Expression Editor for UWP

2018.07.23 更新

グレープシティ株式会社

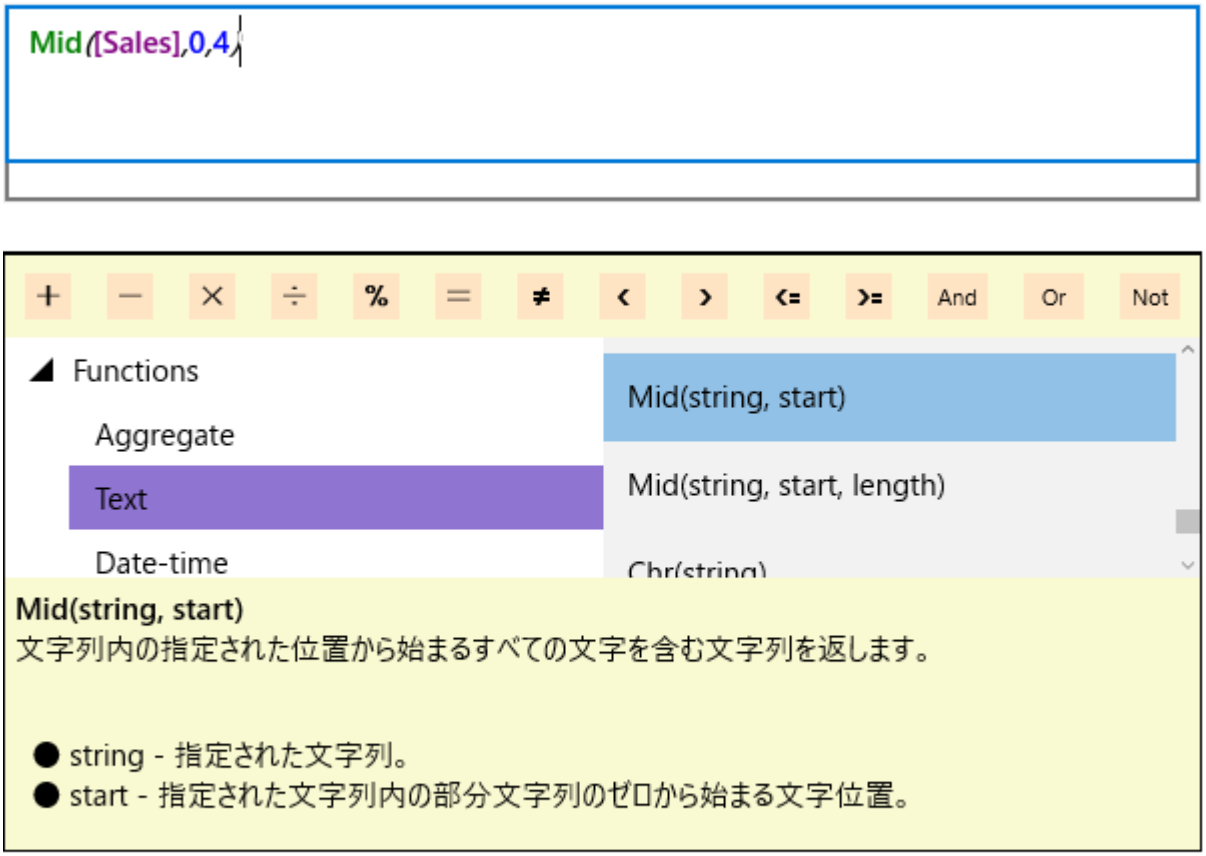
Expression Editor for UWP

目次

Expression Editor for UWPの概要

Expression Editor for UWP は、データの計算や作成に使用できる式を実行時に作成および編集するためのコントロールです。Visual Studio に似た IDE を搭載したExpression Editorは、スマートコード補完、構文の強調表示、エラー報告機能を提供します。データの集計、論理、数学、および変換演算に役立つ標準の演算子、定数、関数が用意されています。この直観的なコントロールは、グリッドやチャートなどの他のデータ管理コントロールや視覚化コントロールと簡単に統合でき、分析に役立ちます。

このコントロールは C1ExpressionEditor コンポーネントと C1ExpressionEditorPanel コンポーネントで構成されます。どちらのコンポーネントも、サポートされている他のコントロールと統合してスタンドアロンコントロールとしても使用できます。たとえば、C1ExpressionEditor コンポーネントを FlexGrid と統合して、Excel の数式バーのように式の入力に使用できます。



結果：

次のトピックでは、Expression Editor コントロールを理解し、高度な機能を説明します。

主な特長

Expression Editor for UWP には、エンドユーザーが複雑な式を作成して編集するために役立つ多数の高度な機能があります。次のものがあります。

- **スマートコード補完**
  Expression Editor は、これまでに入力された内容に基づいてリストボックスにメソッドの候補を表示します。これにより、式をすばやく完成でき、入力ミスが減ります。
- **構文の強調表示**
  Expression Editor は、関数、フィールド、演算子をそれぞれ異なる色を使用して記述します。これらの項目を区別し、特に複数行にまたがる場合に式を読みやすくできます。
- **事前定義の演算子と関数**
  Expression Editor は、式の使用中に集計、論理、日時、数学、変換、およびテキスト演算を簡単に実行できる演算子や関数を幅広く提供しています。
- **エラーレポート**
  Expression Editor は、式の入力が完了した後に即座に入力された式を検証してエラーを検出します。式の構文が正しくない場合は、エディタにエラーメッセージが表示されます。

オブジェクトモデルの概要

Expression Editor には、式エディタを備えた強力なアプリケーションを作成し、式を使用して高度な演算を実行するために便利なさまざまなクラス、オブジェクト、コレクション、関連するメソッドおよびプロパティを提供するリッチオブジェクトモデルが用意されています。これらのオブジェクトの一部とその主なプロパティを次の表に一覧します。

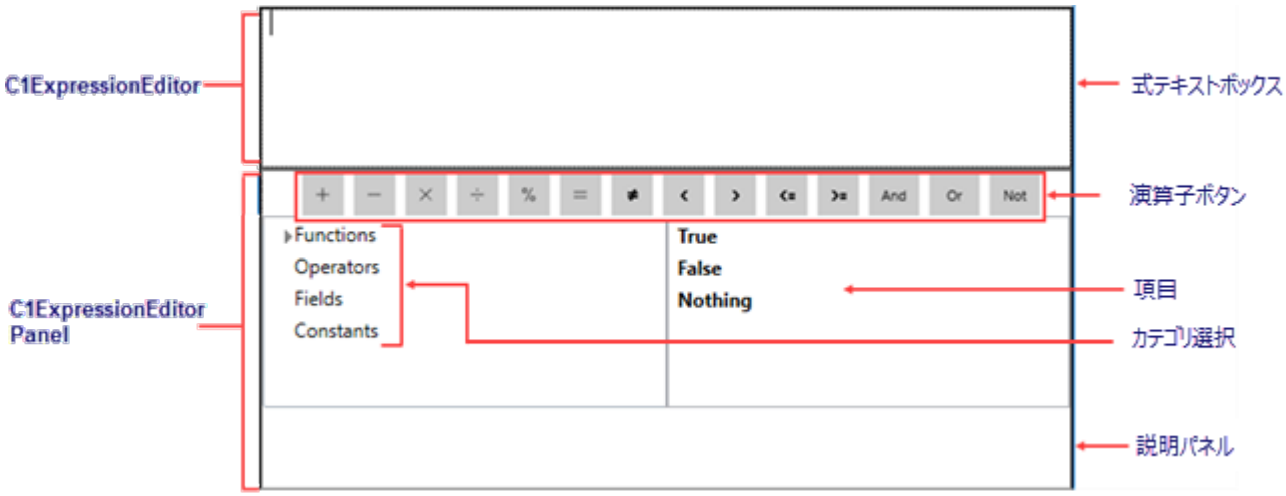
C1ExpressionEditor
プロパティ: DataSource, **Expression**, **ShowErrorBox**, **SyntaxHighlighting**, **UnderlineErrors**. **メソッド: Evaluate**, **SetCustomEngine**.
C1ExpressionEditorPanel
プロパティ: ButtonBackground, **ButtonForeground**, **ExpressionEditor**, **IsMouseOver**, **MouseOverBrush**.
ISupportExpressions インタフェース
プロパティ: ExpressionEditors, **ItemsSource**.

