

Binding Expressions for WPF/Silverlight

2018.02.20 更新

グレースィティ株式会社


目次

製品の概要	2
ComponentOne Studio for WPF/Silverlight のヘルプ	2
主な特長	2
C1Binding Expression の操作	3-4
C1Binding クラス	4-5
C1Binding Expression 構文の使用	5
C1Binding の使用例	5-6
C1Binding の制限	6
C1CalcEngine クラス	6
C1CalcEngine の使用例	6-7
C1Binding Expression 構文の要素	8-10

製品の概要

Binding Expressions for WPF/ Silverlight を使用すれば、連結コンバータを記述する必要はなくなります。

インライン式構文を活用して、開発時間を短縮し、簡潔な XAML を記述できます。複雑な分離コードコンバータを使用しなくても、連結ステートメント内で、文字列を連結したり、式を計算することができます。また、直接 if/else ロジックを適用することもできます。

 **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。

ComponentOne Studio for WPF/Silverlight のヘルプ

はじめに

ComponentOne Studio 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 Studio for WPF/Silverlight ユーザーガイド](#)」を参照してください。

主な特長

Binding Expressions for WPF/Silverlight は、次のユニークな主要機能を備えています。

- **演算子および関数**
のサポート連結ステートメント内で単純な演算 (+、-、*、/) および関数を直接実行できます。Sum、Average、Concatenate などの関数がサポートされています。演算子および関数のリストについては、「[C1Binding Expression 構文の要素](#)」を参照してください。
- **XAML を崩さずに引用符を使用**
XAML を崩すことなく連結式で引用符を使用できます。C1Binding Expression は、" とパイプ文字 (|) の2つの形式のインライン引用符をサポートしています。
- **論理式**
連結ステートメント内で、単純な if/else ロジックを直接適用できます。C1Binding は、if(condition, true value、false value) という構文の 'IF' 関数をサポートしています。
- **簡潔かつ表現力豊かな XAML**
[C1Binding Expression](#) を使用することで、簡潔かつ表現力豊かな XAML を記述できます。XAML は、100% 自己完結型です。ソリューション内から外部のコンバータを参照する必要はありません。
- **計算エンジン**
C1.WPF.Binding, アセンブリには、C1Binding クラスに加えて、式を評価するための **C1CalcEngine** クラスが用意されています。**C1CalcEngine** クラスは、計算される入力ボックス、スプレッドシートなどのさまざまなアプリケーションを実装するために、**C1Binding** とは関係なく使用できます。詳細およびいくつかの使用例については、「[C1CalcEngine クラス](#)」を参照してください。
- **Silverlight5 構文の要素**
C1.Silverlight.Binding クラスには Silverlight5 が必要です。これらのクラスは MarkupExtension に依存しており、Silverlight の以前のバージョンにはこれがないためです。

C1Binding Expression の操作

WPF および Silverlight では、データ値を UI のプロパティに接続するために Binding オブジェクトが使用されます。たとえば、**TextBlock** 要素に顧客の名前を表示するには、次の XAML を使用できます。

XAML

```
<TextBlock Text="{Binding FirstName}" />
```

この例で、**TextBlock** の **Text** プロパティは、**TextBox** 要素またはそのいずれかの祖先の **DataContext** プロパティに現在割り当てられているデータオブジェクトの **FirstName** プロパティの内容と自動的に同期されます。

Binding クラスは便利ですが、柔軟性は高くありません。たとえば、前述の **TextBlock** を顧客のフルネームに連結する場合、または顧客アカウントがマイナスの場合にテキストを赤色にする場合は、**Converter** が必要になります。

Binding オブジェクトの **Converter** パラメータを使用することにより、連結に任意のロジックを追加して、ソースとターゲットの間を転送される値を変換できます。

たとえば、前述の例を使用して、**TextBlock** に顧客のフルネームを表示し、顧客のアカウントがマイナスの場合にテキストを赤色で表示することにします。それには2つのコンバータが必要で、それらを次のように実装します。

