ValueChange = 5
```

C#

```
this.c1RangeSlider1.ValueChange = 5;
```

設計時

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

1. **C1RangeSlider** 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、**ValueChange** プロパティの横にあるテキストボックスに値（たとえば、**5**）を入力します。

   これで、**ValueChange** プロパティは指定された値に設定されます。

オリエンテーション

Orientation プロパティを使用して、コントロールを水平と垂直のどちらかに向けることができます。デフォルトでは、アプリケーションに追加されたコントロールは、最初に水平方向に表示されます。**Orientation** プロパティを使用すると、[プロパティ]ウィンドウ、[XAML]、または[コード]で方向を簡単に変更できます。

XAML の場合

Orientation プロパティを Vertical に設定するには、`<c1:C1RangeSlider>` タグに **Orientation="Vertical"** を追加します。次のようになります。

XAML

```
<c1:C1RangeSlider Name="C1RangeSlider1" Width="26" Orientation="Vertical" />
```

コードの場合

Orientation プロパティを **Vertical** 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.C1RangeSlider1.Orientation = Orientation.Vertical
```

C#

```
this.c1RangeSlider1.Orientation = Orientation.Vertical;
```

設計時

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

1. **C1RangeSlider** 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、**Orientation** プロパティを見つけます。
3. **Orientation** プロパティの横にあるドロップダウン矢印をクリックし、Vertical を選択します。

   これで、**Orientation** プロパティが変更され、コントロールが垂直に表示されます

アプリケーションの実行と動作の確認

C1RangeSlider コントロールが垂直に表示されます。

TabControl

TabControl for WPF/Silverlight を使用することにより、効率的で体系化された方法でコンテンツを配置できます。**C1TabControl** コントロールを使用すると、タブおよび対応するコンテンツページを追加できるため、画面領域の使用量を削減しながら、大量の情報を届けることが可能になります。

次のような主要機能を利用して、**TabControl for WPF/Silverlight** を最大限に活用してください。

- **スクロール要素**

  C1TabControl コントロールでは、タブがコントロールの指定された幅または高さを超えたとき、スクロールボタンが自動的に挿入されます。

- **タブストリップの配置**

  C1TabControl コントロールは、コントロールの上下左右のいずれにも配置できます。

- **タブクローズオプション**

  ユーザーがタブを閉じることができるかどうか、および[閉じる]ボタンを表示する場所を制御します。Visual Studio での[ドキュメント]タブと同様に、[閉じる]ボタンを各タブ項目内またはタブストリップ外のグローバルな場所に表示できます。

- **メニューでのタブの表示**

  C1TabControl コントロールには、**TabStripMenuVisibility** プロパティを「Visible」に設定すると表示されるオプションのタブメニューがあります。ユーザーはタブメニューを使用することにより、ドロップダウンメニューからタブページを開くことができます。

- **ヘッダーの形状をカスタマイズします**

TabControl の **TabItemShape** プロパティを Rounded、Rectangle、および Sloped から選択することにより、タブヘッダーの形状を変更できます。これは、コントロールのテンプレートを変更してタブヘッダーの形状を変更する必要がないため、デザイナでないユーザーに適しています。

- **新しいタブ項目**

  C1TabControl には、新しいタブの外観と操作性を Microsoft Internet Explorer 8 と同様の、他のタブと同じ統一的なものになるようにするサポートが組み込まれています。

- **ヘッダーでの項目の配置**

  タブ項目を背面の最右端または背面の最左端で重ねるかどうかを定義します。選択された項目は常に最前面に置かれます。

- **背景の変更**

  タブの外観を維持しながら、その背景を変更できます。これは簡単な機能のように見えますが、テンプレート全体をカスタマイズすることなく背景を変更できるタブコントロールは、現在、**C1TabControl** 以外は販売されていません。

- **ヘッダーの重なり**

  タブ項目ヘッダー間の重なりをカスタマイズすることにより、Microsoft Visual Studio の[ドキュメント]タブのような階段状のタブを表示できます。

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

クイックスタート

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 TabControl 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。
3. [OK]をクリックしてデフォルト設定を受け入れ、[新しい WPF/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。
4. MainPage.xaml ファイルが開きます。
5. 参照の追加ダイアログボックスで、以下のアセンブリを見つけて選択し、[OK]をクリックしてプロジェクトに参照を追加します。
   - **WPF**: C1.WPF.4.dll
   - **Silverlight**: C1.Silverlight.5.dll
6. ツールボックスで、**C1TabControl** アイコンをダブルクリックして、**C1TabControl** コントロールを アプリケーションに追加します。
7. 次の手順に従って、コントロールに3つのタブを追加します。
   1. **C1TabControl** 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、**Items** 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。

8. [**追加**]ボタンを3回クリックして、3つの **C1TabItem** 項目を **C1TabControl** コントロールに追加します。
9. [**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。

これで、**TabControl for WPF/Silverlight** クイックスタートの最初の手順は終了です。この手順では、プロジェクトを作成し、3つのタブページを持つ **C1TabControl** コントロールを追加しました。次の手順では、コントロールにタブとタブページを追加します。

手順 2:タブページの追加

前の手順では、WPF/Silverlightプロジェクトを作成してから、3つのタブページを持つ C1TabControl コントロールを追加しました。この手順では、各タブページをカスタマイズします。

次の手順に従います。

1. XAML ビューに切り替えます。
2. **Header="タブ1"** を1番目の **<c1:C1TabItem>** 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:C1TabItem Header="タブ1"/>
```

3. **Header="タブ2"** と **Content="Content プロパティに設定された文字列。"** を 2番目の **<c1:C1TabItem>** 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:C1TabItem Header="タブ2" Content="Content プロパティに設定された文字列。"/>
```

4. **Header="タブ3"**、**Content="このタブを閉じることはできません。"**、および **CanUserClose="False"** を3番目の **<c1:C1TabItem>** 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:C1TabItem Header="タブ3" Content="このタブを閉じることはできません。"
CanUserClose="False"/>
```

   CanUserClose プロパティを False に設定すると、実行時にユーザーはタブを閉じることができません。

5. 次の手順に従って、最初のタブに **Calendar** コントロールを追加します。
   1. [デザイン]ビューに切り替えて、最初のタブを選択します。
   2. [ツール]パネルで、**Calendar** アイコンをダブルクリックして、**Calendar** コントロールをタブに追加します。
   3. **Calendar** コントロールを選択してから、次のプロパティを設定します。
      - **Height** プロパティを「Auto」に設定します。
      - **Width** プロパティを「Auto」に設定します。

これで、手順 2 が完了しました。この手順では、**C1TabControl** コントロールの3つのタブページをカスタマイズしました。次の手順では、**C1TabControl** コントロールの外観と動作をカスタマイズします。

手順 3:コントロールのカスタマイズ

前の手順では、カスタマイズした3つのタブページを **C1TabControl** コントロールに追加しました。この手順では、いくつかの

プロパティを設定し、**C1TabControl** 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. **C1TabControl** コントロールを選択します。
2. [プロパティ]ウィンドウで、次のプロパティを設定します。
   - **Height** プロパティを「200」に設定します。
   - **Width** プロパティを「300」に設定します。
   - **TabItemClose** プロパティを **InEachTab** に設定します。これにより、[閉じる]ボタンが3番目以外の各タブに追加されます。3番目のタブは、クイックスタートの最後の手順で閉じることを禁止しました。
   - **TabItemShape** プロパティを **Sloped** に設定します。これにより、タブ項目の形状が Office フォルダのタブに似せて変更されます。
   - **TabStripMenuVisibility** プロパティを **Visible** に設定します。
   - **TabStripPlacement** プロパティを **Bottom** に設定します。これにより、タブストリップがコントロールの下端に配置されます。

これで、手順3が完了し、**C1TabControl** コントロールの機能がカスタマイズされました。次の手順では、プログラムを実行し、このクイックスタートで行ったことを確認します。.

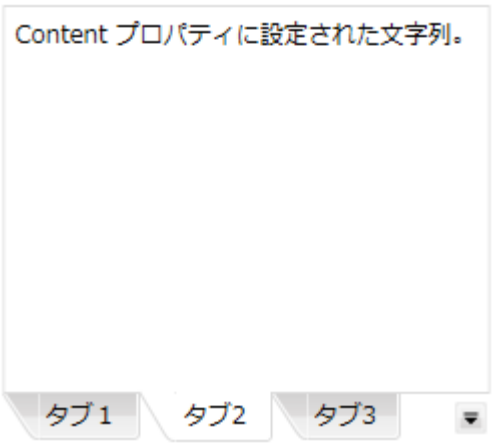
手順 4:アプリケーションの実行

前の手順では、**C1TabControl** コントロールのプロジェクトを作成し、そのコントロールにタブページを追加し、コントロールの外観と動作を変更しました。この手順では、プログラムを実行し、**C1TabControl** コントロールに加えたすべての変更を確認します。

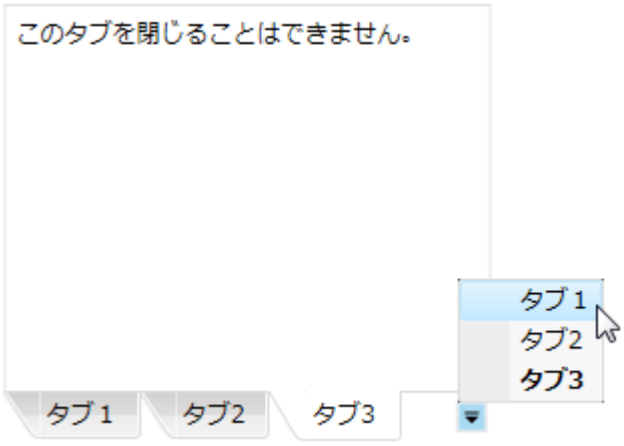
次の手順に従います。

1. [F5]キーを押してプロジェクトを実行します。**C1TabControl** コントロールのタブストリップがコントロールの下端に沿って並び、端が斜めになったタブが配置されていることを確認します。また、最初のタブページがロードされ、コンテンツとして **Calendar** コントロールが配置されていることも確認します。

2. [**タブ2**]をクリックして、そのコンテンツが文字列で設定された **Content** プロパティと同じであることを確認します。

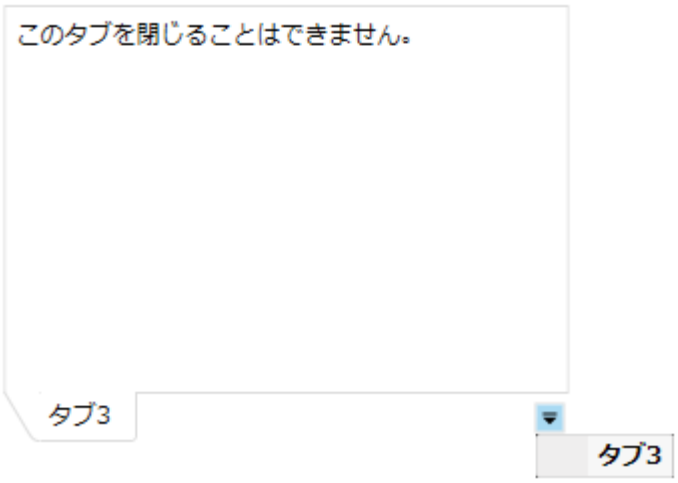


3. ドロップダウンボタンをクリックしてタブメニューを開き、[**タブ1**]を選択します。



[**タブ1**]に再度フォーカスが置かれます。

4. [**タブ1**]の[閉じる]ボタンをクリックしてタブを閉じます。
5. [**タブ2**]の[閉じる]ボタンをクリックしてタブを閉じます。[**タブ3**]のコンテンツにフォーカスが置かれます。このタブは、前の手順で **CanUserClose** プロパティを「False」に設定したため、閉じることはできません。
6. メニューボタンをクリックして、現在のタブページ([**タブ3**])のみが表示されて他の2つのタブは閉じていることを確認します。

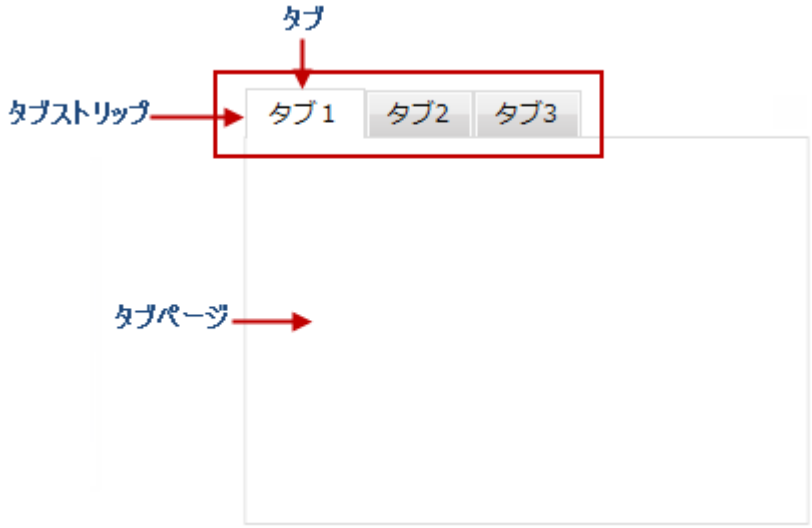


おめでとうございます!

これで、**TabControl for WPF/Silverlight** クイックスタートの4つの手順のすべてが完了しました。このクイックスタートでは、全面的にカスタマイズした **C1TabControl** のプロジェクトを作成しました。

C1TabControl の要素

C1TabControl は3つの要素(タブ、タブストリップ、およびタブページ)で構成され、これらの組み合わせによって **C1TabControl** コントロールが完成します。



以下のトピックでは、**C1TabControl** コントロールの各要素について説明します。

タブ

C1TabControl コントロールの各タブは、**C1TabItem** クラスによって表されます。タブのヘッダーは、**C1TabItem** の **Header** プロパティによって公開されます。各タブは、タブページに関連付けられています(「タブページ」を参照)。

タブは、角丸、斜め、または四角形で表示できます。タブをページに追加したときのデフォルトの外観は斜めです。これは、Office フォルダのタブに似ています。

TabItemClose プロパティを「**InEachTab**」に設定することにより、各タブに[閉じる]ボタンを追加できます。これによってユーザーは、任意のタブをコントロールで閉じることができます。ただし、タブの **CanUserClose** プロパティを「**False**」に設定すると、ユーザーが特定のタブを閉じることを禁止できます。

タグを追加

- Example Title