Expression Editor の要素

Expression Editorは、2 つのコンポーネントで構成されます。

- C1ExpressionEditor: 式を入力するためのテキストボックスを含みます。
- C1ExpressionEditorPanel: 算術および論理演算子ボタンのためのクイックアクセスツールバーと、組み込みの関数、演算子、定数のカテゴリ別ビューが表示されます。

次の図に、Expression Editor コントロールを構成するコンポーネントと、それらに対応する要素を示します。



C1ExpressionEditor

式テキストボックス:式を作成して編集するためのテキストボックス。

C1ExpressionEditorPanel

演算子ボタン:演算子のショートカットとなるボタン。

カテゴリ選択:使用可能な関数、演算子、フィールド、定数を含むツリービュー。

項目:カテゴリ選択セクションで選択された項目を一覧表示するリストボックス。

説明パネル:カテゴリ選択セクションで選択された項目の説明を表示するパネル。

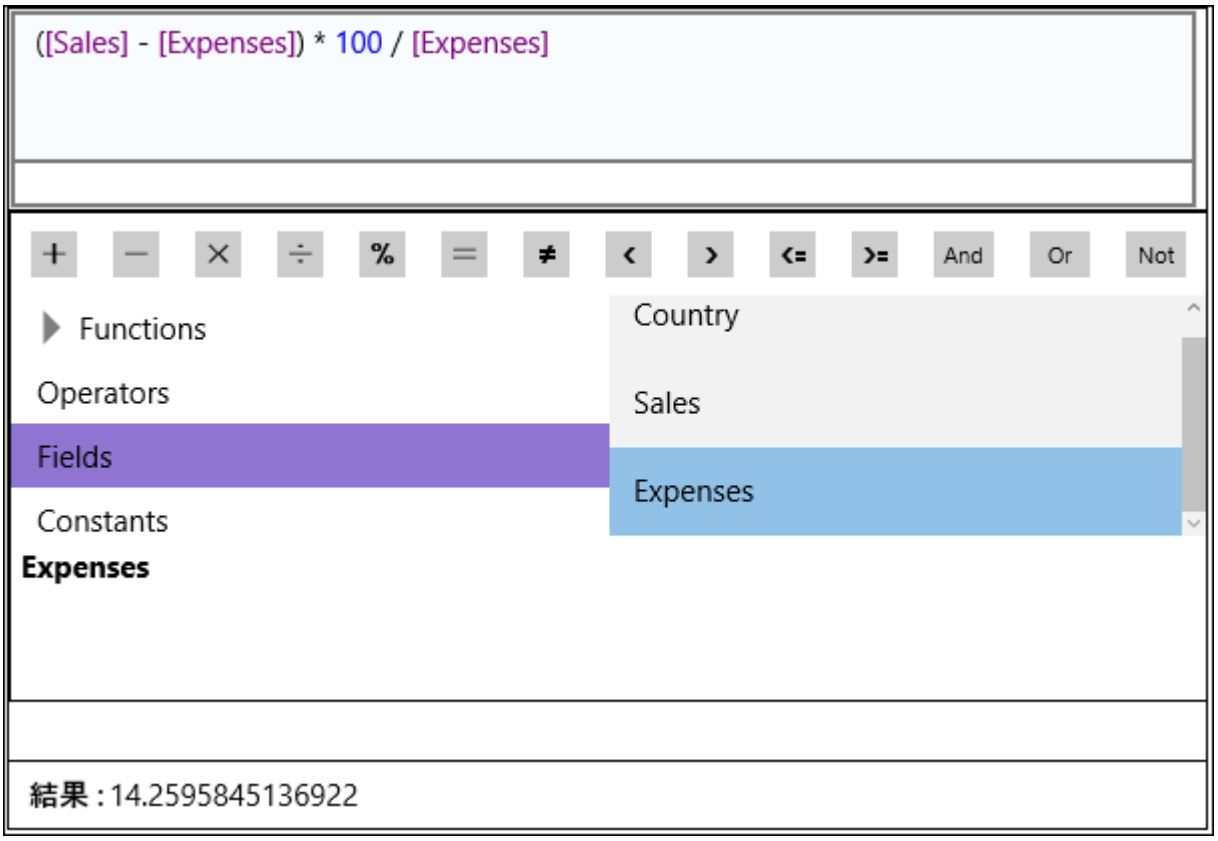
クイックスタート

このクイックスタートでは、Expression Editorコントロールを作成し、オブジェクトに連結するプロセスの手順を説明し、式を作成してフィールドで演算を実行する方法を示します。

Expression Editorコントロールを備えたアプリケーションを作成する手順は、次のとおりです。

- **手順 1:アプリケーションへのExpression Editorコンポーネントの追加**
- **手順 2:コンポーネントの連結**
- **手順 3:結果パネルとエラーパネルの追加**
- **手順 4:ExpressionChanged イベントのサブスクライブと処理**
- **手順 5:オブジェクトへのExpression Editor の連結**
- **手順 6:プロジェクトのビルドおよび実行**

次の図に、さまざまな国の売上と経費を表すクラスのインスタンスに連結された式エディタコントロールを示します。



先頭に戻る

手順 1:アプリケーションへのExpression Editorコンポーネントの追加

1. ユニバーサル Windows アプリケーションを作成し、MainPage.xaml を開きます。
2. コンポーネント C1ExpressionEditor と C1ExpressionEditorPanel をページに追加します。Xaml コードは次のようになります。
    - **Xaml**

```
<c1:C1ExpressionEditor x:Name="ExpressionEditor"
                       Width="{Binding ActualWidth, ElementName=ExpressionPanel}"
                       HorizontalAlignment="Left"
                       VerticalAlignment="Top"
                       MinHeight="100" />
<c1:C1ExpressionEditorPanel x:Name="ExpressionPanel"
                            HorizontalAlignment="Left"
                            Height="300"
                            Width="599" />
```

先頭に戻る

手順 2：コンポーネントの連結

次のコードスニペットでは、**C1ExpressionEditorPanel** クラスが公開する **ExpressionEditor** プロパティを使用してコンポーネント C1ExpressionEditor と C1ExpressionEditorPanel を連結します。

- **Xaml**

```
<c1:C1ExpressionEditor x:Name="ExpressionEditor"
                       Width="{Binding ActualWidth, ElementName=ExpressionPanel}"
                       HorizontalAlignment="Left"
                       VerticalAlignment="Top"
                       MinHeight="100" />
<c1:C1ExpressionEditorPanel x:Name="ExpressionPanel"
                            ExpressionEditor="{Binding ElementName=ExpressionEditor}"
                            HorizontalAlignment="Left"
                            Height="300"
                            Width="599" />
```

先頭に戻る

手順 3：結果パネルとエラーパネルの追加

1. 結果パネルを作成するために、C1ExpressionEditorPanel ブロックの後に StackPanel と 2 つの TextBlock を追加します。
2. エラーパネルを作成するために、StackPanel の後に TextBlock を追加します。

次のコードスニペットは、結果パネルとエラーパネルのマークアップを示します。

- **Xaml**

```
<StackPanel Orientation="Horizontal" Grid.Row="3" Margin="9,36,-9,-27">
    <TextBlock Text="結果 : " FontWeight="Bold"/>
    <TextBlock x:Name="Result" Text="" />
</StackPanel>
<TextBlock x:Name="Errors"
           Grid.Row="3"
           Foreground="Red"
           TextWrapping="Wrap"
           HorizontalAlignment="Left"
           VerticalAlignment="Top"
           MinHeight="25" />
```

先頭に戻る

手順 4：ExpressionChanged イベントのサブスクライブと処理

1. C1ExpressionEditor の ExpressionChanged イベントをサブスクライブします。
   - **C#**

   ```
   ExpressionEditor.ExpressionChanged += ExpressionEditor_ExpressionChanged;
   ```
2. ExpressionChanged イベントを処理して、入力された式の結果を次のように表示します。
   - **C#**

   ```
   private void ExpressionEditor_ExpressionChanged(object sender, EventArgs e)
   {
       C1ExpressionEditor edit = sender as C1ExpressionEditor;

       if (!edit.IsValid)
       {
           Result.Text = "";
           Errors.Text = "";
       }
       else
   ```

```
        {
            Result.Text = edit.Evaluate()?.ToString();
            Errors.Text = "";
        }
    }
```

先頭に戻る

手順 5:オブジェクトへのExpression Editorの連結

Expression Editorは、演算の実行対象となるデータを表すクラスのインスタンスに連結できます。それには、**C1ExpressionEditor** クラスが公開する **DataSource** オブジェクトを使用します。

次のコードスニペットは、Expression Editorのオブジェクトをクラスのオブジェクトに連結する方法を示します。

- **C#**

```
// オブジェクトバインディング
ExpressionEditor.DataSource = new DataItem("フランス", 1210009, 1059000);
```

この例のExpression Editorは、1 つの国の 1 つの会社の売上データと経費データを表すクラスのインスタンスに連結します。

- **C#**

```
// オブジェクトバインディング用のデータ
public class DataItem
{
    public DataItem(string country, int sales, int expenses)
    {
        Country = country;
        Sales = sales;
        Expenses = expenses;
    }

    public string Country { get; set; }
    public int Sales { get; set; }
    public int Expenses { get; set; }

}
```

先頭に戻る

手順 6:プロジェクトのビルドおよび実行

1. **[ビルド]→[ソリューションのビルド]**をクリックして、プロジェクトをビルドします。
2. **[F5]**キーを押してプロジェクトを実行します。

いくつかの国の売上と経費の数値を表すクラスのオブジェクトに連結されたExpression Editorコントロールを表示するユニバーサル Windows アプリケーションを正しく作成できました。

有効な式を入力します。たとえば、ある国の小売店の利益率の合計を計算します。国の販売と経費のデータは、オブジェクトから提供されます。

先頭に戻る

式の作成

Expression Editorでは、実行時に複雑な式を作成および編集できます。式を使用したデータの作成に役立つ関数、演算子、定数の形式で、Excel と同様の数式を提供します。

式を作成するには、次のいずれかの方法を使用します。

- C1ExpressionEditor のテキストボックスで入力を開始するだけです。入力された文字列をコントロールが式に変換します。
- C1ExpressionEditorPanel で、使用可能な関数や演算子をリストから選択します。

