ComponentOne

Expression Editor for WPF

GrapeCity US

GrapeCity
201 South Highland Avenue, Suite 301
Pittsburgh, PA 15206
Tel: 1.800.858.2739 | 412.681.4343
Fax: 412.681.4384
Website: https://www.grapecity.com/en/
E-mail: us.sales@grapecity.com

Trademarks

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

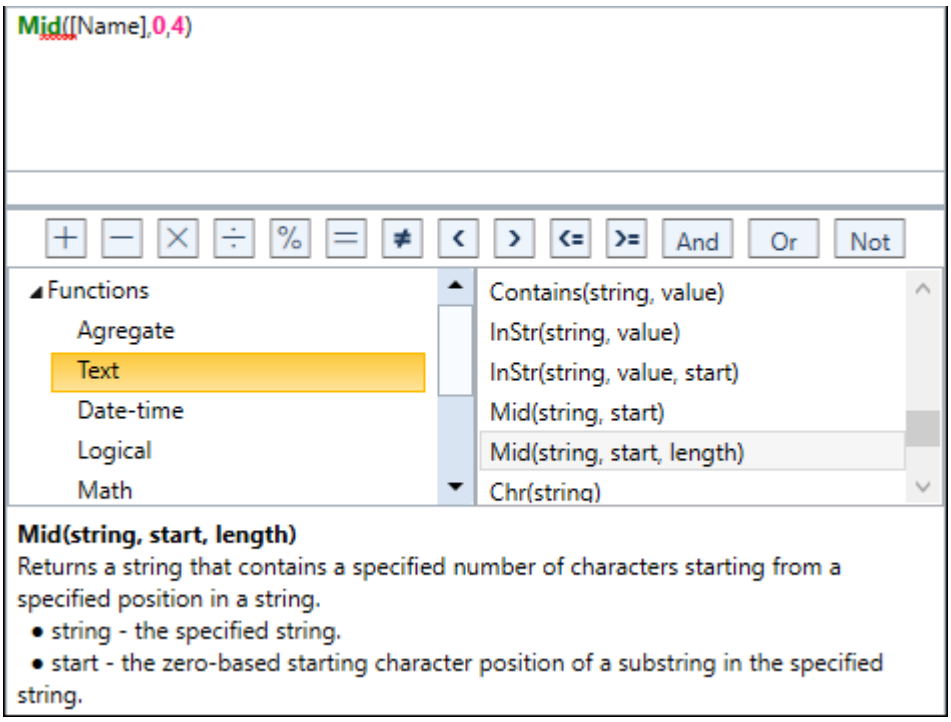
Copying and Distribution

Table of Contents

Expression Editor for WPF Edition

Expression Editor for WPF is a control that enables creating and editing expressions at run-time, which can then be used to perform calculations and shaping data. Powered with Visual Studio-like IDE, Expression Editor offers smart code completion, syntax highlighting, and error reporting functionalities. It provides standard operators, constants, and functions to help perform aggregate, logical, mathematical, and conversion operations on data. This intuitive control can be easily integrated with other data management and visualization controls such as grids and charts, to aid analysis.

The control is composed of C1ExpressionEditor and C1ExpressionEditorPanel components. Both these components can also be used as stand-alone controls in integration with other supported controls. For example, C1ExpressionEditor component can be used as Excel-like formula bar in integration with FlexGrid to enter expressions.



The following topics help you get accustomed with the Expression Editor control, and explore its advanced capabilities.

Help with WPF Edition

For information about installing **ComponentOne Studio WPF Edition**, licensing, creating project with the Expression Editor control, getting technical help, and namespaces, visit Getting Started with WPF Edition.

Key Features

Expression Editor for WPF offers numerous advanced features that help end users create and edit complex expressions. These are as follows:

- **Smart Code Completion**
  Expression Editor provides recommendation of possible methods in a list box based on what you typed so far. This helps in completing expressions quickly and reduces the chances of typing errors.
- **Syntax Highlighting**
  Expression Editor uses different colors to write functions, fields, and operators. This helps in differentiating these items and enhancing the readability of expressions, especially when the expressions span across multiple lines.
- **Pre-defined Operators and Functions**
  Expression Editor provides a wide variety of operators and functions to perform aggregate, logical, date time, math, convert and text operations for the ease of users while working with expressions.
- **Error Reporting**
  Expression Editor validates the entered expression for error detection, immediately after user has finished entering the expression. If the expression syntax is incorrect, then error message is shown within the Editor.

Object Model Summary

Expression Editor comes with a rich object model, providing various classes, objects, collections, and associated methods and properties for creating powerful applications with Expression Editor, to help perform advanced operations using expressions. The following table lists some of these objects and their major properties.