```
<c1:C1TabControl>
  <c1:C1TabItem Content="c1TabItem">
  </c1:C1TabItem>
</c1:C1TabControl>
```

- Visual Basic

```
'Create the tab and add content
```

```
Dim C1TabItem1 As New C1TabItem()
C1TabItem1.Content = "C1TabItem1"
'Add the tab to the control
c1TabControl1.Items.Add(C1TabItem1)
```

C#

```
//Create the tab and add content
c1TabItem c1TabItem1 = new c1TabItem();
c1TabItem1.Content = "c1TabItem1";
//Add the tab to the control
c1TabControl1.Items.Add(c1TabItem1);
```

タブヘッダーを指定する

このトピックでは、Blend、XAML、およびコードを使用して、Header プロパティを設定することでヘッダーをタブに追加します。このトピックでは、少なくとも1つの C1TabItem が既に C1TabControl コントロールに追加されていることが前提です。

XAML

```
<c1:C1TabItem Header="Hello World"/>
```

Visual Basic

```
C1TabItem1.Header = Hello World
```

C#

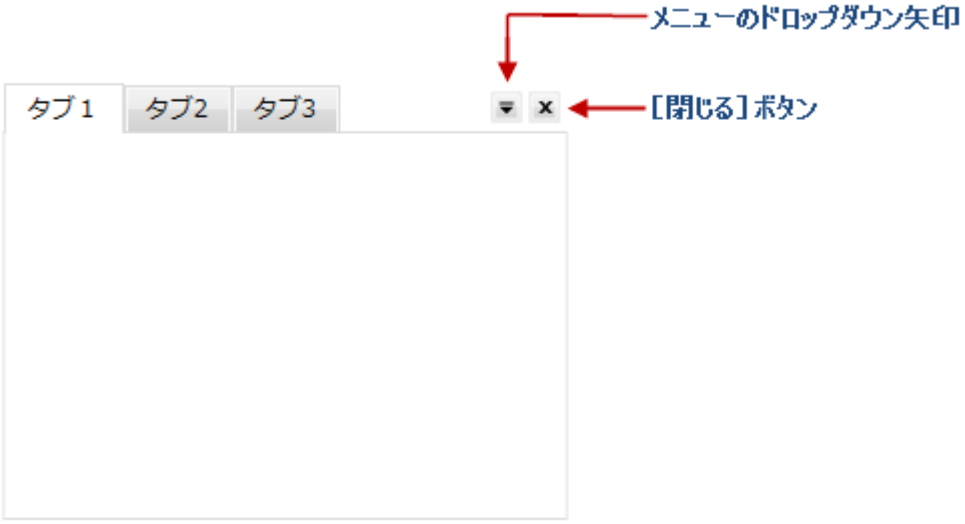
```
c1TabItem1.Header = Hello World;
```

タブストリップ

C1TabControl コントロールのタブストリップは、1つ以上の **C1TabItem** 項目をコントロールに追加して作成します。ストリップ内の各タブは、タブページに関連付けられています(「**タブ**」および「**タブページ**」を参照)。

デフォルトで、タブストリップは **C1TabControl** コントロールの上部に表示されます。ただし、**TabStripPlacement** プロパティの設定によってストリップの位置を調整できます。また、**TabStripOverlap** プロパティと **TabStripOverlapDirection** プロパティの設定によってタブの重なりを調整することもできます。

タブストリップには、2つのオプション項目([閉じる]ボタンとメニューのドロップダウン矢印)も含めることができます。[閉じる]ボタンをクリックすると、選択したタブが閉じます。メニューのドロップダウン矢印をクリックすると、ドロップダウンメニューが表示されます。ユーザーはこのメニューから別のタブを開くことができます。



タブページ

タブ(**C1TabItem**)を **C1TabControl** コントロールに追加すると、初期状態として空のスペースを持つタブページがそれに対応して作成されます。タブページには、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]ビューを使用すると、簡単なドラッグ アンド ドロップ操作でタブページに要素を追加できます。

タブページ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** 要素をタブページに追加します。実行時にタブページに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。

C1TabItem 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、**C1TabItem** 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、タブページに複数のコントロールを表示できます。

タブページへのコンテンツの追加

- **XAML の場合**

  タブページに単純な要素(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:C1TabItem Content="Button" Height="Auto" Width="Auto"/>
```

ただし、タブページに、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。

XMAL

```
<c1:C1TabItem>
  <c1:C1TabItem.Content>
    <StackPanel>
      <TextBlock Text="Hello"/>
      <TextBlock Text="World"/>
    </StackPanel>
  </c1:C1TabItem.Content>
</c1:C1TabItem>
```

- **コードの場合**

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
c1TabItem1.Content = (NewButton)
```

C#

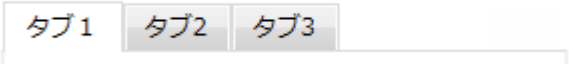
```
/Button コントロールを作成します
Button NewButton = new Button();
NewButton.Content = "ボタン";
//Button コントロールの Width および Height プロパティを設定します
NewButton.Width = double.NaN;
NewButton.Height = double.NaN;
//ボタンをコンテンツ領域に追加します
c1TabItem1.Content = (NewButton);
```

TabControlの特長

タブ形状

C1TabControl コントロールのタブは、**四角形**、**角丸**、または**斜め**にすることができます。デフォルトのタブは角丸です。ただし、**C1TabControl コントロールの TabItemShape** プロパティを設定して、タブの形状を変更できます。

Rectangle



Rounded

Sloped

XAML の場合

タブの形状を変更するには、**TabItemShape="Sloped"** を **<c1:C1TabControl>** 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:C1TabControl TabItemShape="Sloped"></c1:C1TabControl>
```

コードの場合

次の手順に従います。

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

   C#

   ```
   c1TabControl1.TabItemShape = Sloped;
   ```

3. プログラムを実行します。

設計時

次の手順に従います

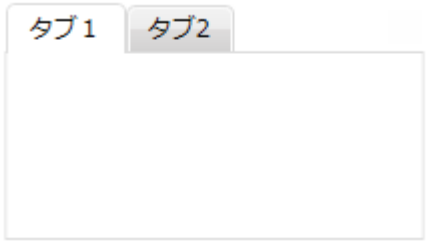
1. **C1TabControl** コントロールを選択します。
2. [プロパティ]ウィンドウで、**TabItemShape** ドロップダウン矢印をクリックし、リストから[**Sloped**]を選択します。.