> 変数は角かっこ [] で囲みます。

組み込みの関数と演算子を使用した式の作成の例を次に示します。

論理および集計関数の使用	**説明**
Iif([Price] + [Cost]) > 500,300,400)	この式は、式「([Price] + [Cost]) > 500」で示される条件に基づいて、300 と 400 の 2 つの定数を評価します。
IsNull([Sales])	この式は、Sales が null かどうかを評価します。
DateTimeFunctions の使用	**説明**
AddDays([OrderDate], 30)	この式は、整数値 30 に当たる日数を DateTime 型の OrderDate 値に追加します。
DateDiffDay([DateStart], [DateEnd])	この式は、DateStart と DateEnd の間の日数をカウントします。
算術関数の使用	**説明**
Sign([Value])	この式は、提供された値の符合を示す整数値を返します。
Exp([Value])	この式は、自然対数の底のべき乗値を計算して返します。
文字列関数の使用	**説明**
Mid([Name],0,4)	この式は、指定された [Name] 文字列の先頭から、指定された文字数 4 の文字列を返します。
Remove([Name], 0, 3)	この式は、0 として指定された文字列の先頭から、指定された文字数 3 の文字を削除します。

組み込み関数、演算子、定数

Expression Editorには、次の組み込み機能があります。

- **関数**
- **演算子**
- **定数**

関数

集計関数	構文	説明
Average(value1....valueN)	Average()	指定された数値シーケンス（または列挙可能型）の平均を計算します。
Count(value1....valueN)	Count()	指定されたシーケンスに実際に含まれる要素の数を取得します
First(value1....valueN)	First()	指定されたシーケンスの最初の要素を返します。
Last(value1....valueN)	Last()	指定されたすべての要素の最後の要素を返します。
Max(value1....valueN)	Max()	指定された数値シーケンス（または列挙可能型）の最大値を返します。
Min(value1....valueN)	Min()	指定された数値シーケンス（または列挙可能型）の最小値を返します。
Sum(value1....valueN)	Sum()	指定された数値シーケンス（または列挙可能型）の合計を計算します。
DateTime 関数	**構文**	**説明**
Now()	Now()	ローカルタイムで表される、このコンピュータの現在の日時に設定された System.DateTime オブジェクト。
Today()	Today()	現在の日付を取得します。
AddDays(DateTime, DaysCount)	AddDays(DateTime date, int days)	指定された System.DateTime 値に指定された日数を加算した新しい System.DateTime を返します。
AddHours(DateTime, HoursCount)	AddHours(DateTime date, int hours)	指定された System.DateTime 値に指定された時間数を加算した新しい System.DateTime を返します。
AddMilliSeconds(DateTime, MilliSecondsCount)	AddMilliSeconds(DateTime date, int milliSeconds)	指定された System.DateTime 値に指定されたミリ秒数を加算した新しい System.DateTime を返します。
AddMinutes(DateTime, MinutesCount)	AddMinutes(DateTime date, int minutes)	指定された System.DateTime 値に指定された分数を加算した新しい System.DateTime を返します。
AddMonths(DateTime,	AddMonths(	指定された System.DateTime 値に指定された月数を加算し

MonthsCount)	DateTime date, int months)	た新しい System.DateTime を返します。
AddSeconds(DateTime, SecondsCount)	AddSeconds(DateTime date, int seconds)	指定された System.DateTime 値に指定された秒数を加算した新しい System.DateTime を返します。
AddTicks(DateTime, TicksCount)	AddTicks(DateTime date, int ticks)	指定された System.DateTime 値に指定されたティック数を加算した新しい System.DateTime を返します。
AddTimeSpan(DateTime, TimeSpan)	AddTimeSpan(DateTime date, TimeSpan timeSpan)	指定された System.DateTime 値に指定された System.TimeSpan を加算した新しい System.DateTime を返します。
AddYears(DateTime, YearsCount)	AddYears(DateTime date, int years)	指定された System.DateTime 値に指定された年数を加算した新しい System.DateTime を返します。
DateDiffDay(startDate, endDate)	DateDiffDay(,)	2 つの非 null 日付の間の日数をカウントします。
DateDiffHour(startDate, endDate)	DateDiffHour(,)	2 つの非 null 日付の間の時間数をカウントします。
DateDiffMilliSecond(startDate, endDate)	DateDiffMilliSecond(,)	2 つの非 null 日付の間のミリ秒数をカウントします。
DateDiffMinute(startDate, endDate)	DateDiffMinute(,)	2 つの非 null 日付の間の分数をカウントします。
DateDiffSecond(startDate, endDate)	DateDiffSecond(,)	2 つの非 null 日付の間の秒数をカウントします。
DateDiffTick(startDate, endDate)	DateDiffTick(,)	2 つの非 null 日付の間のティック数をカウントします。
GetDate(DateTime)	GetDate()	指定された System.DateTime 値の日付コンポーネントを取得します。
GetDay(DateTime)	GetDay()	指定された System.DateTime 値で表される日の月通算日を取得します。
GetDayOfWeek(DateTime)	GetDayOfWeek()	指定された System.DateTime 値で表される日の曜日を取得します。
GetDayOfYear(DateTime)	GetDayOfYear()	指定された System.DateTime 値で表される日の年通算日を取得します。
GetHour(DateTime)	GetHour()	指定された System.DateTime 値の時間コンポーネントを取得します。
GetMilliSecond(DateTime)	GetMilliSecond()	指定された System.DateTime 値のミリ秒コンポーネントを取得します。
GetMinute(DateTime)	GetMinute()	指定された System.DateTime 値の分コンポーネントを取得します。
GetMonth(DateTime)	GetMonth()	指定された System.DateTime 値の月コンポーネントを取得

		します。
GetSecond(DateTime)	GetSecond()	指定された System.DateTime 値の秒コンポーネントを取得します。
GetTimeOfDay(DateTime)	GetTimeOfDay()	指定された System.DateTime 値の時刻を取得します。
GetYear(DateTime)	GetYear()	指定された System.DateTime 値の年コンポーネントを取得します。
UtcNow()	UtcNow()	協定世界時（UTC）で表される、このコンピュータの現在の日時に設定された System.DateTime オブジェクト。
論理関数	**構文**	**説明**
IsNull(Value)	IsNull(object param)	指定された値が NULL の場合は True を返します。
Iif(condition, resultTrue,resultFalse)	Iif(bool condition, object expressionTrue, object expressionFalse)	条件に基づいて、2 つの式のいずれかの評価を返します。
算術関数	**構文**	**説明**
Abs(Value)	Abs()	数値の絶対値を返します。
Acos(Value)	Acos()	指定された数値を余弦とする角度を返します。
Asin(Value)	Asin()	指定された数値を正弦とする角度を返します。
Atan(Value)	Atan()	指定された数値を正接とする角度を返します。
Atan2(Value1, Value2)	Atan2(,)	指定された 2 つの数値の商を正接とする角度を返します。
Ceiling(Value)	Ceiling()	指定された 10 進数値または double 値以上の最小の整数値を返します。
Cos(Value)	Cos()	指定された角度の余弦を返します。
Cosh(Value)	Cosh()	指定された角度の双曲線余弦を返します。
Exp(Value)	Exp()	指定された e（自然対数の底）のべき乗を返します。
Floor(Value)	Floor()	指定された 10 進数または double 値以下の最大の整数を返します。
Log(Value)	Log()	指定された数の自然対数（底 e）または指定された数の指定された数を底とする対数を返します。
Log(Value, Base)	Log(,)	指定された数の自然対数（底 e）または指定された数の指定された数を底とする対数を返します。
Log10(Value)	Log10()	指定された数値の 10 を底とする対数を返します。
Pow(Value1, Value2)	Pow()	指定された数値の指定されたべき乗を返します。
Rand(Value)	Rand()	負ではない乱数を返します。
RandBetween(Value1, Value2)	RandBetween()	指定された範囲内の乱数を返します。
Sign(Value)	Sign()	数値の符合を示す整数値を返します。

Sin(Value)	Sin()	指定された角度の正弦を返します。
Sinh(Value)	Sinh()	指定された角度の双曲線正弦を返します。
Sqrt(Value)	Sqrt()	指定された数値の平方根を返します。
Tan(Value)	Tan()	指定された角度の正接を返します。
Tanh(Value)	Tanh()	指定された角度の双曲線正接を返します。
変換関数	**構文**	**説明**
CBool(string)	CBool()	指定された論理値の文字列表現を相当する System.Boolean 値に変換します。文字列が System.Boolean.TrueString または System.Boolean.FalseString の値に該当しない場合は、例外が生成されます。
CByte(string)	CByte()	数値の文字列表現を相当する System.Byte 値に変換します。
CChar(string)	CChar()	指定された文字列の値を相当する Unicode 文字に変換します。
CDate(string)	CDate()	指定された日時の文字列表現を相当する System.DateTime 値に変換します。
Cdbl(string)	CDbl()	数値の文字列表現を相当する倍精度浮動小数点数値に変換します。
CDec(string)	CDec()	数値の文字列表現を相当する System.Decimal 値に変換します。
CInt(string)	CInt()	数値の文字列表現を相当する 32 ビット符合付き整数値に変換します。
CLng(string)	CLng()	数値の文字列表現を相当する 64 ビット符合付き整数値に変換します。
CObj(value)	CObj()	指定された要素を表す System.Object を返します。
CSByte(string)	CSByte()	数値の文字列表現を相当する 8 ビット符合付き整数値に変換します。
CShort(string)	CShort()	数値の文字列表現を相当する 16 ビット符合付き整数値に変換します。
CSng(string)	CSng()	数値の文字列表現を相当する単精度浮動小数点数値に変換します。
CStr(value)	CStr()	指定された式を評価し、結果を文字列として返すように試みます。
CType(value,type)	CType(,)	指定されたオブジェクトに相当する値を持つ、指定された型のオブジェクトを返します。 value - 変換するオブジェクト。 type - 返されるオブジェクトの型名。
CUint(value)	CUint()	数値の文字列表現を相当する 32 ビット符合なし整数値に変換します。
CULong(value)	CULong()	数値の文字列表現を相当する 64 ビット符合なし整数値に変換します。