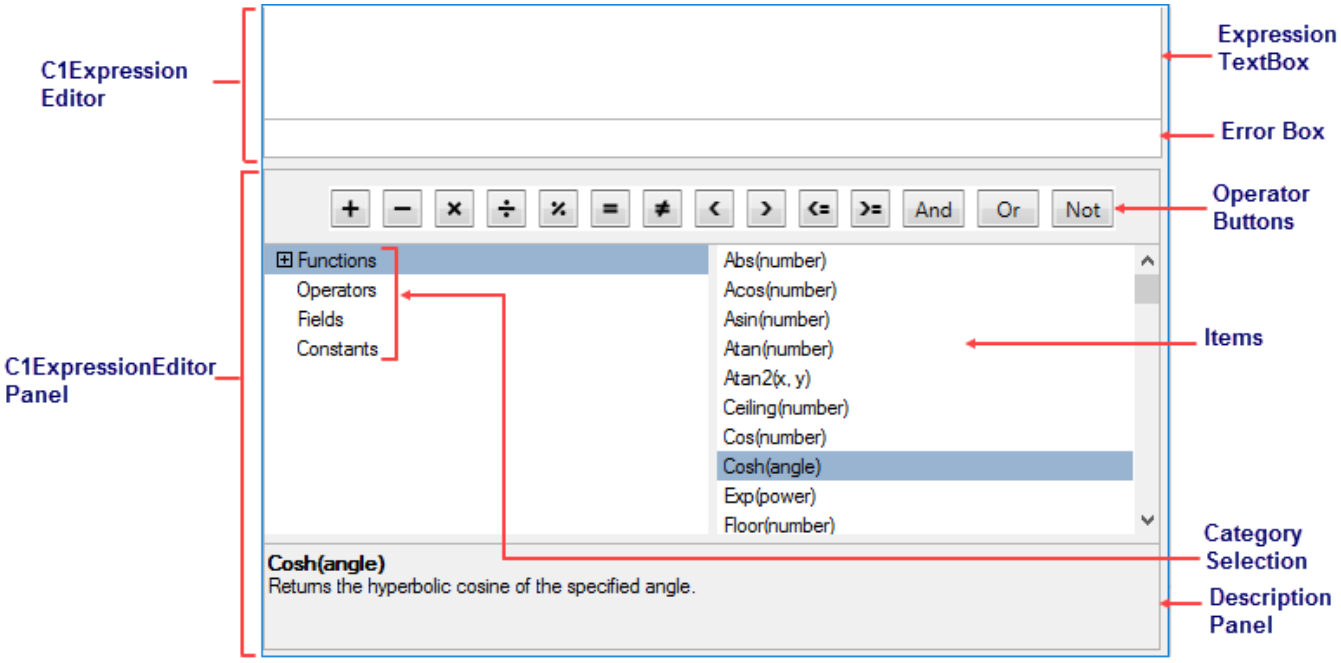
C1ExpressionEditor
Properties: DataSource, Expression, ShowErrorBox, SyntaxHighlighting, UnderlineErrors. **Methods:** Evaluate, SetCustomEngine.
C1ExpressionEditorPanel
Properties: ButtonBackground, ButtonForeground, ExpressionEditor, MouseOverBrush, PressedBrush.
ISupportExpressions Interface
Properties: ExpressionEditors, ItemsSource.

Expression Editor Elements

Expression Editor comprises two components.

- C1ExpressionEditor: Contains an expression text box where you can type expressions.
- C1ExpressionEditorPanel: Displays a quick access toolbar containing Arithmetic and Logical operator buttons, and a categorized view of built-in functions, operators, and constants.

The following image illustrates constituting components of Expression Editor control and their respective elements.



C1ExpressionEditor

Expression TextBox: A text box that enables user to create and edit expressions.

ErrorBox: Shows the error information for the entered expression.

C1ExpressionEditorPanel

Operator Buttons: Buttons that represent shortcuts for operators.

Category Selection: A tree view which contains available functions, operators, fields and constants.

Items: A list box that lists the items selected from the Category Selection section.

Description Panel: A panel that displays the description of the item selected from the Category Selection section.

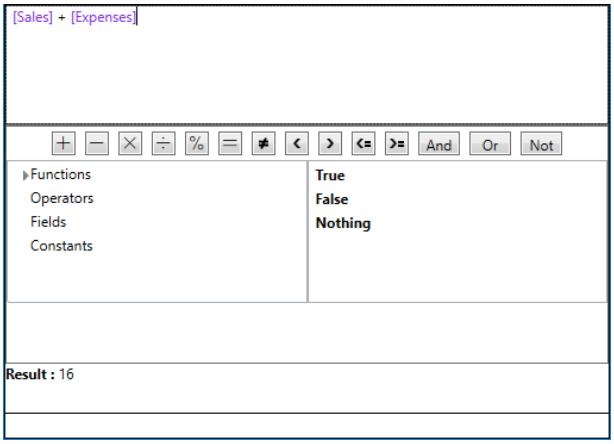
Quick Start

This quick start topic guides you through a step-by-step process of creating Expression Editor control, and binding it to an object to demonstrate creating expressions and performing operations on its fields.