下の図は、端が斜めになったタブを持つ **C1TabControl** を示します。

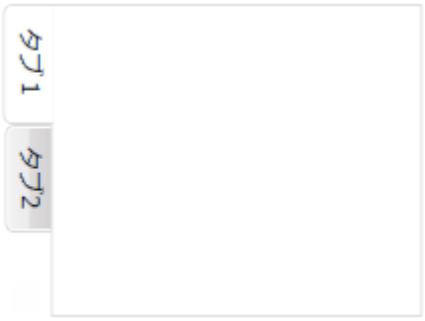
タブストリップの配置の変更

C1TabControl コントロールのタブストリップは、デフォルトでコントロールの上端に沿って表示されます。ただし、**TabStripPlacement** プロパティを「**Bottom**」、「**Left**」、または「**Right**」に設定して、タブストリップの位置を変更できます。

Top

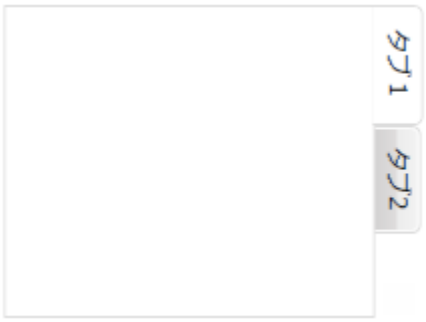


Left



Bottom

Right



XAML の場合

タブストリップの配置を変更するには、**TabStripPlacement="Right"** を **<c1:C1TabControl>** タブに追加します。マークアップは次のようになります。

```
XAML
<c1:C1TabControl TabStripPlacement="Right"></c1:C1TabControl>
```

コードの場合

次の手順に従います。

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

Visual Basic
```
C1TabControl1.TabStripPlacement = Right
```

C#
```
c1TabControl1.TabStripPlacement = Right;
```

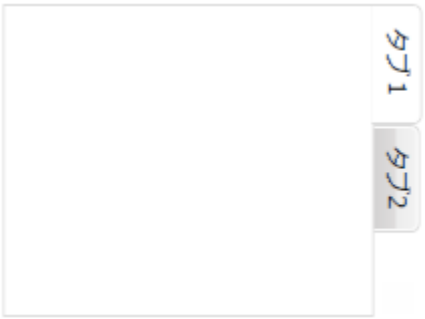
3. プログラムを実行します。

場合

次の手順に従います。

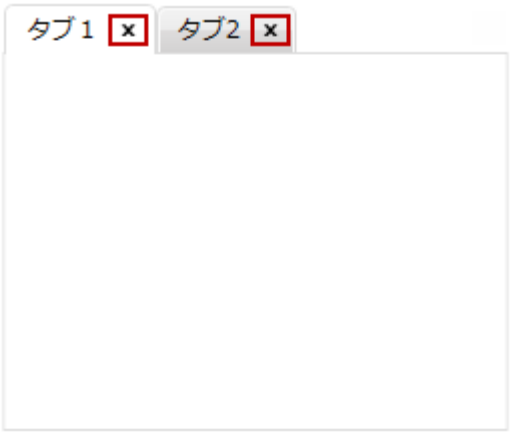
1. **C1TabControl** コントロールを選択します。
2. [プロパティ]ウィンドウで、**TabStripPlacement** ドロップダウン矢印をクリックし、リストから[Right]を選択します。

次の図は、タブストリップが右側に配置された **C1TabControl** コントロールを示します。



タブクローズ

TabItemClose プロパティを「**InEachTab**」に設定することにより、各タブに[閉じる]ボタンを追加できます。これにより、ユーザーは任意のタブをコントロールで閉じることができます。タブの[閉じる]ボタンは次のように表示されます。



必要に応じて、[閉じる]ボタンをタブ上に直接作成するのではなく、グローバルな[閉じる]ボタンをタブストリップ上に作成できます。グローバルな[閉じる]ボタンを追加するには、**TabItemClose** プロパティを「**GlobalClose**」に設定します。グローバルな[閉じる]ボタンをクリックすると、現在選択されているタブが閉じます。グローバルな[閉じる]ボタンは次のように表示されます。

XAML の場合

ユーザーが共通のボタンを使ってタブを閉じることを許可するには、**TabItemClose="GlobalClose"** を **<c1:C1TabControl>** タブに追加します。マークアップは次のようになります。

XAML

```
<c1:C1TabControl TabItemClose="GlobalClose"></c1:C1TabControl>
```

コードの場合

次の手順に従います。

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

Visual Basic

```
C1TabControl1.TabItemClose = GlobalClose
```

C#

```
c1TabControl1.TabItemClose = GlobalClose;
```

3. プログラムを実行します。

設計時

次の手順に従います。

1. **C1TabControl** コントロールを選択します。
2. [プロパティ]ウィンドウで、**TabItemClose** のドロップダウン矢印をクリックし、リストから[**GlobalClose**]を選択します。

XAML の場合

ユーザーによるタブのクローズを禁止するには、ユーザーによるクローズを禁止するタブの **<c1:C1TabItem>** タグに、**CanUserClose="False"** を追加します。XAML は次のようになります。

XAML

```
<c1:C1TabItem CanUserClose="False"></c1:C1TabControl>
```

コードの場合

次の手順に従います。

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

Visual Basic

```
C1TabItem1.CanUserClose = False
```

C#

```
c1TabItem1.CanUserClose = false;
```

3. プログラムを実行します。

設計時

次の手順に従います。

1. ユーザーが閉じることを許可しないタブを選択します。
2. [プロパティ]ウィンドウで、[**CanUserClose**]チェックボックスをオフにします。

オプションのタブメニュー

C1TabControl コントロールを使用して、ユーザーがメニューからタブとタブページを選択できるドロップダウンメニューを追加できます。このドロップダウンメニューは、有効にするとコントロールのタブストリップからアクセスできます。ドロップダウンメニューは次のようになります。