CUShort(value)	CUShort()	数値の文字列表現を相当する 16 ビット符合なし整数値に変換します。
テキスト関数	**構文**	**説明**
Replace(string,oldValue,newValue)	Replace(,,)	指定された文字列値のすべてを別の文字列値に置き換えます。
Rset(value, length)	RSet(,)	現在の文字列の末尾をスペースまたは指定された文字で埋めて、指定された長さの新しい文字列を返します。
Rset(value, length,char)	RSet(,,)	現在の文字列の末尾をスペースまたは指定された文字で埋めて、指定された長さの新しい文字列を返します。
Remove(string,start)	Remove(,)	このインスタンスから指定された位置以降のすべての文字を削除します。
Remove(string,start,count)	Remove(,,)	このインスタンスから指定された位置以降のすべての文字を削除します。
LSet(string,length)	LSet(,)	現在の文字列の先頭をスペースまたは指定された文字で埋めて、指定された長さの新しい文字列を返します。
LSet(string,length,char)	LSet(,,)	現在の文字列の先頭をスペースまたは指定された文字で埋めて、指定された長さの新しい文字列を返します。
UCase(string)	UCase()	文字列の小文字データが大文字に変換されて返されます。
LCase(string)	LCase()	文字列の大文字データを小文字に変換して返します。
Insert(string,index,value)	Insert(,,)	このインスタンスの指定されたインデックス位置に指定された文字列を挿入して新しい文字列を返します。
Len(string)	Len()	指定された文字列式の文字数を返します。
Trim(string)	Trim()	文字列の先頭または末尾からスペース文字 char(32) または指定された文字を削除します。
StartsWith(string,value)	StartsWith(,)	この文字列インスタンスの先頭が指定された文字列と一致するかどうかを判定します。
StrReverse(string)	StrReverse()	文字列値を逆順序で返します。
EndsWith(string,value)	EndsWith(,)	この文字列インスタンスの末尾が指定された文字列と一致するかどうかを判定します。
Contains(string,value)	Contains(,)	この文字列内に指定された部分文字列があるかどうかを示す値を返します。
InStr(string,value)	InStr(,)	式から別の式を検索し、見つかった場合は最初の位置を返します。
InStr(string,value,start)	InStr(,,)	式から別の式を検索し、見つかった場合は最初の位置を返します。
Mid(string,start)	Mid(,)	文字列内の指定された位置から始まるすべての文字を含む文字列を返します。
Mid(string,start,length)	Mid(,,)	文字列内の指定された位置から始まり、指定された数の文字を含む文字列を返します。
Chr(string)	Chr()	整数値の ASCII コードを文字に変換します。
Asc(string)	Acs()	文字式の左端の文字の ASCII コード値を返します。

Concat(string,value1.....valueN)	Concat(,)	2 つ以上の文字列値を連結した結果の文字列を返します。

先頭に戻る

演算子

演算子	構文	説明
プラス	+	2 つの数を合計します。
連結	&	2 つの式の文字列連結を生成します。
マイナス	-	2 つの数値の差を求めます。または、数式の符号を逆にした値を示します。
乗算	*	2 つの数を乗算します。
除算	/	2 つの数値を除算し、浮動小数点数で結果を返します。
剰余	Mod	2 つの数値で除算を実行し、剰余だけを返します。
等しい	=	左右の式が等しいかどうかを示す Boolean 値を返します。
より大きい	>	左の式が右の式より大きいかどうかを示す Boolean 値を返します。
より小さい	<	左の式が右の式より小さいかどうかを示す Boolean 値を返します。
等しくない	<>	左右の式が等しくないかどうかを示す Boolean 値を返します。
以上	>=	左の式が右の式以上かどうかを示す Boolean 値を返します。
以下	<=	左の式が右の式以下かどうかを示す Boolean 値を返します。
And	And	2 つの式の論理積を実行します。
Or	Or	2 つの式の論理和を実行します。
Not	Not	式の論理否定を実行します。

先頭に戻る

定数

定数	構文	説明
True	True	true を返します。
False	False	false を返します。
Nothing	Nothing	Null を返します。

先頭に戻る

機能

Expression Editor は、式エディタを備えたアプリケーションを作成し、複雑な式を使用できるようにするために、次の機能をサポートしています。

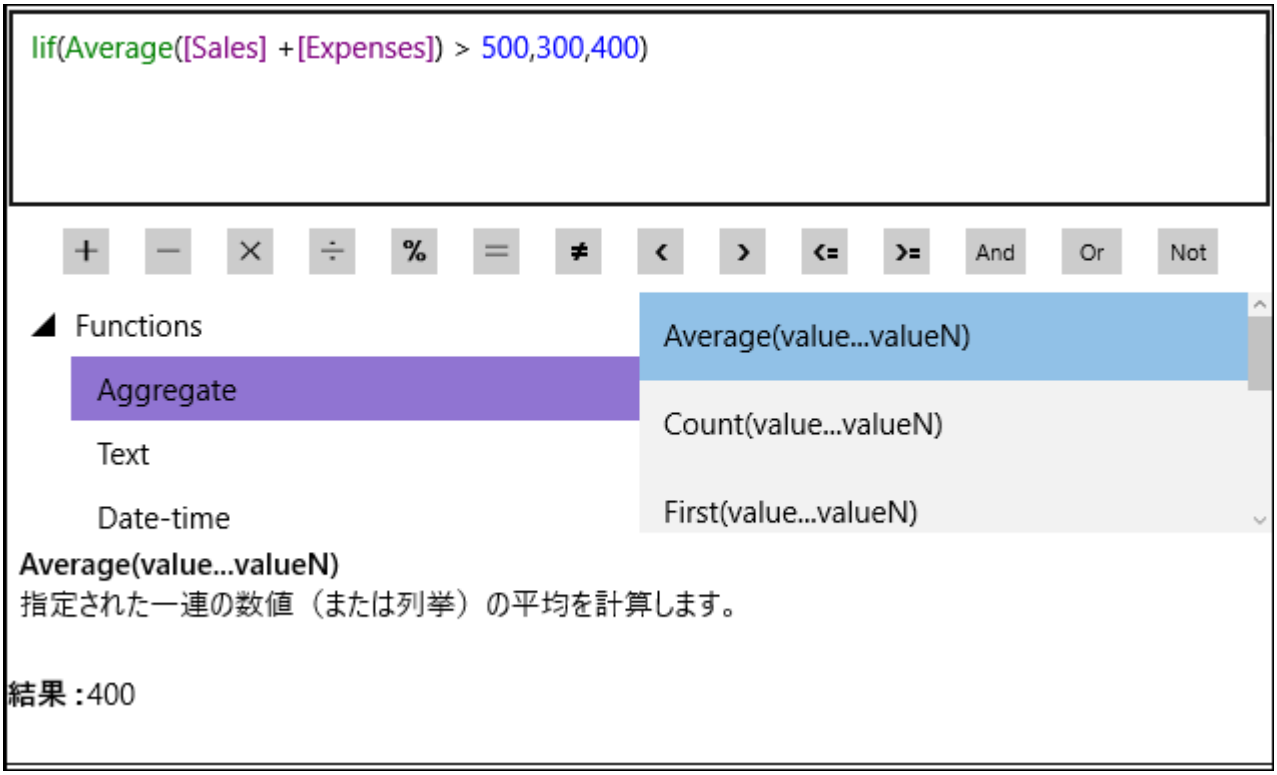
エンドユーザー機能
: Expression Editorの機能について説明します。これらの機能を使用して、エンドユーザーはコントロールを操作し、式の作成と編集を簡単に行うことができます。