XAML

```
/// <summary>
/// Foreground プロパティのコンバータ:顧客の金額がマイナスの場合は赤色のブラシを返し、
/// そうでない場合は黒色のブラシを返します。
/// </summary>
public class ForegroundConverter : System.Windows.Data.IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var c = value as Customer;
        return c.Amount < 0
            ? new SolidColorBrush(Colors.Red)
            : new SolidColorBrush(Colors.Black);
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
/// <summary>
/// 顧客のフルネームのコンバータ:顧客のフルネームを含む
/// 文字列を返します。
/// </summary>
public class FullNameConverter : System.Windows.Data.IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        var c = value as Customer;
        return string.Format("{0} {1}", c.FirstName, c.LastName);
    }
}
```

```
public object ConvertBack(object value, Type targetType, object parameter,
    CultureInfo culture)
{
    throw new NotImplementedException();
}
}
```

コンバータを定義したら、次のように XAML で使用できます。

XAML

```
<Grid>
  <Grid.Resources>
    <local:FullNameConverter x:Key="_cvtFullName" />
    <local:ForegroundConverter x:Key="_cvtForeground" />
  </Grid.Resources>

  <TextBlock
    Text="{Binding Converter={StaticResource _cvtFullName}}"
    Foreground="{Binding Converter={StaticResource _cvtForeground}}" />
</Grid>
```

このソリューションは正しく動作しますが、いくつか欠点があります。

- XAML を見ても、どのように出力されるのかわかりません。たとえば、この色コンバータは、どのようなロジックを使用してどのような色を返すのかわかりません。それを確認するには、コンバータの実装をしてみるしかありません。
- **TextBlock** 要素の **Opacity**、**Visibility**、**FontWeight** などのプロパティを設定するには、さらにいくつかのコンバータクラスを作成し、それらをいつも XAML とセットにしておく必要があります。
- コンバータクラスのセマンティクスは、あるオブジェクトを別のオブジェクトに変換することだけを規定しています。これにより、柔軟性は高くなりますが、コンバータは多少脆弱になります。

C1Binding クラス

これらの問題を対処するために、**C1Binding** クラスでは、表現力豊かな式を使用して連結を行うことができます。**C1Binding** を使用すると、前述の例を次のように記述できます。

XAML

```
<TextBlock
  Text="{c1:C1Binding Expression='concatenate(FirstName, | |, LastName)'}"
  Foreground="{c1:C1Binding Expression='if(Amount &lt; 0, |red|, |black|)'}" />
```

この方がずっと簡潔かつ直感的であることがわかります。リソースもコンバータもありません。連結の意図が XAML からはっきりとわかります。

XAML の作成者がフルネームを「姓、名」の順で表示したいと思ったら、XAML で直接変更できます。マイナスの金額を太字で表示させたい場合も、簡単に変更できます。

XAML

```
<TextBlock
  Text="{c1:C1Binding Expression='concatenate(FirstName, | |, LastName)'}"
  Foreground="{c1:C1Binding Expression='if(Amount &lt; 0, |red|, |black|)'}" />
  FontWeight="{c1:C1Binding Expression='if(Amount &lt; 0, |bold|, |normal|)'}" />
```

Binding Expressions for WPF/Silverlight

上の式で、縦棒は引用符を表します。**"**を使用することもできますが、縦棒の方が読みやすく入力も簡単です。

上の式内の **"Expression="** 部分は、WPF ではオプションですが、Silverlight の場合は Silverlight 5 が解放された後必要ではありません。

C1Binding Expression 構文の使用

C1Binding オブジェクトは、**C1CalcEngine** クラスを使用して、式を解析および評価します。サポートされている構文は、Microsoft Excel で使用されている構文と類似しています。

一般的な論理演算子(=、>、<、<>、>=、<=)、算術演算子(+、-、*、/、^)はすべて使用できます。また、かっこを使用して式をグループ化できます。次に例を示します。

```
"1+2*3 < (1+2)*3" returns TRUE (7 < 9).
```

また、式内では、連結ソースオブジェクトの任意のパブリックプロパティを使用できます。次に例を示します。

```
"Price * 8.5%" は、Price プロパティに 0.085 を乗算した値を返します。
```