XAML の場合

メニューをタブストリップに追加するには、**TabStripMenuVisibility="Visible"** を **<c1:C1TabControl>** 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:C1TabControl TabStripMenuVisibility="Visible"></c1:C1TabControl>
```

コードの場合

次の手順に従います。

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

   Visual Basic

   ```
   C1TabControl1.TabStripMenuVisibility = Visible
   ```

   C#

   ```
   c1TabControl1.TabStripMenuVisibility = Visible;
   ```
3. プログラムを実行します。

設計時

次の手順に従います。

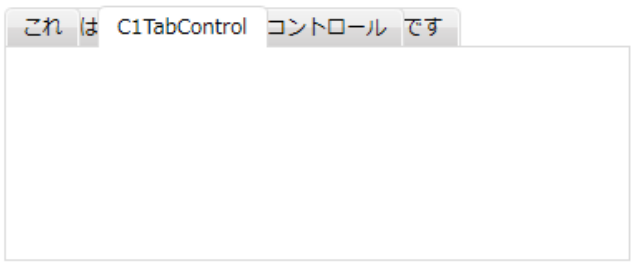
1. **C1TabControl** コントロールを選択します。
2. [プロパティ]ウィンドウで、**TabStripMenuVisibility** ドロップダウン矢印をクリックし、リストから[**Visible**]を選択します

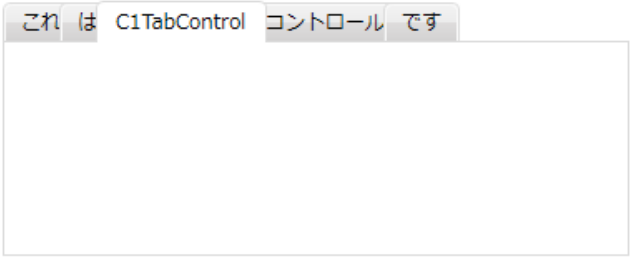
タブの重なり

TabStripOverlap プロパティと **TabStripOverlapDirection** プロパティを設定することにより、タブの重なりを制御できます(タスク別ヘルプについては、「タブストリップでのタブの重なり」を参照)。**TabStripOverlap** プロパティでタブの重なりのピクセル数を制御し、**TabStripOverlapDirection** プロパティで重なりの方向を設定する列挙値を選択します。**TabStripOverlapDirection** プロパティには3つの列挙値(**Right**、**Left**、および **RightLeftFromSelected**)があります。

Note: 次の各例で示す **TabStripOverlap** プロパティは、値が 10 に設定されています。

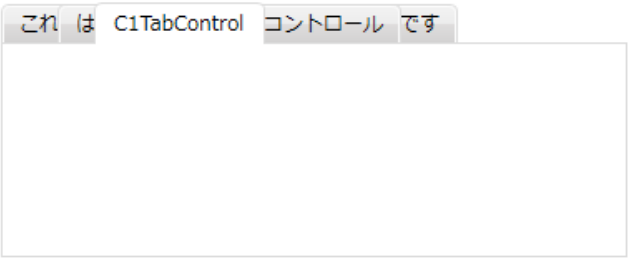
Right	**Left**
最右端のタブは後方に置かれ、選択されたタブは手前に置かれます。	最左端のタブは後方に置かれ、選択されたタブは手前に置かれます。





RightLeftFromSelected

最左端のタブは後方に、最右端のタブも後方に、選択されたタブは手前に置かれます。



TreeView

TreeView for WPF/Silverlight は、データ項目を階層的に表示します。このコントロールは、Windows フォームで使用される標準の TreeView コントロールに似ていますが、キーボードベースの検索、ドラッグアンドドロップ機能、自動検索、階層テンプレートなど、さらに多くの機能を備えています。

次の主要な機能を利用して、**TreeView for WPF/Silverlight** を最大限に活用してください。

- **TreeView 内のノードのドラッグアンドドロップ**

  TreeView は、ツリー内のドラッグアンドドロップ操作をサポートします。**AllowDragDrop** プロパティを true に設定するだけで、ツリー内のノードをドラッグして並べ替えることができます。

- **ドラッグアンドドロップ動作のカスタマイズ**

  TreeView は、ドラッグアンドドロップ動作をカスタマイズできるように、ドロップアンドドロップの操作中にイベントを発生します。たとえば、一部のノードのドラッグを禁止したり、一部のノードがドロップ先にならないように指定することができます。

- **自動検索**

  TreeView コントロールでは、自動検索を使用して、ノード内の1文字に簡単にジャンプできます。1文字を入力するだけで、特定のツリーノードに移動します。

- **階層テンプレート**

  C1TreeViewItem クラスをサブクラス化しなくても、ノードタイプごとに異なるテンプレートを使用できます。

- **ノードのカスタマイズ**

  ノードヘッダーはコンテンツ要素なので、任意の種類の要素をホストできます。画像、チェックボックスなど、アプリケーションに必要なあらゆる要素を追加できます。

- **キーボードによる移動**

  カーソルキーを使用して、ノード間を移動したり、ノードを展開または折りたたむことができます。また、自動検索機能を

使用して、特定のノードをすばやく簡単に見つけることができます。

- **ClearStyle 技術のサポート**

  TreeView for WPF/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、**C1TreeView** コントロールのスタイルを簡単に設定できます。

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

クイックスタート

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 **TreeView for WPF/Silverlight** を使用する 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[アプリケーション]を選択します。
3. プロジェクトの名前(たとえば、「QuickStart」)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。
4. [**OK**]をクリックしてデフォルト設定を受け入れ、[新しい WPF 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。
5. 参照の追加ダイアログボックスで、以下のアセンブリを見つけて選択し、[OK]をクリックしてプロジェクトに参照を追加します。
   - **WPF**: C1.WPF.4.dll
   - **Silverlight**: C1.Silverlight.5.dll
6. 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。

   ツールボックスに移動し、[**C1TreeView**]アイコンをダブルクリックして、ツリービューコントロールを追加します。

   Note: C1.WPF or C1. Silverlight.5 名前空間と **<c1:C1TreeView></c1:C1TreeView>** タグがプロジェクトに追加されていることがわかります。
7. **<c1:C1TreeView></c1:C1TreeView>** タグの間にあるすべてのマークアップを削除します。XAML マークアップは次のようになります。

   XAML
   ```
   <Grid>
     <c1:C1TreeView></c1:C1TreeView>
   </Grid>
   ```
8. **x:Name="Tree"** を **<c1:C1TreeView />** タグに追加して、グリッドに名前を付けます。次のようになります。

   XAML
   ```
   <c1:C1TreeView Name="Tree">
   ```

   コントロールに一意の識別子を付けると、コードでその **C1TreeView** コントロールにアクセスできるようになります。

C1TreeView 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、**C1TreeView** 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：項目の追加

このレッスンでは、XAML マークアップおよび分離コードファイルで、C1TreeView コントロールに静的な C1TreeView 項目を追加する方法について説明します。

XAML の場合

XAML で **C1TreeView** コントロールに静的な **C1TreeViewItem** を追加するには

1. **C1TreeViewItem** を追加して、「**ジャンル一覧**」という名前の最上位ノードを作成します。`<c1:C1TreeViewItem>` タグ内に、`Header="ジャンル一覧` を追加します。これにより、最上位ノードが作成されます。XAML マークアップは次のようになります。

   XAML
   ```
   <c1:C1TreeViewItem Header="ジャンル一覧"></c1:C1TreeViewItem>
   ```

2. **<c1:C1TreeViewItem Header="ジャンル一覧"></c1:C1TreeViewItem>** タグの間に2つの子 **C1TreeViewItem** を追加して、**ジャンル一覧** ノードの下に2つの子ノードを作成します。XAML

   XAML
   ```
   <c1:C1TreeViewItem Header="文学"/>
   <c1:C1TreeViewItem Header="ノンフィクション"/>
   ```

3. もう1つの `<c1:C1TreeViewItem>` タグを追加し、2つの子ノードを含む新しい最上位ノードを作成します。XAML マークアップは次のようになります。

   XAML
   ```
   <c1:C1TreeViewItem Header="ビジネス">
     <c1:C1TreeViewItem Header="経済学"/>
     <c1:C1TreeViewItem Header="マーケティング"/>
   </c1:C1TreeViewItem>
   ```

   XAML
   ```
    <Grid>
           <c1:C1TreeView Name="Tree" >
               <c1:C1TreeViewItem Header="ジャンル一覧">
                   <c1:C1TreeViewItem Header="文学"/>
                   <c1:C1TreeViewItem Header="ノンフィクション"/>
                   <c1:C1TreeViewItem Header="ビジネス">
                       <c1:C1TreeViewItem Header="経済学"/>
                       <c1:C1TreeViewItem Header="マーケティング"/>
                   </c1:C1TreeViewItem>
               </c1:C1TreeViewItem>
           </c1:C1TreeView>
       </Grid>
   </Window>
   ```

4. プロジェクトを実行します。**Book** ノードは展開されていません。このノードは、矢印マークをクリックすることで展開できます。

コードの場合

分離コードファイルで **C1TreeView** コントロールに静的な **C1TreeView** 項目を追加するには、コードエディタで次のコードを

追加します。

WPF

Visual Basic

```
Imports C1.WPF
Class MainWindow
    Public Sub New()
        InitializeComponent()
        InitializeTreeView()
    End Sub
    Private Sub InitializeTreeView()
        ' 設計時に追加された項目を削除します
        Tree.Items.Clear()
        Dim booklist As New C1TreeViewItem()
        booklist.Header = "ジャンル一覧"
        Tree.Items.Add(booklist)
        ' 子項目を追加します
        Dim language As New C1TreeViewItem()
        language.Header = "文学"
        booklist.Items.Add(language)
        ' 子項目を追加します
        Dim security As New C1TreeViewItem()
        security.Header = "ノンフィクション"
        booklist.Items.Add(security)
        ' 子項目を追加します
        Dim classic As New C1TreeViewItem()
        classic.Header = "ビジネス"
        booklist.Items.Add(classic)
        ' 子項目を追加します
        Dim subclassic As New C1TreeViewItem()
        subclassic.Header = "経済学"
        classic.Items.Add(subclassic)
        Dim subclassic2 As New C1TreeViewItem()
        subclassic2.Header = "マーケティング"
        classic.Items.Add(subclassic2)
    End Sub
End Class
```

C#

```
using C1.WPF;
public MainWindow()
{
    InitializeComponent();
    InitializeTreeView();
}
void InitializeTreeView()
{
    // 設計時に追加された項目を削除します
    Tree.Items.Clear();
    C1TreeViewItem booklist = new C1TreeViewItem();
```

```
    booklist.Header = "ジャンル一覧";
    Tree.Items.Add(booklist);
   // 子項目を追加します
    C1TreeViewItem language = new C1TreeViewItem();
    language.Header = "文学";
    booklist.Items.Add( language );
    // 子項目を追加します
    C1TreeViewItem security = new C1TreeViewItem();
    security.Header = "ノンフィクション";
    booklist.Items.Add(security);
    // 子項目を追加します
    C1TreeViewItem classic = new C1TreeViewItem();
    classic.Header = "ビジネス";
    booklist.Items.Add(classic);
    // 子項目を追加します
    C1TreeViewItem subclassic = new C1TreeViewItem();
    subclassic.Header = "経済学";
    classic.Items.Add(subclassic);
    C1TreeViewItem subclassic2 = new C1TreeViewItem();
    subclassic2.Header = "マーケティング";
    classic.Items.Add(subclassic2);
```

Silverlight

VisualBasic

```
Imports C1.Silverlight
Class MainPage
Public Sub New()
    InitializeComponent()
    InitializeTreeView()
End Sub
Private Sub InitializeTreeView()
    ' 設計時に追加された項目を削除します
    'Tree.Items.Clear();
    Dim booklist As New C1TreeViewItem()
    booklist.Header = "ジャンル一覧"
    Tree.Items.Add(booklist)
    ' 子項目を追加します
    Dim language As New C1TreeViewItem()
    language.Header = "文学"
    booklist.Items.Add(language)
    ' 子項目を追加します
    Dim security As New C1TreeViewItem()
    security.Header = "ノンフィクション"
    booklist.Items.Add(security)
    ' 子項目を追加します
    Dim classic As New C1TreeViewItem()
    classic.Header = "ビジネス"
    booklist.Items.Add(classic)
    ' 子項目を追加します
    Dim subclassic As New C1TreeViewItem()
```

```
    subclassic.Header = "経済学"
    classic.Items.Add(subclassic)
    Dim subclassic2 As New C1TreeViewItem()
    subclassic2.Header = "マーケティング"
    classic.Items.Add(subclassic2)
End Sub.
```

C#

```
using C1.Silverlight;
public MainPage()
        {
            InitializeComponent();
            InitializeTreeView();
        }
       void InitializeTreeView()
       {
            // 設計時に追加された項目を削除します
           //Tree.Items.Clear();
            C1TreeViewItem booklist = new C1TreeViewItem();
            booklist.Header = "ジャンルー覧";
           Tree.Items.Add(booklist);
           // 子項目を追加します
          C1TreeViewItem language = new C1TreeViewItem();
          language.Header = "文学";
          booklist.Items.Add( language );
         // 子項目を追加します
         C1TreeViewItem security = new C1TreeViewItem();
         security.Header = "ノンフィクション";
         booklist.Items.Add(security);
        // 子項目を追加します
        C1TreeViewItem classic = new C1TreeViewItem();
        classic.Header = "ビジネス";
        booklist.Items.Add(classic);
       // 子項目を追加します
       C1TreeViewItem subclassic = new C1TreeViewItem();
       subclassic.Header = "経済学";
       classic.Items.Add(subclassic);
       C1TreeViewItem subclassic2 = new C1TreeViewItem();
       subclassic2.Header = "マーケティング";
       classic.Items.Add(subclassic2);
       }
```

手順 3：外観と動作のカスタマイズ

前の手順では、Visual Studio で、XAML を使用して **C1TreeViewItem** を作成しました。この手順では、Visual Studio で、XAML コードを使用して **C1TreeView** コントロールの外観と動作をカスタマイズします。

1. `<c1:C1TreeView>` タグ内にカーソルを置きます。

   `<c1:C1TreeView>` タグ内に **SelectionMode="Extended"** を追加します。これにより、[Shift]キーまたは[Ctrl]キーを押しながら複数のツリー項目を選択できる最上位ノードが作成されます。XAML マークアップは次のようになります。

XAML