外観とスタイル設定
: Expression Editor コントロールの外観を変更する方法を学習します。

エンドユーザー機能

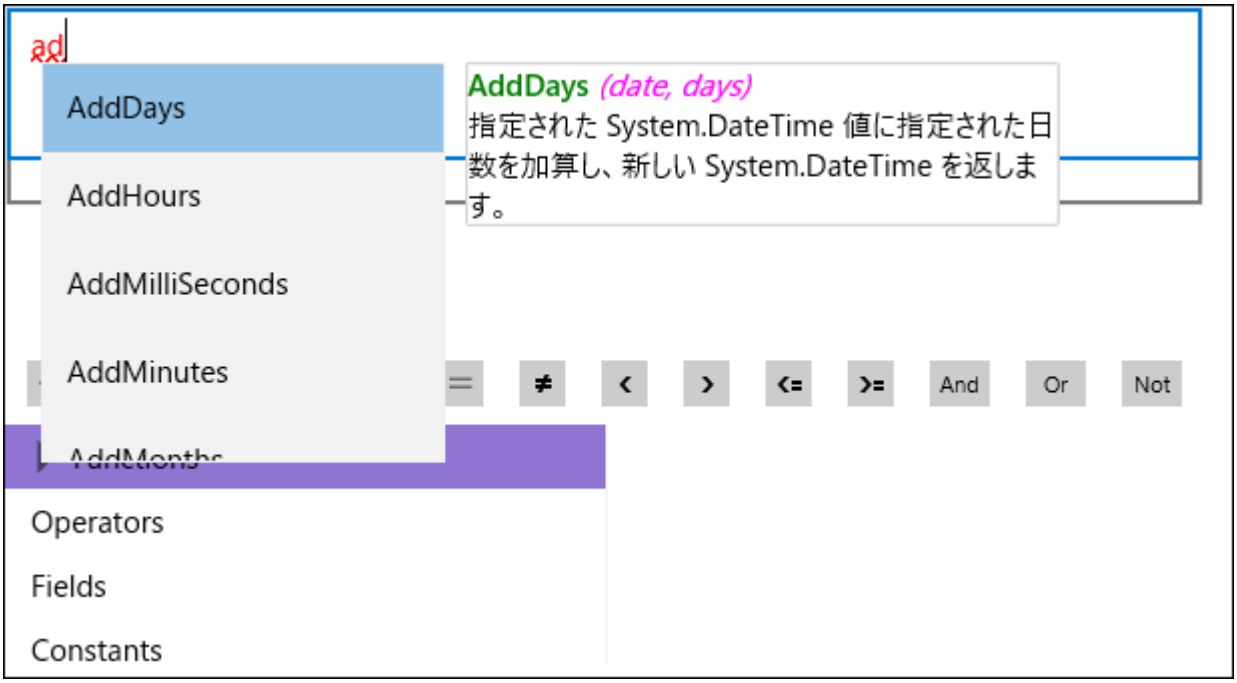
Expression Editorは、エンドユーザーがこのコントロールを操作するために便利なさまざまな機能を提供します。Expression Editor コントロールには、次の機能があります。

構文の強調表示



Expression Editorでは、SQL Query エディタと同様に、関数やフィールドを異なる色で表示できます。これらの項目が色分けされるため、関数、演算子、フィールドの区別が容易になり、式が読みやすくなります。C1ExpressionEditor クラスの **SyntaxHighlighting** プロパティは、式の項目を強調表示するかどうかを制御します。演算子と定数は強調表示されません。

スマートコード補完



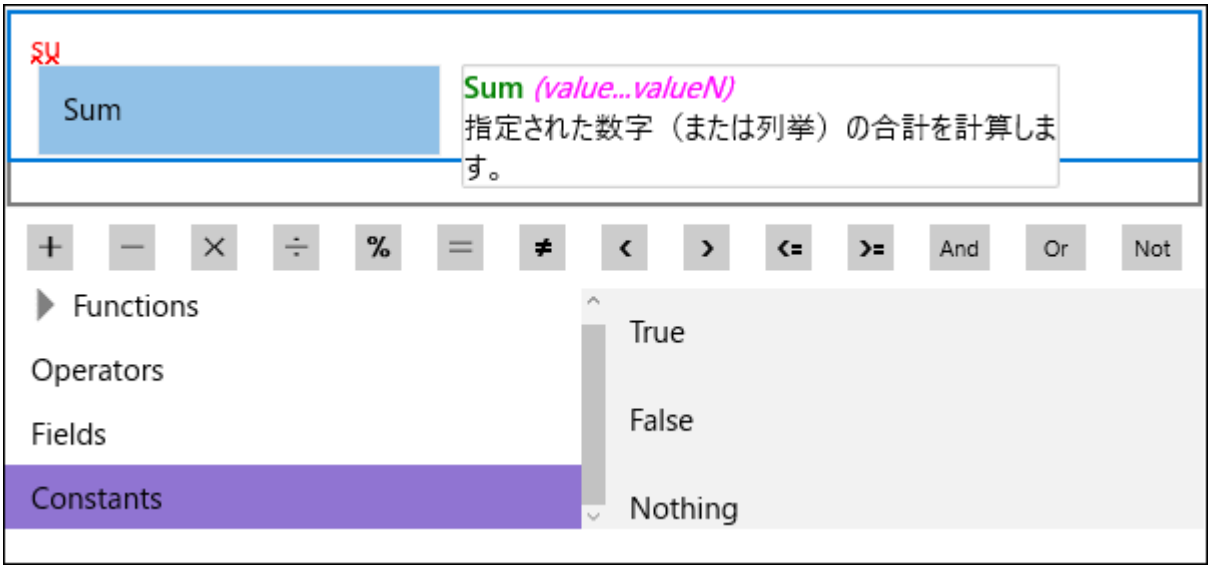
式を入力すると、Expression Editorは入力に基づいてリストに関数やフィールドの候補を表示して入力を補完します。この機能により、複雑な式をすばやく完成でき、入力ミスが減ります。

> ⚠ スマートコード補完機能はキーボード入力だけに反応し、エディタでテキストを貼り付けたときには候補は表示されません。

エラーレポート

Expression Editorは、式の記述中に構文の誤りが発生することを防ぎます。エディタは入力された式を即座に検証し、誤った構文を検出すると即座にエラーメッセージを表示します。C1ExpressionEditor クラスの **IsValid** プロパティを使用して、この動作を制御します。

ツールチップヘルパー



マウスが関数に置かれるたびに、Expression Editorは、ツールチップに説明と構文を表示します。

結果プレビュー

Expression Editorは、式を完成させる前に最終的な出力を表示することでユーザーが間違いを修正できるようにします。有効な式を記述している場合は、プレビューに結果が表示されます。無効な式を記述すると、エラーが表示されます。

先頭に戻る

外観とスタイル設定

各要素の外観を変化させることで、Expression Editor の外観をカスタマイズできます。

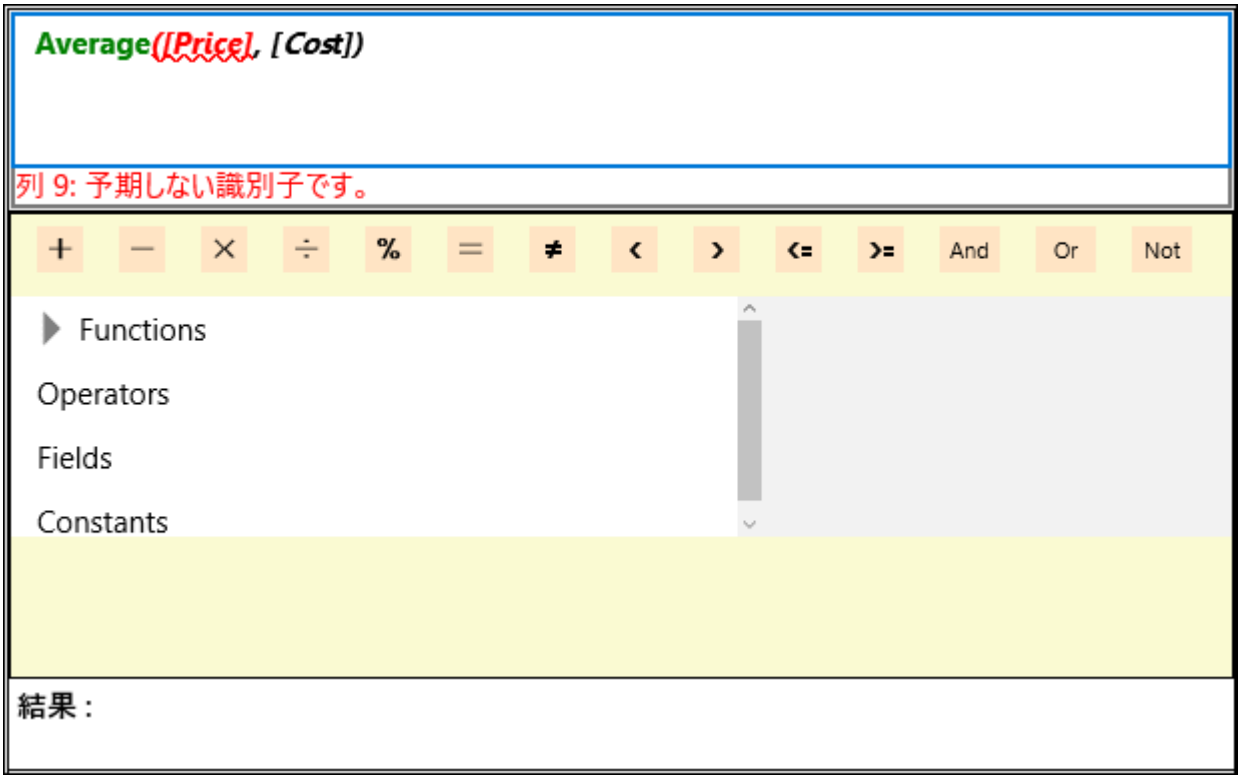
C1ExpressionEditorPanel クラスは、Expression Editor のパネルの外観を変更するための次のプロパティを提供します。