コレクションや辞書を返すプロパティもサポートされています。たとえば、**DataContext** オブジェクトに **Children** プロパティがあるします。

```
"Children(Children.Count-1).Name" は、最後の子の名前を返します。
```

さらに、**C1CalcEngine** は、Excel で使用できる関数のサブセットをサポートします。これらの関数のリストについては、「[C1Binding Expression 構文の要素](#)」を参照してください。

C1Binding の使用例

単純な計算:

ある金額にかかる税を表示するとします。従来の連結を使用する場合、課税額を計算するにはコンバータが必要になります。**C1Binding** では、単純な式を使用して課税額を計算できます。

XAML

```
<TextBlock
  Text="{c1:C1Binding Expression='Amount * 8.5%' }"h
```

値の結合:

上の例を拡張し、課税額に加えて総額を表示するとします。それには、複数の **TextBlock** 要素を使用することもできますが、1つの **TextBlock** と、CONCATENATE 関数による **C1Binding** を使用する方が効率的です。

XAML

```
<TextBlock
  Text="{c1:C1Binding Expression=
  'CONCATENATE(Amount, | 税額: |, Amount * 8.5%)' }"
```

値の書式設定:

C1Binding および通常の **Binding** オブジェクトには、連結値の書式を制御するための **StringFormat** プロパティがあります。連結式で出力の一部を書式設定する必要がある場合は、次のように TEXT 関数を使用できます。

XAML

```
<TextBlock
```

```
Text="{cl:C1Binding Expression=
  'CONCATENATE(TEXT(Amount, |c|), | 税額: |, TEXT(Amount * 8.5%, |c|))' }"
```

条件付き書式設定:

一定の値を超える金額を太字および赤色で表示するユーザーインターフェイスを作成するとします。これは、条件付きロジックを実行する IF 関数を使用して作成できます。

XAML

```
<TextBlock
  Text="{cl:C1Binding Expression='Amount', StringFormat='c'}"
  Foreground="{cl:C1Binding Expression='if(Amount > 1000, |red|, |black|)'}" />
  FontWeight="{cl:C1Binding Expression='if(Amount > 1000, |bold|, |normal|)'}" />
```

C1Binding の制限

C1Binding オブジェクトは、一方向連結シナリオでのみ使用できます。

理由は単純です。通常の連結は単純なプロパティに基づいているため、値の評価と割り当てが可能です。一方、**C1Binding** は式に基づいているため、評価は可能ですが、値を割り当てることはできません。たとえば、"Amount" というプロパティに新しい値を割り当てることはできますが、"Credit- Debit" という式に値を割り当てることはできません。

これ以外にも、**C1Binding** オブジェクトには、通常の **Binding** オブジェクトと同じ制限があります。

- これらのオブジェクトは依存プロパティにのみ割り当てることができます。依存プロパティは、**UIElement**、**FrameworkElement** などの依存オブジェクトにのみ存在します。
- プロパティが変更されたときに自動的に連結を更新するには、連結ソースオブジェクトで **INotifyPropertyChanged** インターフェイスを実装する必要があります。
- ソースプロパティとターゲットプロパティの型に互換性がない場合は、ターゲットプロパティの側に適切なコンバータが必要です。たとえば、**Color**、**FontWeight** などの大部分の型は、文字列値から変換することができます。

C1CalcEngine クラス

C1CalcEngine クラスは、**C1Binding** オブジェクトで使用される式の解析および評価を受け持ちます。ただし、これはパブリッククラスなので、**C1Binding** とは関係なく使用することもできます。

C1CalcEngine クラスには、**Parse** と **Evaluate** の2つの主要なメソッドがあります。

1. **Parse** は、文字列を解析して **Expression** オブジェクトに変換します。
2. **Evaluate** は、文字列を解析し、結果の式を評価します。

C1CalcEngine の使用例

計算される TextBox:

計算される **TextBox** をアプリケーションに追加するとします。ユーザーが **TextBox** に式を入力できるようにします。コントロールがフォーカスを失うか、ユーザーが Enter を押すと、式が評価されます。それには、たとえば次のコードを使用します。

XAML

```
public MainWindow()
```

```
{
  InitializeComponent();
  var tb = new TextBox();
  tb.LostFocus += (s,e) => { Evaluate(s); };
  tb.KeyDown += (s, e) => { Evaluate(s); };
}
void Evaluate(object sender)
{
  var tb = sender as TextBox;
  var ce = new C1.WPF.Binding.C1CalcEngine();
  try
  {
    var value =ce.Evaluate(tb.Text);
    tb.Text = value.ToString();
  }
  catch (Exception x)
  {
    MessageBox.Show("Error in Expression: " + x.Message);
  }
}
```

グリッドからスプレッドシートへの変換:

C1CalcEngine クラスの典型的な使用方法として、計算されるセルをグリッドコントロールやデータオブジェクトに実装することがあります。この方法を使用して、グリッドコントロールにスプレッドシート機能を追加するサンプルを作成しました。このコードは長すぎるのでここには掲載ませんが、<http://demos.componentone.com/silverlight/ExcelBook/> でサンプルの動作を確認できます。

C1Binding Expression 構文の要素

C1Binding クラスは、**C1CalcEngine** クラスを使用して、式を解析および評価します。以下に説明する構文要素は、どちらのクラスにも適用されます。

大文字と小文字

すべての式で、大文字と小文字は区別されません。

演算子

式では、次の演算子を使用できます。

演算子の種類	演算子
比較	< > = <= >=
加算/減算	+ -
乗算/除算	* /
べき算	^
グループ化	() , .

これらの演算子は通常の優先順位に従います。したがって、" $1 + 2 * 3$ " は 7 になります。式は、かっこを使用してグループ化できます。たとえば、" $(1 + 2) * 3$ " は 9 になります

関数

式では、次の関数呼び出しを使用できます。

論理関数

名前	説明	構文
AND	引数のすべてが TRUE の場合に、TRUE を返します	=AND(logical1[, logical2,...])
FALSE	論理値 FALSE を返します	=FALSE
IF	実行する論理テストを指定します	=IF(logical_test, value_if_true, value_if_false)
NOT	引数のロジックを反転させます	=NOT(logical)
OR	いずれかの引数が TRUE の場合に、TRUE を返します	=OR(logical1[, logical2,...])
TRUE	論理値 TRUE を返します	=TRUE

数学関数

名前	説明	構文
ABS	数値の絶対値を返します	=ABS(number)
ACOS	数値の逆余弦を返します	=ACOS(number)
ASIN	数値の逆正弦を返します	=ASIN(number)
ATAN	数値の逆正接を返します	=ATAN(number)
ATAN2	X および Y 座標からの逆正接を返します	=ATAN2(x_num, y_num)

Binding Expressions for WPF/Silverlight

CEILING	数値を最も近い整数または最も近い基準値の倍数に切り上げます	=CEILING(number)
COS	数値の余弦を返します	=COS(number)
COSH	数値の双曲線余弦を返します	=COSH(number)
EXP	e を底とする数値のべき乗を返します	=EXP(number)
FLOOR	数値を 0 に近い方に切り捨てます	=FLOOR(number)
INT	数値をより小さな最も近い整数に丸めます	=INT(number)
LN	数値の自然対数を返します	=LN(number)
LOG	指定された数を底とする数値の対数を返します	=LOG(number[, base])
LOG10	10 を底とする数値の対数を返します	=LOG10(number)
PI	円周率を返します	=PI()
POWER	数値のべき乗を返します	=POWER(number, power)
RAND	0 ~ 1 の間の乱数を返します	=RAND()
RANDBETWEEN	指定された範囲内の整数の乱数を返します	=RANDBETWEEN(bottom, top)
SIGN	数値の符号を返します	=SIGN(number)
SIN	指定された角度の正弦を返します	=SIN(number)
SINH	数値の双曲線正弦を返します	=SINH(number)
SQRT	正の平方根を返します	=SQRT(number)
SUM	引数を合計します	=SUM(number1[, number2, ...])
TAN	数値の正接を返します	=TAN(number)
TANH	数値の双曲線正接を返します	=TANH(number)
TRUNC	数値を整数に切り捨てます	=TRUNC(number);