```
<c1:C1TreeView x:Name="Tree" SelectionMode="Extended">
```

2. 最初の <c1:C1TreeViewItem> タグ内にカーソルを置きます。

   <c1:C1TreeViewItem> タグ内に、**IsExpanded="True"** および **IsSelected="True"** を追加します。これにより、実行時に選択および展開されて表示される最上位ノードが作成されます。XAML マークアップは次のようになります。

XAML

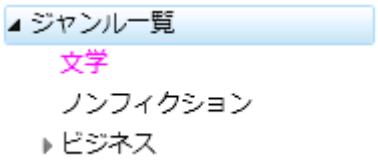
```
<c1:C1TreeViewItem Header="ジャンル一覧" IsExpanded="True" IsSelected="True">
```

3. **<c1:C1TreeViewItem Header="文学">** タグに移動します。**<c1:C1TreeViewItem Header="文学">** 内に、**Foreground="Fuchsia"** を追加します。これにより、「文学」ツリー項目の背景は明るいピンク色になり、テキストは赤紫色になります。XAML マークアップは次のようになります。

XAML

```
  <c1:C1TreeViewItem Header="文学" Foreground="Fuchsia" />
```

4. [**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。**C1TreeView** の動作と外観が次のように変更されていることを確認します。



- **C1TreeView** が展開されて表示される。
- 最初の **C1TreeViewItem** が選択されて表示される。
- 2番目の **C1TreeViewItem** のテキストが赤紫色に表示される。

おめでとうございます!

TreeView 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、**TreeView for WPF/Silverlight** アプリケーションを作成してカスタマイズし、静的な **C1TreeViewItem** 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。

ツリービュー要素

C1TreeView クラスは、次の2つの要素から成る1つの **StackPanel** です。

- 実際のノードを表すヘッダー。子を展開/折りたたむためのボタンを1つ持ちます。
- 別の StackPanel から成る本体。他のノードを格納します。

C1TreeView コントロールに追加する **C1TreeViewItem** は、XAML または分離コードで静的な項目として定義できます。または、ページまたはユーザーコントロール上に次のいずれかの方法を使用して定義することもできます。

- XAML 構文を使用するか、分離コードファイルを使用したプログラムによる静的な作成。
- コンストラクタを使用して **C1TreeViewItem** クラスの新しいインスタンスを作成する動的な作成。
- **C1TreeView** を SiteMapDataSource、XMLDataSource、または AccessDataSource に連結することによるデータソースの作成。

静的なツリービューの作成

ツリー内の各ノードは、ツリーノードのテキストプロパティと値プロパティで定義される名前/値のペアで表されます。ノードのテキストはレンダリングされますが、ノードの値はレンダリングされず、通常はポストバックイベントを処理するための追加データとして使用されます

静的なメニューは、ツリービュー構造を作成するための最も簡単な方法です。

XAML 構文を使用して静的な **C1TreeViewItem** を表示するには、最初に、**C1TreeView** コントロールの開始タグと終了タグの間に <c1:C1TreeViewItem> の開始タグと終了タグをネストします。次に、<c1:C1TreeViewItem> の開始タグと終了タグの間に <c1:C1TreeViewItem> 要素をネストして、ツリービュー構造を作成します。各 <c1:C1TreeViewItem> 要素は、コントロール内の1つのノードを表し、**C1TreeViewItem** オブジェクトにマップされます。

宣言構文を使用すると、**C1TreeViewItem** をページ内でインライン定義できます。

次に例を示します。

XAML

```
Grid x:Name="LayoutRoot">
    <c1:C1TreeView x:Name="Tree">
         <c1:C1TreeViewItem Header="ジャンル一覧" IsExpanded="True" IsSelected="True">
                <c1:C1TreeViewItem Header="文学"/>
                <c1:C1TreeViewItem Header="ノンフィクション"/>
                <c1:C1TreeViewItem Header="ビジネス">
                    <c1:C1TreeViewItem Header="経済学"/>
                    <c1:C1TreeViewItem Header="マーケティング"/>
                </c1:C1TreeViewItem>
         </c1:C1TreeViewItem>
     </c1:C1TreeView>
</Grid>
```

動的なツリービューの作成

動的なツリービューは、サーバー側またはクライアント側で作成できます。サーバー側で動的なツリービューを作成する場合は、**C1TreeView** クラスの新しいインスタンスを動的に作成するためのコンストラクタを使用します。次に例を示します。

Visual Basic

```
Public Sub New()
            InitializeComponent()
            InitializeTreeView()
        End Sub
        Private Sub InitializeTreeView()

            ' 設計時に追加された項目を削除します

            Tree.Items.Clear()

            Dim booklist As New C1TreeViewItem()
            booklist.Header = "ジャンル一覧"
            'booklist.Foreground = new SolidColorBrush( Colors.Brown);
            Tree.Items.Add(booklist)

            ' 子項目を追加します
            Dim language As New C1TreeViewItem()
            language.Header = "文学"
            booklist.Items.Add(language)
```

```
            ' 子項目を追加します
            Dim security As New C1TreeViewItem()
            security.Header = "ノンフィクション"
           booklist.Items.Add(security)

            ' 子項目を追加します
            Dim classic As New C1TreeViewItem()
            classic.Header = "ビジネス"
            booklist.Items.Add(classic)
            'チェックボックスを追加します
            classic.Header = New CheckBox()

            ' 子項目を追加します
            Dim subclassic As New C1TreeViewItem()
            subclassic.Header = "経済学"
            classic.Items.Add(subclassic)
            Dim subclassic2 As New C1TreeViewItem()
            subclassic2.Header = "マーケティング"
            classic.Items.Add(subclassic2)
        End Sub
```

C#

```
public MainWindow()
        {
            InitializeComponent();
            InitializeTreeView();

        }
       void InitializeTreeView()
       {
            // 設計時に追加された項目を削除します
            Tree.Items.Clear();

            C1TreeViewItem booklist = new C1TreeViewItem();
            booklist.Header = "ジャンル一覧";
           //booklist.Foreground = new SolidColorBrush( Colors.Brown);
           Tree.Items.Add(booklist);

           // 子項目を追加します
          C1TreeViewItem language = new C1TreeViewItem();
          language.Header = "文学";
          booklist.Items.Add( language );
         // 子項目を追加します
         C1TreeViewItem security = new C1TreeViewItem();
         security.Header = "ノンフィクション";
         booklist.Items.Add(security);
        // 子項目を追加します
        C1TreeViewItem classic = new C1TreeViewItem();
        classic.Header = "ビジネス";
        booklist.Items.Add(classic);
        //チェックボックスを追加します
        classic.Header = new CheckBox();
```

```
        // 子項目を追加します
        C1TreeViewItem subclassic = new C1TreeViewItem();
        subclassic.Header = "経済学";
        classic.Items.Add(subclassic);
        C1TreeViewItem subclassic2 = new C1TreeViewItem();
        subclassic2.Header = "マーケティング";
        classic.Items.Add(subclassic2);
}
```

データソースを活用したツリービュー項目の作成

XMLDataSource、SiteMapDataSource などの階層化データソースコントロールから TreeView 項目を作成できます。これにより、コードを編集しなくてもツリービュー項目を更新できるようになります。

C1TreeView の ItemsSource として複数レベルのデータを使用する場合は、それらの項目に対して1つの **C1HierarchicalDataTemplate** を指定する必要があります。

このテンプレートは、**C1TreeView** に次のレベルのデータがある場所を通知します。これは、**C1HierarchicalDataTemplate** の "ItemsSource" プロパティによって行われます。

あるノードに画像を追加するには、その最初の子(ヘッダー)を取得し、それを StackPanel にキャストし、画像要素を任意の場所に挿入します。次に例を示します。

Code

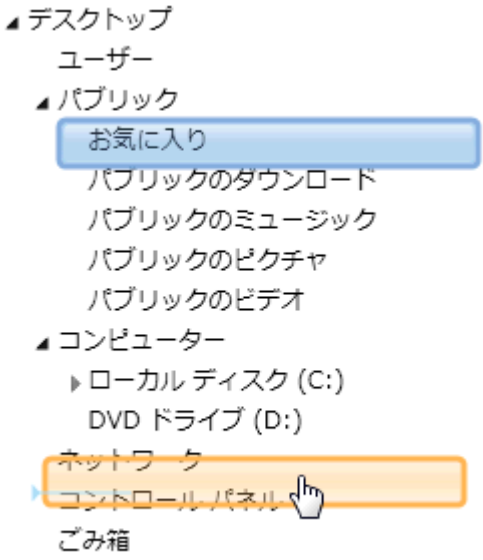
```
StackPanel nodeHeader = TreeNode.Children[0] as StackPanel;
nodeHeader.Children.Insert(0, myImage);
```

ツリービューの機能

ノードのドラッグアンドドロップ

AllowDragDrop プロパティが True に設定されている場合は、**C1TreeViewItem** をノード、ノードの間、または1つのツリーから別のツリーにドラッグアンドドロップすることができます。

次の図では、**C1TreeViewItem** を1つの **C1TreeView** から別の **C1TreeView** にドラッグしています。**DragDropArrowMarker** プロパティまたは **DragDropLineMarker** プロパティのいずれかを適用すると、矢印または垂直線を使用して、**C1TreeViewItem** のドロップ先を視覚的に示すことができます。



AllowDragDrop プロパティを **True** に設定します。

Visual Basic

```
C1TreeView.AllowDragDrop = True
```

C#

```
C1TreeView.AllowDragDrop = true;
```

DragDropArrowMarker プロパティまたは **DragDropLineMarker** プロパティを設定してドラッグアンドドロップのインジケータを視覚的に示す設定を有効にできます。

ロードオンデマンド

アプリケーションの起動時に各ノードをすべて生成するのではなく、ユーザーがノードを展開すると、オンデマンドでノードにデータが挿入される「遅延ロード」という技術を使用することができます。これにより、アプリケーションの読み込み速度が向上し、リソースを効率よく使用できます。

この実装は、ユーザーがノードを展開しようとしたときに、ノードにデータを挿入できるように、**Expanding** イベントのイベントハンドラを登録します。**Tag** プロパティには、ノードにデータを挿入するために必要な情報を保存します。最後に、ユーザーがこのノードを展開して、ノードにデータを挿入する **Expanding** イベントをトリガできるように、ダミーの子ノードを追加します。

> **メモ**: **Tag** プロパティを使用する代わりに、**C1TreeViewItem** からカスタムクラスを継承し、そのクラスにすべての遅延ロードロジックを組み込むこともできるはずです。その方が洗練された方法ですが、あいにく WPF ではテンプレートの継承がサポートされていません。テンプレートを持つクラス（**Button**、**C1TreeViewItem** など）からクラスを継承しても、テンプレートは継承されないため、テンプレートを各自で提供する必要があります。そうしないと、その派生クラスは単に空のコントロールになります。

ノードの遅延ロードを実装するには、次のコードを使用します。

C#