プロパティ	目的
ButtonBackground	コントロール内のボタンの背景色を設定します。
ButtonForeground	コントロール内のボタンの前景色を設定します。
MoveOverBrush	コントロール内のボタンにマウスが置かれると、ボタンを強調表示します。
Background	コントロールの背景色を設定します。
Foreground	コントロールの前景色を取得または設定します。
BorderThickness	コントロールの境界線の幅を調整します。
BorderBrush	コントロールの境界線の色を取得または設定します。
FontFamily	コントロール内のテキストのフォントを変更します。
FontStyle	フォントがレンダリングされるスタイル（イタリック、標準、斜体）を変更します。
FontSize	コントロール内のテキストのフォントサイズを変更します。
FontStretch	コントロールのフォントの伸縮を指定します。
FontWeight	コントロールのフォントの太さを変更します。

C1ExpressionEditor クラスは、Expression Editor のテキストボックスの外観を変更するための次のプロパティを提供します。

プロパティ	目的
Background	コントロールの背景色を設定します。
Foreground	コントロールの前景色を取得または設定します。
BorderThickness	コントロールの境界線の幅を調整します。
BorderBrush	コントロールの境界線の色を取得または設定します。
FontFamily	コントロール内のテキストのフォントを変更します。
FontStyle	フォントがレンダリングされるスタイル（イタリック、標準、斜体）を変更します。
FontSize	コントロール内のテキストのフォントサイズを変更します。
FontStretch	コントロールのフォントの伸縮を指定します。
FontWeight	コントロールのフォントの太さを変更します。

次の図に、Expression Editorの外観をカスタマイズした例を示します。



次のコードは、Expression Editorコントロールの外観を変更する例を示します。

- **VB**

```
' C1ExpressionEditorコンポーネントの外観を変更します
ExpressionEditor.FontFamily = New FontFamily("Bradley Hand ITC")
ExpressionEditor.FontSize = 14
ExpressionEditor.FontStyle = FontStyle.Oblique
ExpressionEditor.FontWeight = FontWeights.Normal

' C1ExpressionEditorPanelコンポーネントの外観を変更します
ExpressionPanel.ButtonBackground = New SolidColorBrush(Colors.Bisque)
ExpressionPanel.ButtonForeground = New SolidColorBrush(Colors.Black)
ExpressionPanel.MouseOverBrush = New SolidColorBrush(Colors.Bisque)
ExpressionPanel.Background = New SolidColorBrush(Colors.LightGoldenrodYellow)
ExpressionPanel.BorderBrush = New SolidColorBrush(Colors.Black)
ExpressionPanel.BorderThickness = New Thickness(1.0)
```

- **C#**

```
// C1ExpressionEditorコンポーネントの外観を変更します
ExpressionEditor.FontFamily = new FontFamily("Bradley Hand ITC");
ExpressionEditor.FontSize = 14;
ExpressionEditor.FontStyle = FontStyle.Oblique;
ExpressionEditor.FontWeight = FontWeights.Normal;

// C1ExpressionEditorPanelコンポーネントの外観を変更します
ExpressionPanel.ButtonBackground = new SolidColorBrush(Colors.Bisque);
ExpressionPanel.ButtonForeground = new SolidColorBrush(Colors.Black);
ExpressionPanel.MouseOverBrush = new SolidColorBrush(Colors.Bisque);
ExpressionPanel.Background = new SolidColorBrush(Colors.LightGoldenrodYellow);
ExpressionPanel.BorderBrush = new SolidColorBrush(Colors.Black);
ExpressionPanel.BorderThickness = new Thickness(01.00);
```

先頭に戻る

Expression Editor の操作

以下のトピックでは、Expression Editor コントロールの高度な機能を確認します。Expression Editorでは、簡単な式だけでなく複雑な式も作成できます。Expression Editorを使用して、グリッドコントロールやチャートコントロールで使用されるデータを作成できます。

ここでは、読者が Visual Studio を使用したプログラミングに精通し、クイックスタートトピックを読み終わっていることを前提としています。

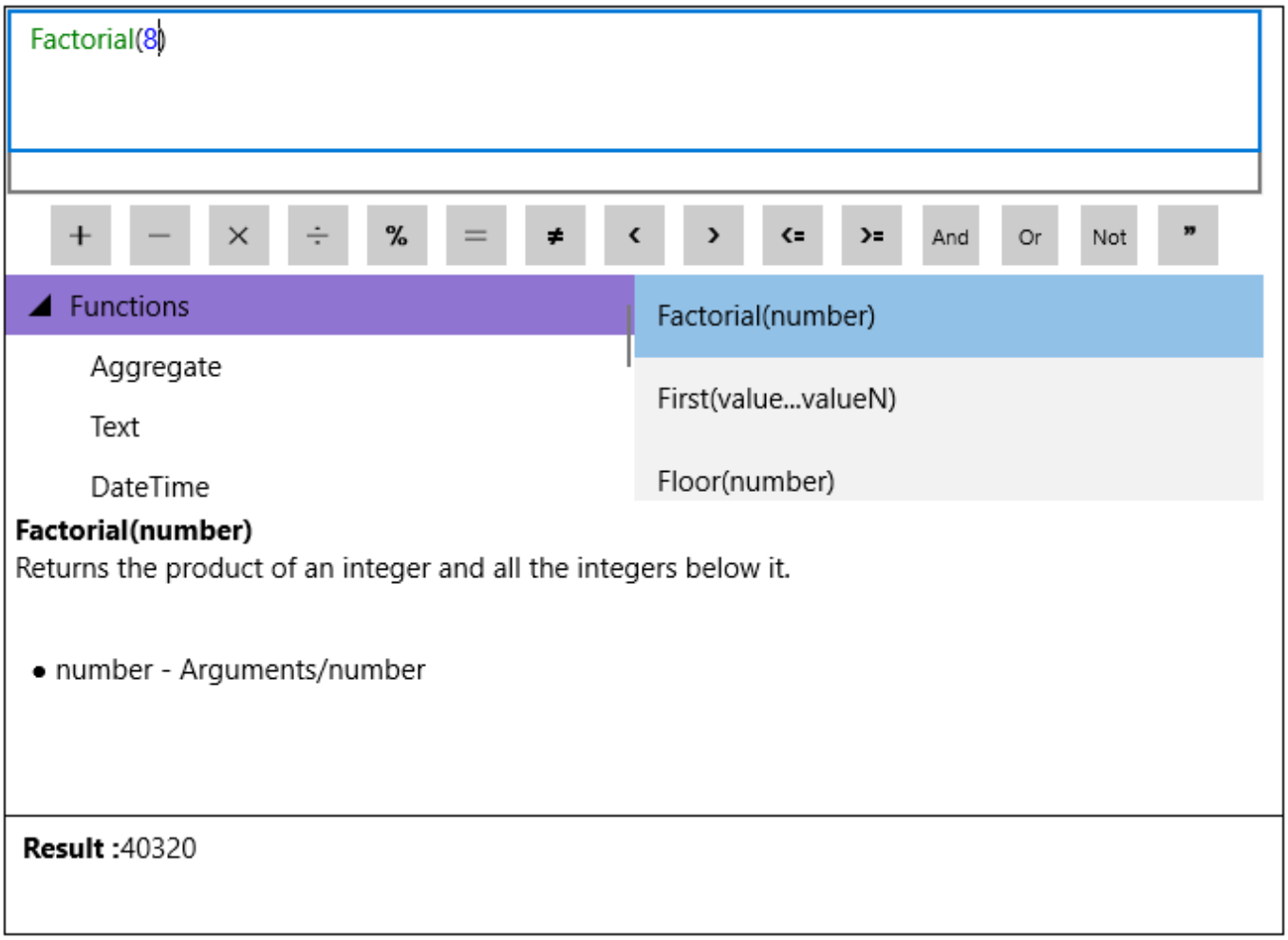
FlexGrid との統合
: Expression Editor コントロールを FlexGrid と統合し、式を使用して非連結列の列計算を実装する方法を説明します。

FlexChart との統合
: Expression Editor コントロールを FlexChart と統合する方法を説明します。また、式を使用してフィルタを適用することで、チャート表示を変更します。

カスタム関数の作成

Expression Editor provides various built-in functions to create expressions for your applications. Furthermore, it allows you to define custom functions according to your application's requirement. The **AddFunction** method of **C1ExpressionEditor** class allows you to add custom functions to Expression Editor. This custom function gets added to the ExpressionEditor engine and is accessible at runtime by C1ExpreesionEditor for performing calculations.

The following image shows a custom function used in the ExpressionEditor control.