統計関数

名前	説明	構文
AVERAGE	引数の平均値を返します	
AVERAGEA	数値、テキスト、論理値を含む引数の平均値を返します	=AVERAGE(number1 [, number2, ...])
COUNT	引数リストに含まれる数値の個数を返します	=AVERAGEA(number1 [, number2, ...])
COUNTA	引数リストに含まれる値の個数を返します	=COUNT(number1 [, number2, ...])
COUNTBLANK	範囲内に含まれる空白セルの個数を返します	=COUNTA(number1 [, number2, ...])
COUNTIF	範囲内に含まれる特定の基準を満たすセルの個数を返します	=COUNTIF(range, criteria)
MAX	引数リスト内の最大値を返します	=MAX(number1 [, number2, ...])
MAXA	数値、テキスト、論理値を含む引数リスト内の最大値を返します	=MAXA(number1 [, number2, ...])
MIN	引数リスト内の最小値を返します	=MIN(number1 [, number2, ...])
MINA	数値、テキスト、論理値を含む引数リスト内の最小値を返します	=MINA(number1 [, number2, ...])
STDEV	標本標準偏差を返します	=STDEV(number1 [, number2, ...])

STDEVA	数値、テキスト、論理値を含む標本標準偏差を返します	=STDEVA(number1 [, number2, ...])
STDEVP	母集団全体の標準偏差を返します	=STDEVP(number1 [, number2, ...])
STDEVPA	数値、テキスト、論理値を含む母集団全体の標準偏差を返します	=STDEVPA(number1 [, number2, ...])
実行時に設定	標本分散を返します	=VAR(number1 [, number2, ...])
VARA	数値、テキスト、論理値を含む標本分散を返します	=VARA(number1 [, number2, ...])
VARP	母集団全体の分散を返します	=VARP(number1 [, number2, ...])
VARPA	数値、テキスト、論理値を含む母集団全体の分散を返します	=VARPA(number1 [, number2, ...])

テキスト関数

名前	説明	構文
CODE	テキスト文字列の最初の文字の数値コードを返します	=CODE(text)
CONCATENATE	複数のテキスト項目を1つのテキスト項目に結合します	=CONCATENATE(text1 [, text2, ...])
FIND	テキストから別のテキスト値を検索します(大文字と小文字は区別されます)	=FIND(find_text, within_text [, start_num])
LEFT	テキスト値の先頭から指定された数の文字を返します	=LEFT(text[, num_chars])
LEN	テキスト文字列の長さ(文字数)を返します	=LEN(text)
LOWER	テキストを小文字に変換します	=LOWER(text)
MID	テキスト文字列の指定された位置から、指定された数の文字を返します	=MID(text, start_num, num_chars)
PROPER	テキスト値の各単語の先頭の文字を大文字に変換します	=PROPER(text)
REPLACE	テキスト内のいくつかの文字を置き換えます	=REPLACE(old_text, stat_num, num_chars, new_text)
REPT	テキストを指定された回数だけ繰り返します	=REPT(text, number_times)
RIGHT	テキスト値の末尾から指定された数の文字を返します	=RIGHT(text[, num_chars])
SEARCH	テキストから別のテキスト値を検索します(大文字と小文字は区別されません)	=SEARCH(find_text, within_text[, start_num])
SUBSTITUTE	テキスト文字列内の指定されたテキストを新しいテキストで置き換えます	=SUBSTITUTE(text, old_text, new_text[, instance_num])
T	引数をテキストに変換します	=T(value)
TEXT	数値を書式設定し、テキストに変換します	=TEXT(value, format_text)
TRIM	テキストからスペースを削除します	=TRIM(text)
UPPER	テキストを大文字に変換します	=UPPER(text)
VALUE	テキスト引数を数値に変換します	=VALUE(text)