```
public Page()
{
    InitializeComponent();
    // ここは変更しません
```

```
    // ...
    // C1TreeView を初期化します
    InitializeTreeView();
}
void InitializeTreeView()
{
  // 設計時に追加された項目を削除します
  _tv.Items.Clear();

   // アセンブリ内のすべての型をスキャンします
  foreach (Type t in _tv.GetType().Assembly.GetTypes())
  {
    if (t.IsPublic && !t.IsSpecialName && !t.IsAbstract)
    {
      // この型のノードを追加します
      C1TreeViewItem node = new C1TreeViewItem();
      node.Header = t.Name;
      node.FontWeight = FontWeights.Bold;
      _tv.Items.Add(node);
      // プロパティ、イベント、およびメソッドのサブノードを追加します
      node.Items.Add(CreateMemberNode("Properties", t, MemberTypes.Property));
      node.Items.Add(CreateMemberNode("Events", t, MemberTypes.Event));
      node.Items.Add(CreateMemberNode("Methods", t, MemberTypes.Method));
    }
  }
}
C1TreeViewItem CreateMemberNode(string header, MemberTypes memberTypes)
{
  // ノードを作成します
  C1TreeViewItem node = new C1TreeViewItem();
  node.Header = header;
  node.Foreground = new SolidColorBrush(Colors.DarkGray);
   // ノードを展開する前にノードにデータを挿入するためのイベントハンドラを登録します
  node.Expanding += node_Expanding;
   // ノードにデータを挿入するために必要な情報を保存します
  node.Tag = memberTypes;
   // このノードを展開できるように、ダミーノードを追加します
  node.Items.Add(new C1TreeViewItem());
  node.IsExpanded = false;
   // 終了します
  return node;
}
//ノードにデータを挿入します
void node_Expanding(object sender, RoutedEventArgs e)
{
  // イベントが発生したノードを取得します
  C1TreeViewItem node = sender as C1TreeViewItem;
  // イベントハンドラの登録を解除します（ノードにデータを挿入したら、このハンドラは不要になります）
  node.Expanding -= node_Expanding;
   // ダミーノードを削除します
  node.Items.Clear();
   // ノードにデータを挿入します
  Type type = (Type)node.Parent.Tag;
```

```
  MemberTypes memberTypes = (MemberTypes)node.Tag;
   BindingFlags bf = BindingFlags.Public | BindingFlags.Instance;
  foreach (MemberInfo mi in type.GetMembers(bf))
  {
    if (mi.MemberType == memberTypes)
    {
      if (!mi.Name.StartsWith("get_") && !mi.Name.StartsWith("set_"))
      {
        C1TreeViewItem item = new C1TreeViewItem();
        item.Header = mi.Name;
        item.FontSize = 12;
        node.Items.Add(item);
      }
    }
  }
}
```

ノードの選択

実行時にノードをクリックすると、そのノードは自動的に選択中としてマークされます。カスタム機能を提供するために、ノードをクリックすると、**SelectionChanged** イベントが発生します。ノードがクリックされなくても選択中としてマークするには、**IsSelected** プロパティを有効にします。

ユーザーが新しい項目を選択すると、**C1TreeView** は **SelectionChanged** イベントを発生します。そこで、**SelectedItem** プロパティを使用すると、選択中の項目を取得できます。

それには、いくつかの方法があります。

Visual Basic

```
' ノードを作成し、その Tag プロパティにデータを割り当てます
  Dim item As New C1TreeViewItem()
  item.Header = "Beverages"
  item.Tag = beveragesID
```

C#

```
// ノードを作成し、その Tag プロパティにデータを割り当てます
  C1TreeViewItem item = new C1TreeViewItem();
  item.Header = "Beverages";
  item.Tag = beveragesID;
```

後で、この情報を適当に使用できます:

Visual Basic

```
Dim item As C1TreeViewItem = _tv.SelectedItem
    ' 飲料 ノードを処理します
If TypeOf item.Tag Is Integer AndAlso CInt(item.Tag) = beveragesID Then
End If
```

C#

```
C1TreeViewItem item = _tv.SelectedItem;
if (item.Tag is int && (int)item.Tag == beveragesID)
```

```
{
    // 飲料 ノードを処理します
}
```

XAML

```
<c1:C1TreeView Name="C1TreeView1" Height="300" Width="200" >
    <c1:C1TreeViewItem IsExpanded="True" Margin="10">
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="Desktop" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
    <c1:C1TreeViewItem>
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="User" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
    </c1:C1TreeViewItem>
</c1:C1TreeView>
```

選択された項目を取得する

Header プロパティは、**C1TreeViewItem** に含まれる値を返します。次のコードを使用して、その文字列値を取得できます。

Visual Basic

```
Dim item As C1TreeViewItem = _tree.SelectedItem
_textBlock.Text = item.Header.ToString()
```

C#

```
C1TreeViewItem item = _tree.SelectedItem;
_textBlock.Text = item.Header.ToString();
```

Windows

次のような主要機能を利用して、**Windows for WPF/Silverlight** を最大限に活用してください。

- **モーダルおよびモードレスダイアログウィンドウ**

  ダイアログウィンドウがモーダルおよびモードレスダイアログウィンドウとして表示されているときに、ユーザーが他のウィンドウを操作できるかどうかを決定します。

- **ウィンドウオブジェクトを定義するための個別の XAML ファイル**

  C1Window オブジェクトは、ページ上のどの要素の子でもありません。このため、これらのオブジェクトは、**C1Window** オブジェクトをルート要素とする個別の XAML ファイルで定義されます。

- **サイズ変更可能なウィンドウ**

  ウィンドウのサイズを変更できるかどうかを簡単に指定できます。ウィンドウが小さくてすべてのコンテンツを表示できな

くなると、自動的にスクロールバーが追加されます。

- **ウィンドウの状態**

  Windows for WPF/Silverlightは、最小化したり、元のサイズに戻すことができます。開発者がウィンドウの現在の状態を設定できます。ウィンドウを最大化できないように設定することもできます。

- **ウィンドウ要素**

  XAML ファイルを編集し、任意の要素を追加することができます。**Windows for WPF/Silverlight** を使用すると、**C1Window** オブジェクトの作成と保守がたいへん簡単になります。

- **XBAP サポート**

  XBAP は、Web 向けに構築された WPF アプリケーションです。**Windows for WPF/Silverlight** は、モーダルとモードレスの両方のウィンドウ向けの XBAP シナリオでサポートされます。

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

クイックスタート

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 Windows for WPF/Silverlight を使用する WPF/Silverlight アプリケーションを作成します。この手順では、フォームに2つのボタンと1つのテキストボックスを追加します。

プロジェクトをセットアップし、C1Window 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。

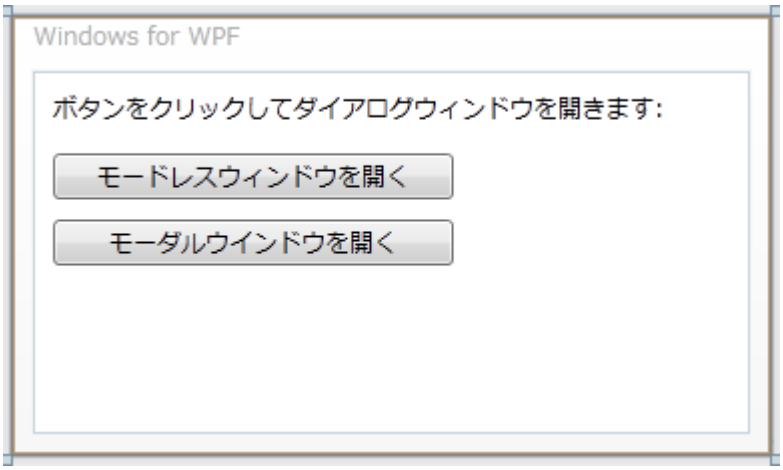
1. Visual Studio で、[**ファイル**]→[**新しいプロジェクト**]を作成します。
2. [新しいプロジェクト]ダイアログボックスの左ペインから言語を選択し、テンプレートリストから[WPF アプリケーション]または[Silverlight アプリケーション]を選択します。
3. プロジェクトの名前(たとえば、「QuickStart」)を入力し、[OK]をクリックします。[新しい WPF/Silverlight アプリケーション]ダイアログボックスが表示されます。
4. [OK]をクリックしてデフォルト設定を受け入れ、[新しい WPF アプリケーション]ダイアログボックスを閉じると、プロジェクトが作成されます。MainPage.xaml ファイルが開きます。
5. 参照の追加ダイアログボックスで、以下のアセンブリを見つけて選択し、[OK]をクリックしてプロジェクトに参照を追加します。
   - **WPF**: C1.WPF.4.dll
   - **Silverlight**: C1.Silverlight.5.dll
6. フォームを1回クリックして選択し、Visual Studio のツールボックスに移動し、1つの TextBlock コントロールと2つの Button コントロールを追加します(ツールボックスの項目をダブルクリックすることでフォームに追加します)。
7. フォームのサイズを変更し、フォームにあるコントロールのサイズを変更します。
   - 各コントロールを順に選択し、[プロパティ]ウィンドウでそれぞれについて以下のプロパティを設定します。
   - **TextBlock1** の **Text** プロパティを「ボタンをクリックしてダイアログウィンドウを開きます:」。
   - **Button1** の **Content** プロパティを「モードレスウィンドウを開く」に設定します。
   - **Button2** の **Content** プロパティを「モーダルウインドウを開く」に設定します。

XAML マークアップは次のようになります。

XAML