The following code demonstrates how a custom function is created and added to the Expression Editor panel:

Visual Basic

```
Public NotInheritable Partial Class MainPage
    Inherits Page

    Public Sub New()
        Me.InitializeComponent()
        ExpressionPanel.ExpressionEditor = ExpressionEditor
        Dim factItem As ExpressionItem = New ExpressionItem("Factorial",
"Factorial()", "Returns the product of an integer and all the integers
below it.", ItemType.MathFuncs)
        factItem.Arguments.Add(New Argument("number",
GetType(System.Double)))
        ExpressionEditor.AddFunction(factItem, AddressOf Factorial, 1,
1)
    End Sub

    Private Function Factorial(ByVal list As List(Of Object)) As Object
        Dim items As List(Of Object) = New List(Of Object)()
        items = items.Concat(list).ToList()
        Dim i, number As Integer, fact As Integer = 1
        number = Convert.ToInt32(items(0))

        For i = 1 To number
            fact = fact * i
        Next

        Return fact
    End Function

    Private Sub ExpressionEditor_ExpressionChanged(ByVal sender As
Object, ByVal e As EventArgs)
        If Not ExpressionEditor.IsValid Then
            Result.Text = ""
        Else
            Result.Text = ExpressionEditor.Evaluate()?.ToString()
        End If
    End Sub
End Class
```

- **C#**

```
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        this.InitializeComponent();

        ExpressionPanel.ExpressionEditor = ExpressionEditor;
        ExpressionItem factItem = new ExpressionItem("Factorial", "Factorial()",
        "Returns the product of an integer and all the integers below it.",
        ItemType.MathFuncs);
        factItem.Arguments.Add(new Argument("number", typeof(System.Double)));
        ExpressionEditor.AddFunction(factItem, Factorial, 1, 1);
    }
```

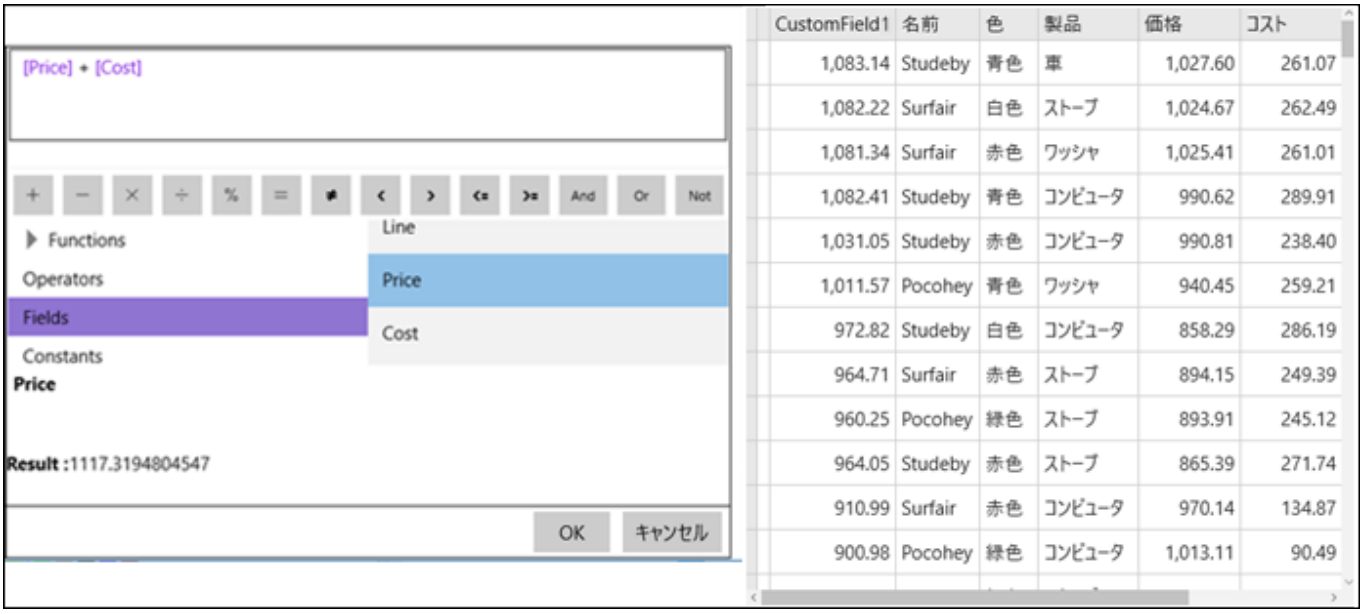
```
    private object Factorial(List<object> list)
    {
        List<object> items = new List<object>();
        items = items.Concat(list).ToList();

        int i, fact = 1, number;
        number = Convert.ToInt32(items[0]);
        for (i = 1; i <= number; i++)
        {
            fact = fact * i;
        }

        return fact;
    }

    private void ExpressionEditor_ExpressionChanged(object sender, EventArgs e)
    {
        if (!ExpressionEditor.IsValid)
        {
            Result.Text = "";
        }
        else
        {
            Result.Text = ExpressionEditor.Evaluate()?.ToString();
        }
    }
}
```

FlexGrid との統合

Expression Editor は、FlexGrid コントロールとの統合をサポートします。Expression Editorをグリッドと統合すると、式を使用してグリッドデータのフィルタ処理、グループ化、ソート、列計算などの演算を実行できます。

Expression Editorを FlexGrid と統合するには、グリッドデータを生成するためのオブジェクトのコレクションを受け取る、C1FlexGrid クラスの ItemsSource プロパティを使用する必要があります。グリッドを生成したら、C1ExpressionEditor クラスの **DataSource** プロパティを使用して、Expression Editorのデータソースを FlexGrid のデータソースに連結できます。

次の図に、FlexGrid コントロールと統合されたExpression Editorを示します。このグリッドには、各ハードウェア製品の機能とコストの情報が表示されます。



次のコードは、FlexGrid と Expression Editor の統合を示します。

次のコードスニペットに示すように、DataSource プロパティから FlexGrid の C1CollectionView コンポーネントのインスタンスを Expression Editor に連結します。

- **VB**

```
Me.InitializeComponent()
    Dim items As List(Of Product) = Product.GetData(200)
    flexGrid.ItemsSource = items
    C1.ExpressionEditor.DataSource = flexGrid.CollectionView.CurrentItem
```

- **C#**

```
this.InitializeComponent();
List<Product> items = Product.GetData(200);
flexGrid.ItemsSource = items;
ExpressionEditor.DataSource = flexGrid.CollectionView.CurrentItem;
```

詳細なデータについては、インストーラに付属のサンプルプロジェクト **ExpressionEditorSamples** を参照してください。

次のトピックでは、式を使用した FlexGrid の列の計算について説明します。ただし、フィルタ処理、ソート、グループ化に Expression Editor を活用する方法を理解するには、インストーラに付属のサンプルとブログ記事を参照してください。

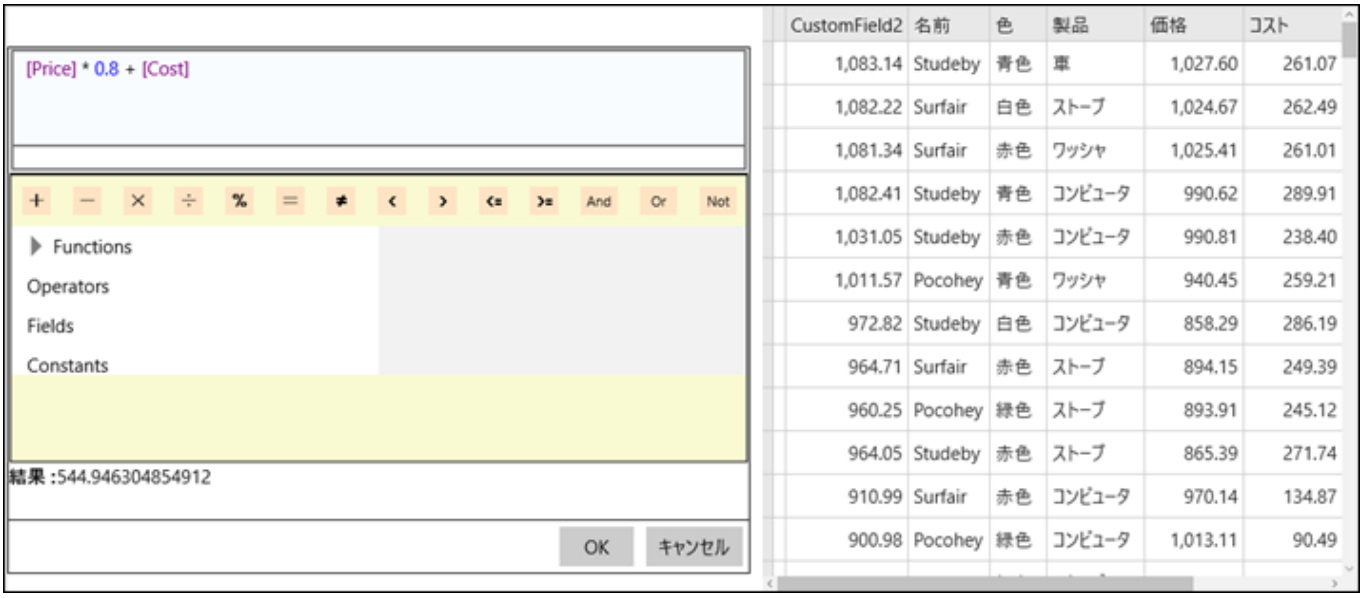
先頭に戻る

FlexGrid の列計算

Expression Editor をグリッドと統合すると、FlexGrid の非連結列の列データを計算できます。

式を使用して FlexGrid の非連結列のデータを生成するために、C1FlexGrid クラスを継承して **iSupportExpressions** インタフェース実装するクラスを作成します。次に、計算された値を表示する新しい FlexGrid の非連結列にExpression Editorを追加します。