```
<Grid>
        <TextBlock Height="23" Margin="10,10,31,0" Name="TextBlock1"
VerticalAlignment="Top" Text="Click a button to open a dialog window:" />
        <Button Height="23" Margin="10,41,148,0" Name="Button1"
VerticalAlignment="Top">Open a modeless window.</Button>
        <Button Margin="10,74,148,0" Name="Button2" Height="23"
VerticalAlignment="Top">Open a modal window.</Button>
</Grid>
```

フォームは次の図のように表示されます。



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ボタンコントロールを追加しました。この手順では、引き続き、ユーザーコントロールに C1Window コントロールを追加します。

次の手順に従います。

1. Visual Studio のソリューションエクスプローラで、プロジェクトを右クリックし、[**追加**]→[**新しい項目**]オプションを選択します。[**新しい項目の追加**]ダイアログボックスが表示されます。
2. [**新しい項目の追加**]ダイアログボックスの左側で WPF/Silverlight の項目を選択し、右側の[テンプレート]セクションで、[ユーザーコントロール(WPF)または ユーザーコントロール (Silverlight)]を選択し、新しいコントロールに **"MyWindow.xaml"** と名前を付け、[**追加**]をクリックして新しいユーザーコントロールを追加します。

   ファイルが自動的に開かない場合は、ソリューションエクスプローラで **MyWindow.xaml** ファイルをダブルクリックして開きます。
3. ツールボックスに移動し、**TextBlock** 項目をダブルクリックして、コントロールをフォームに追加します。
4. **TextBlock** の **Text** プロパティを「**Hello World!**」に設定します。ユーザーコントロールの XAML マークアップは次のようになります。

XAML

```
<Grid>
   <TextBlock Height="21" HorizontalAlignment="Left" Margin="10,10,0,0"
Name="TextBlock1" VerticalAlignment="Top" Width="120" Text="Hello World!" />
```

```
</Grid>
```

必要に応じて、その他のコントロールをフォームに追加できます。追加する項目は、**C1Window** コントロールの本体に挿入されます。

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ボタンを押しても、何も実行されません。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. フォームのデザインビューに戻ります。
2. フォームの **Button1** を選択し、[プロパティ]ウィンドウに移動し、稲妻の[**イベント**]アイコンをクリックしてイベントを表示し、**Click** 項目の横にある領域をクリックして **Button1_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。
3. デザインビューに戻り、**Button2** で前の手順を繰り返して **Button2_Click** 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
または
Imports C1.Silverlight
```

C#

```
using C1.WPF;
または
using C1.Silverlight;
```

5. 前に追加したイベントハンドラに次のコードを追加します。

Visual Basic

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    ShowWindow(False)
End Sub
Private Sub Button2_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    ShowWindow(True)
End Sub
```

C#

```
void button1_Click(object sender, RoutedEventArgs e)
{
  ShowWindow(false);
}
void button2_Click(object sender, RoutedEventArgs e)
{
```

```
  ShowWindow(true);
}
```

6. Button_Click イベントハンドラの下に次のコードを追加します。

Visual Basic

```
Private Sub ShowWindow(ByVal showModal As Boolean)
    Dim wnd As New C1Window()
    wnd.Header = "ヘッダー領域"
    wnd.Height = 120
    wnd.Width = 200
    wnd.Content = New MyWindow()
    wnd.CenterOnScreen()
    If showModal Then
        wnd.ShowModal()
    Else
        wnd.Show()
    End If
End Sub
```

C#

```
private void ShowWindow(bool showModal)
{
    C1Window wnd = new C1Window();
    wnd.Header = "ヘッダー領域";
    wnd.Height = 120;
    wnd.Width = 200;
    wnd.Content = new MyWindow();
    wnd.CenterOnScreen();
    if (showModal)
wnd.ShowModal();
    else
wnd.Show();
}
```

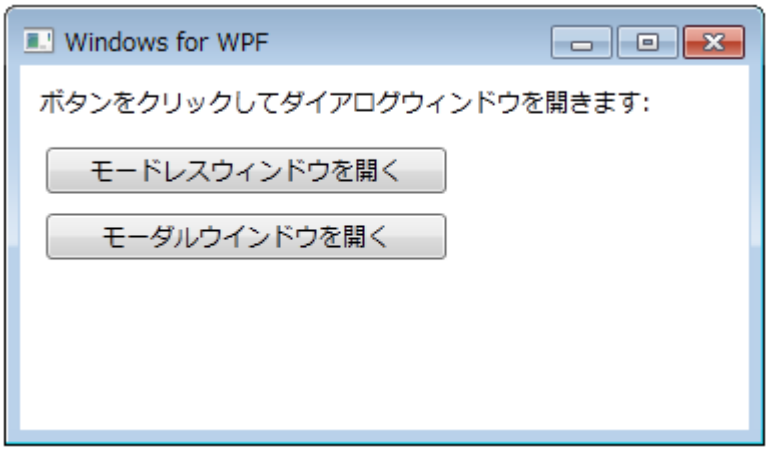
このコードは、ウィンドウのサイズを指定し、新しいウィンドウを開きます。

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。

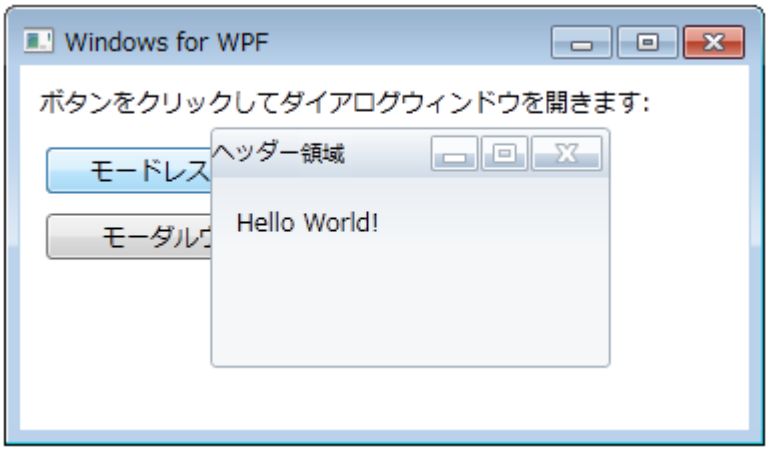
手順 4：アプリケーションの実行

これまでに WPF/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て **Windows for WPF/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、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。

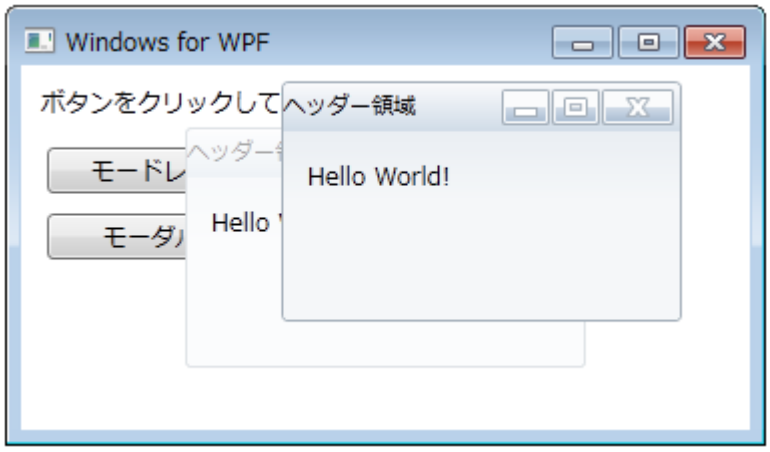


2. [**モードレスウィンドウを開く**]ボタンをクリックします。モードレスダイアログウィンドウが開きます。

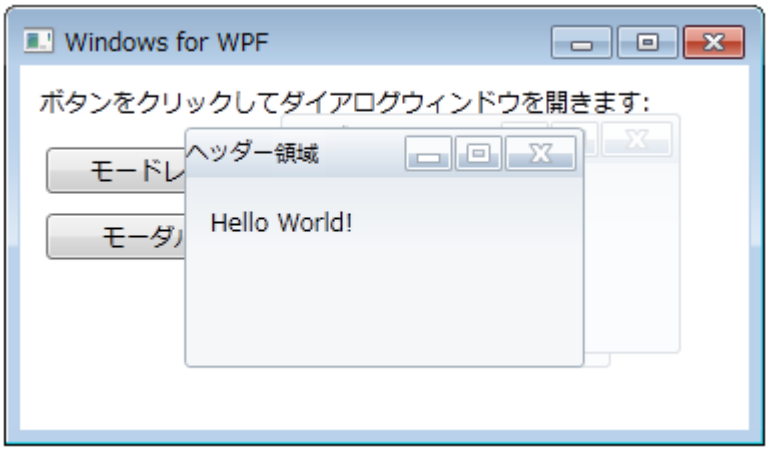


モードレスダイアログウィンドウを使用すると、このウィンドウを開いたまま、ページ内の他の項目を操作できます。

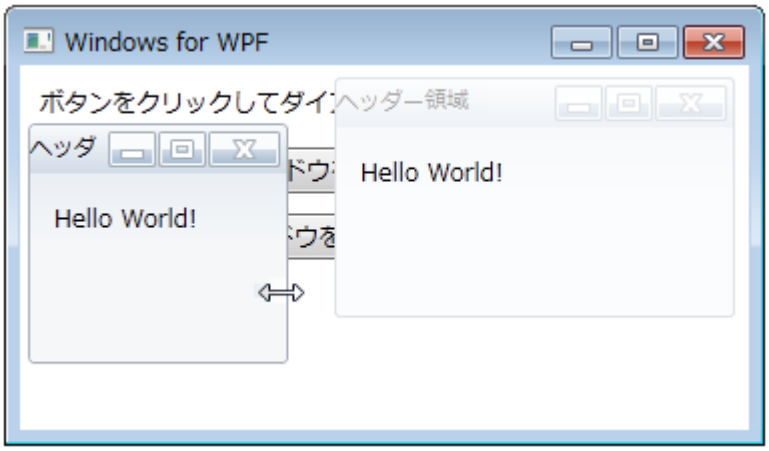
3. [モードレスウィンドウを開く]ボタンを再度クリックします。別のモードレスダイアログウィンドウが開きます。



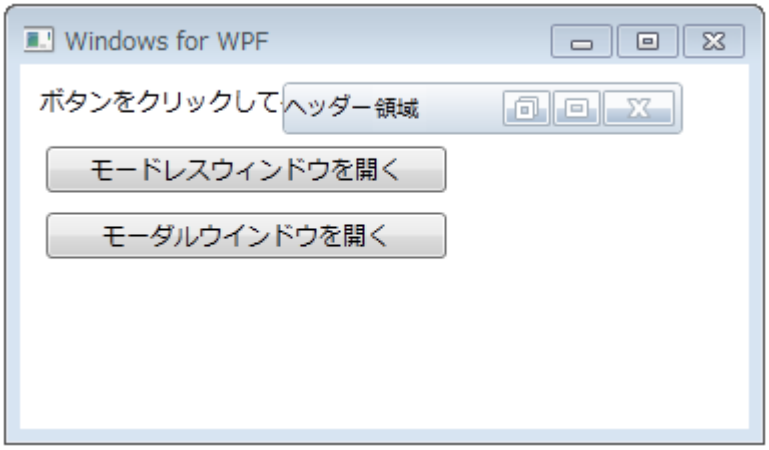
4. 最初のダイアログボックスをクリックして、そのダイアログボックスにフォーカスが移動することを確認します。



5. ダイアログウィンドウのヘッダーをクリックし、ドラッグして移動します。
6. 境界線をクリックしてドラッグし、ダイアログウィンドウのサイズを変更します.



7. [X]ボタンをクリックしてダイアログウィンドウを閉じます。
8. [_]ボタンをクリックして2つ目のダイアログウィンドウを最小化します。



9. [**モーダルウインドウを開く**]ボタンをクリックします。モーダルダイアログウィンドウが開きます。

この新しいダイアログウィンドウ以外は、ページ全体が淡色表示されることがわかります。モーダルダイアログウィンドウを使用すると、このダイアログウィンドウが開いている間は、他の要素を操作できなくなります。

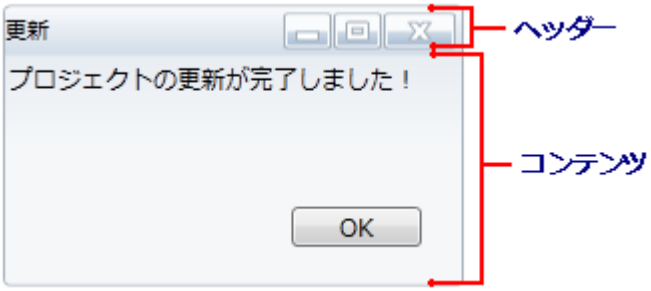
10. 最小化されているダイアログウィンドウを選択しようとしても、できないことがわかります。
11. 閉じるボタン([X])をクリックしてモーダルダイアログボックスを閉じます。ページ内の要素を操作できるようになったことを確認します。

おめでとうございます。

これで**Windows for WPF/Silverlight** クイックスタートは完了です。簡単な WP/Silverlight アプリケーションを作成し、**C1Window** コントロールなどのコントロールを追加し、**Windows for WPF/Silverlight** の実行時機能をいくつか確認しました。

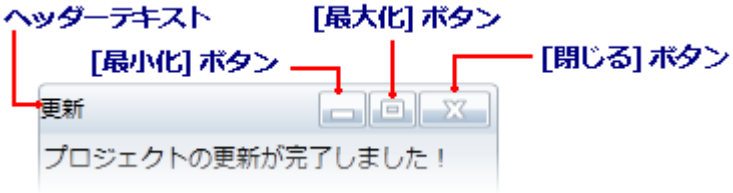
C1Window の要素

このセクションでは、C1Window 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。通常のダイアログボックスと同様に、C1Window コントロールには、ヘッダーとコンテンツ領域という2つの主要な要素があります。



ヘッダー

ヘッダー領域には、タイトルテキスト、[**最小化**]**ボタン**、[**最大化**]**ボタン**、[**閉じる**]ボタンなどの通常のキャプションバー要素が含まれています。



Header プロパティを使用して、ヘッダーのテキストを設定できます。**ShowCloseButton**、**ShowMaximizeButton**、**ShowMinimizeButton** の各プロパティを使用して、ボタンの表示を設定

できます。デフォルトでは、この3つのプロパティはすべて **True** で、これらのボタンが表示されます。**Header** プロパティを **UIElement** に設定することもできます。したがって、たとえば、キャプションバーのテキストの横にアイコンを追加することができます。

タイトルのテキストの設定

- **XAML の場合**

XAML