次の図に、式を使用して列計算を行う FlexGrid コントロールの例を示します。



次のコードは、式を使用して FlexGrid 列の列計算を行う例を示します。

1. 次のコードスニペットに示すように、C1FlexGrid を継承し、**iSupportExpressions** インタフェースを実装するクラスを作成します。
   - **VB**

```
Public Class FlexGridEE
    Inherits C1FlexGrid
    Implements ISupportExpressions

    Public Sub New()
        MyBase.New()
        ExpressionEditors = New ExpressionEditorCollection(Me)
    End Sub

    ' ISupportExpressions

    Public Sub SetCellValue(row As Integer, colName As String,
                            value As Object) Implements
                        ISupportExpressions.SetCellValue
        Me(row, colName) = value
    End Sub

    Public Overloads ReadOnly Property ExpressionEditors()
                As ExpressionEditorCollection Implements
                 ISupportExpressions.ExpressionEditors

    Private Property ISupportExpressions_ItemsSource
    As IEnumerable Implements ISupportExpressions.ItemsSource
        Get
            Return Me.ItemsSource
        End Get
        Set(value As IEnumerable)
            Me.ItemsSource = value
        End Set
    End Property

End Class
```

   - **C#**

```
public class FlexGridEE : C1FlexGrid, ISupportExpressions
{

    public FlexGridEE() : base()
```

```
        {
            ExpressionEditors = new ExpressionEditorCollection(this);
        }

        // ISupportExpressionsの使用
        public void SetCellValue(int row, string colName, object value)
        {
            this[row, colName] = value;
        }
        public ExpressionEditorCollection ExpressionEditors { get; }

    }
```

2. ここで、次のコードスニペットに示すように、新しい FlexGrid の非連結列にExpression Editorを追加します。
    - **VB**

```
Dim c1ExpressionEditor1 As New C1ExpressionEditor()
Dim c1ExpressionEditor2 As New C1ExpressionEditor()
c1ExpressionEditor1.Expression = "[Price] + [Cost]"
c1ExpressionEditor2.Expression = "[Price] * 0.8 + [Cost]"
flexGrid.ExpressionEditors.Add("CustomField1", c1ExpressionEditor1)
flexGrid.ExpressionEditors.Add("CustomField2", c1ExpressionEditor2)
```

    - **C#**

```
C1ExpressionEditor c1ExpressionEditor1 = new C1ExpressionEditor();
C1ExpressionEditor c1ExpressionEditor2 = new C1ExpressionEditor();
c1ExpressionEditor1.Expression = "[Price] + [Cost]";
c1ExpressionEditor2.Expression = "[Price] * 0.8 + [Cost]";
flexGrid.ExpressionEditors.Add("CustomField1", c1ExpressionEditor1);
flexGrid.ExpressionEditors.Add("CustomField2", c1ExpressionEditor2);
```

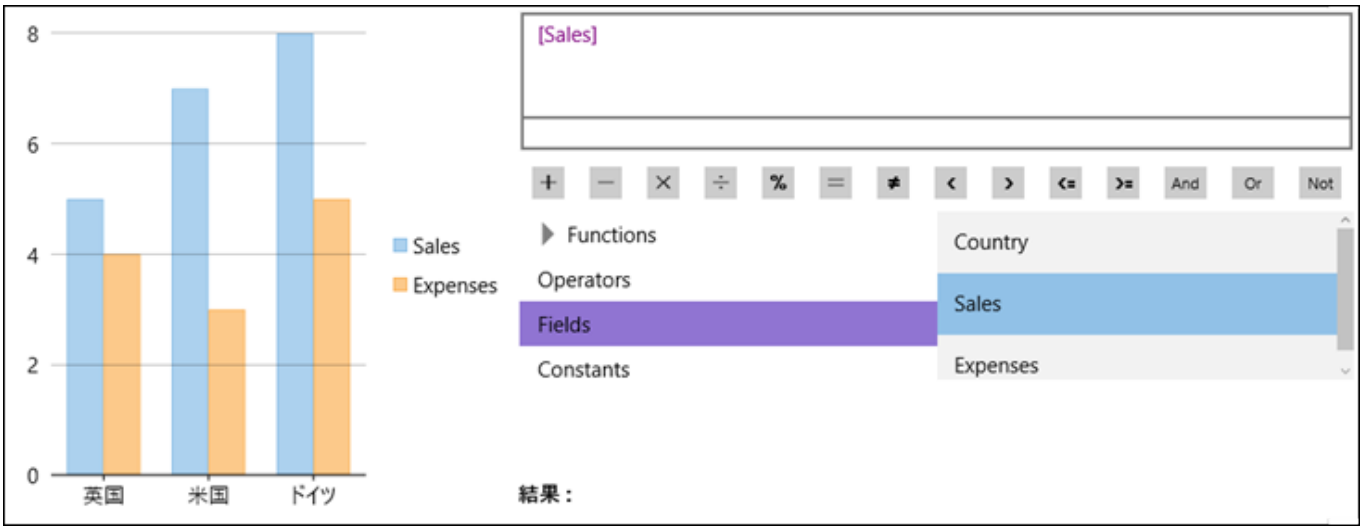
先頭に戻る

FlexChart との統合

Expression Editor は、FlexChart コントロールとの統合をサポートします。Expression Editorをチャートと統合すると、式を使用してチャートの視覚化を操作できるようになります。

Expression Editor を FlexChart と統合するには、**CollectionViewSource** オブジェクトから系列データを含むオブジェクトのコレクションを受け取る、C1FlexChart クラスの ItemsSource プロパティを使用する必要があります。系列データを取得したら、**DataSource** プロパティを使用して、FlexChart のデータソースをExpression Editorのデータソースに連結できます。

次の図に、FlexChart コントロールと統合されたExpression Editorを示します。このチャートは、ある小売店の国ごとの販売と経費をプロットします。



次のコードは、FlexChart と Expression Editor コントロールの統合を示します。

C1CollectionView 型のフィールドを作成します。

VB

```
Public View As C1CollectionView
```

C#

```
public C1CollectionView View;
```

FlexChart の C1CollectionView コンポーネントのインスタンスを Expression Editor にその DataSource プロパティを通して連結します。

- **VB**

```
Me.InitializeComponent()
'C1CollectionViewコンポーネントのインスタンスを作成します
View = New C1CollectionView(DataCreator.CreateData())
'系列のデータを含むオブジェクトの集合を取得します
flexChart.ItemsSource = View
flexChart.BindingX = "Country"
flexChart.Series.Add(New Series() With {
        .SeriesName = "Sales",
        .Binding = "Sales"
    })
flexChart.Series.Add(New Series() With {
        .SeriesName = "Expenses",
        .Binding = "Expenses"
    })
'C1CollectionViewコンポーネントのインスタンスをExpressionEditorに連結します
editor.DataSource = View.CurrentItem
```

- **C#**

```
this.InitializeComponent();

// C1CollectionViewコンポーネントのインスタンスを作成します
View = new C1CollectionView(DataCreator.CreateData());

// 系列のデータを含むオブジェクトの集合を取得します
flexChart.ItemsSource = View;
flexChart.BindingX = "Country";
flexChart.Series.Add(new Series() { SeriesName = "Sales", Binding = "Sales" });
flexChart.Series.Add(new Series() { SeriesName = "Expenses", Binding = "Expenses" });

// C1CollectionViewコンポーネントのインスタンスをExpressionEditorに連結します
editor.DataSource = View.CurrentItem;
```

詳細なデータについては、インストーラに付属のサンプルプロジェクト **ExpressionEditorSamples** を参照してください。

先頭に戻る

FlexChart のフィルタ処理

Expression Editorを FlexChart と統合すると、式を使用してデータ項目をフィルタ処理できます。

Expression Editorで入力された式に基づいて FlexChart のデータ項目をフィルタ処理するには、**C1CollectionView のフィルタ**述語を使用できます。C1Expression エディタのデータソースは FlexChart のデータソースと同じです。

次の図に、Expression Editor コントロールに入力された式を使用してフィルタ処理する FlexChart コントロールの例を示します。

次のコードは、Expression Editor コントロールに入力された式に基づいて FlexChart のビューをフィルタ処理する例を示します。

1. 「FlexChart との統合」に従って、FlexChart コントロールをExpression Editor コントロールに連結します。
2. 次のコードスニペットに示すように、フィルタデリゲートをインスタンス化します。
   - **VB**

```
View.Filter = New Predicate(Of Object)(AddressOf Contains)
```

   - **C#**

```
View.Filter = new Predicate<object>(Contains);
```

3. 次のコードスニペットに示すように、デリゲートのメソッドを定義します。
   - **VB**

```
Private Function Contains(obj As Object) As Boolean
    editor.DataSource = TryCast(obj, DataItem)
    Dim value As Object = editor.Evaluate()
    Dim ret As Object = True
    If value IsNot Nothing Then
        [Boolean].TryParse(value.ToString(), ret)
    End If
    Return ret
End Function
```

   - **C#**

```
private bool Contains(object obj)
{
    editor.DataSource = obj as DataItem;
    var value = editor.Evaluate();
    var ret = true;
    if (value != null)
        Boolean.TryParse(value.ToString(), out ret);
```

```
            return ret;
        }
```

先頭に戻る