```
<c1:C1Window Height="110" Width="220" Content="C1Window" Header="Hello World!"/>
```

- **コードの場合**

Header 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.C1Window1.Header = "Hello World!"
```

C#

```
this.c1window1.Header = "Hello World!";
```

- **設計時**

Header プロパティを設定するには、次の手順に従います。

1. **C1Window** コントロールをクリックして選択します。
2. [プロパティ]タブに移動し、[ヘッダー]項目を探します。
3. Header 項目の横にあるテキストボックスをクリックし、「Hello World!」などのテキストを入力します。

これで、Header プロパティとダイアログウィンドウのキャプションバーのテキストが、選択したテキストに設定されます。

ヘッダーボタンの非表示

デフォルトでは、ウィンドウに**最小化**、**最大化**、**閉じる**の各ボタンが表示されますが、XAML、およびコードでこれをカスタマイズできます。

- **XAML の場合L**

XAML

```
<c1:C1Window Height="129" Width="220" Content="C1Window"
ShowMaximizeButton="False" ShowMinimizeButton="False"/>
```

- **コードの場合**

Visual Basic

```
Me.C1Window1.ShowMaximizeButton = False
Me.C1Window1.ShowMinimizeButton = False
```

C#

```
this.c1window1.ShowMaximizeButton = false;
this.c1window1.ShowMinimizeButton = false;
```

- **設計時**

Blend で[**最大化**] ボタンと[**最小化**] ボタンを非表示にするには、次の手順に従います。

1. **C1Window** コントロールをクリックして選択します。
2. [プロパティ]ウィンドウタブに移動します。
3. **ShowMaximizeButton** 項目を見つけ、この項目の横にあるチェックボックスをオフにします。
4. **ShowMinimizeButton** 項目を見つけ、この項目の横にあるチェックボックスをオフにします。

Windows の機能

モーダルおよびモードレスダイアログウィンドウ

ダイアログボックスは、アプリケーションでユーザーから入力を得るためによく使用されます。たとえば、ダイアログボックスを使ってユーザーに入力を促し、入力を得ると、自動的にダイアログボックスを閉じるか破棄するアプリケーションがあります。

また、ダイアログボックスを使って情報を表示したまま、ユーザーが別のウィンドウで作業できるようにしているアプリケーションもあります。たとえば、Microsoft Word でスペルをチェックする際は、ダイアログボックスが開いたままになるため、ドキュメントのテキストを読んで編集しながら、スペルチェッカによって次のスペルミスの単語を検索できます。アプリケーションがさまざまな方法でダイアログボックスを使用できるように、**C1Window** は、**モーダル**と**モードレス**の2つのタイプのダイアログウィンドウをサポートします。

モーダルウィンドウ

モーダルダイアログウィンドウは、先に閉じないと、現在のアプリケーションの作業を続けることができない子ウィンドウです。通常、モーダルダイアログウィンドウを閉じるまでは、そのウィンドウがシステム全体または表示元のアプリケーションを制御します。たとえば、モーダルダイアログウィンドウを使用すれば、ユーザーからログイン情報を取得するまで、アプリケーションの操作を続行できないようにすることができます。モーダルウィンドウは、重要な情報を表示したり、ユーザーに操作を要求するときに便利です。

ShowModal メソッドを使用すると、**C1Window** コントロールをモーダルダイアログボックスとして表示できます。

Visual Basic

```
'The Show method is seen here in an If...Then...Else statement.
If showModal Then
        wnd.ShowModal()
    Else
        wnd.Show()
    End If
```

C#

```
//The Show method is seen here in an if...else statement.
if (showModal)
      wnd.ShowModal();
    else
      wnd.Show();
```

モーダル背景色の設定

ModalBackground プロパティを設定すると、この背景色をカスタマイズできます。

1. Button_Click イベントのコードは、次のようになります。

Visual Basic

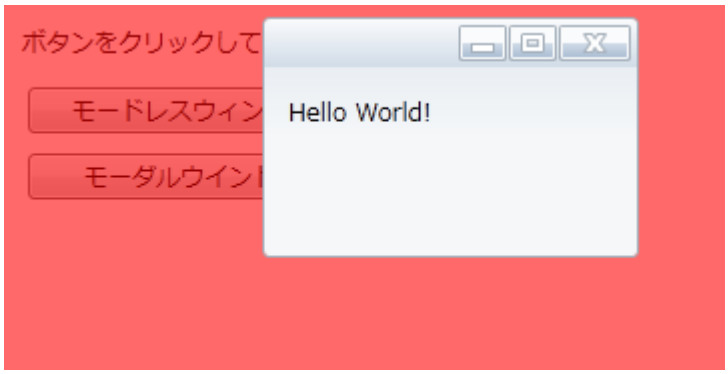
```
Private Sub ShowDialog(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim window = New C1Window()
    window.Content = New MyWindow()
    window.CenterOnScreen()
    Dim bgcol As New SolidColorBrush()
    bgcol.Color = Color.FromArgb(150, 255, 0, 0)
    window.ModalBackground = bgcol
```

```
    window.ShowModal()
End Sub
```

C#

```
void ShowDialog(object sender, RoutedEventArgs e)
{
    var window = new C1Window();
    window.Content = new MyWindow();
    window.CenterOnScreen();
    SolidColorBrush bgcol = new SolidColorBrush();
    bgcol.Color = Color.FromArgb(150, 255, 0, 0);
    window.ModalBackground = bgcol;
    window.ShowModal();
}
```

3. アプリケーションを実行し、ダイアログウィンドウを**モーダルダイアログボックスとして開きます**。ウィンドウの背景色が赤色などの選択した色になっていることを確認します。



モードレスウィンドウ

モードレスダイアログウィンドウは、このダイアログウィンドウが表示されていても、ユーザーが他のウィンドウを操作できるようにします。要求する情報が作業の続行に必須でない場合は、このタイプのダイアログウィンドウを使用します。モードレスダイアログウィンドウは、入力フォーカスを持ち続けないため、ユーザーは同時に2つのアプリケーションで作業できます。

メニューやヘルプシステムでダイアログウィンドウとアプリケーションウィンドウを同時に使用できるようにする場合に、モードレスダイアログウィンドウはよく使用されます。たとえば、ツールバーをアプリケーションからデタッチし、ツールバー内の項目を選択して、その機能をデタッチされたアプリケーションに適用できる場合、そのツールバーはモードレスダイアログウィンドウです。

Show メソッドを使用すると、**C1Window** コントロールをモードレスダイアログボックスとして表示できます。

Visual Basic

```
'The Show method is seen here in an If...Then...Else statement.
If showModal Then
        wnd.ShowModal()
    Else
        wnd.Show()
    End If
```

C#

```
//The Show method is seen here in an if...else statement.
if (showModal)
```

```
    wnd.ShowModal();
  else
    wnd.Show();
```