The steps to create an application with Expression Editor control are as follows:

- **Step 1: Add Expression Editor components to application**
- **Step 2: Bind the components**
- **Step 3: Add Result and Error panels**
- **Step 4: Subscribe to and handle ExpressionChanged event**
- **Step 5: Bind Expression Editor to an object**
- **Step 6: Build and Run the Project**

The following image exhibits the Expression Editor control which is bound to an instance of a class that represents sales and expenses for different countries.



Back to Top

Step 1: Add Expression Editor components to application

1. Create a Universal Windows application, and open MainPage.xaml.
2. Add C1ExpressionEditor and C1ExpressionEditorPanel components to your page. The Xaml code appears as follows.
   - **Xaml**

```
<c1:C1ExpressionEditor  x:Name="ExpressionEditor"
MinHeight="100" Width="{Binding ActualWidth, ElementName=ExpressionPanel}"/>
<c1:C1ExpressionEditorPanel x:Name="ExpressionPanel"
Grid.Row="1"/>
```

Back to Top

Step 2: Bind the components

Bind the C1ExpressionEditor and C1ExpressionEditorPanel components using ExpressionEditor property exposed by **C1ExpressionEditorPanel** class, as shown in the following code snippet.

- **Xaml**

```
<c1:C1ExpressionEditor  x:Name="ExpressionEditor"
 MinHeight="100"
  Width="{Binding ActualWidth, ElementName=ExpressionPanel}"/>
<c1:C1ExpressionEditorPanel x:Name="ExpressionPanel"
 ExpressionEditor="{Binding ElementName=ExpressionEditor}"
 Grid.Row="1"/>
```

Back to Top

Step 3: Add Result and Error panels

1. To create result panel, add a StackPanel with two TextBlocks within it after the C1ExpressionEditorPanel block.
2. To create error panel, add a TextBlock after the StackPanel.

The following code snippet shows markup for result and error panels

- **Xaml**

```
<StackPanel Orientation="Horizontal" Grid.Row="2">
    <TextBlock Text="Result : " 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" />
```

Back to Top

Step 4: Subscribe to and handle the ExpressionChanged event

1. Subscribe to the ExpressionChanged event of C1ExpressionEditor.
   - **C#**

```
ExpressionEditor.ExpressionChanged += ExpressionEditor_ExpressionChanged;
```

2. Handle the ExpressionChanged event, to show result of the entered expressions, as follows.
   - **C#**

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

    if (!editer.IsValid)
    {
        Result.Text = "";
        Errors.Text = "";
```

```
            editer.GetErrors().ForEach(x => { Errors.Text += x.FullMessage + "\n"; });
        }
        else
        {
            Result.Text = editer.Evaluate()?.ToString();
            Errors.Text = "";

        }
    }
```

Back to Top

Step 5: Bind Expression Editor to an object

Expression Editor can be bound to instance of a class that represents data to perform operations on. This is possible by **DataSource** object exposed by the **C1ExpressionEditor** class.

The following code snippet demonstrates how an object of expression editor binds to object of a class.

- **C#**

```
ExpressionEditor.DataSource = new DataItem("France",11,5);
```

In this example, Expression Editor binds to the instance of a class that represents sales and expenses data for an organization in a country.

- **C#**

```
class DataCreator
{

        public static List<DataItem> CreateData()
        {
            var data = new List<DataItem>();
            data.Add(new DataItem("UK", 5, 4));
            data.Add(new DataItem("USA", 7, 3));
            data.Add(new DataItem("Germany", 8, 5));
            data.Add(new DataItem("Japan", 12, 8));
            data.Add(new DataItem("India", 24, 10));
            data.Add(new DataItem("Kenya", 20, 15));
            data.Add(new DataItem("South Africa", 9,5));
        return data;
        }
    }

    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; }

}
```

Back to Top

Step 6: Build and Run the Project

1. Click **Build | Build Solution** to build the project.
2. Press **F5** to run the project.

You have successfully created a Universal Windows application showing Expression Editor control bound to an object of class depicting sales and expenses figures for countries.

Enter a valid expression, for example calculate the sum of Sales and Expenses for a retail store in a country. The sales and expenses data for the country have been provided through object.

Back to Top

Create Expressions

Expression Editor enables creating and editing complex expressions at run-time. It provides Excel-like formulas, in the form of functions as well as operators and constants that help in shaping data using expressions.

To create expressions, use any of the following ways:

- Simply begin typing within the text box of C1ExpressionEditor. The control can convert the typed string to an expression.
- Select from the available list of functions and operators from C1ExpressionEditorPanel.


Note that variables are enclosed in square brackets [].

Some examples of creating expressions using in-built functions and operators are as follows.

Using Logical and Aggregate Functions	**Description**
Iif([Price] + [Cost]) > 500,300,400)	This expression evaluates the two constants 300 and 400 based on the condition provided by the expression "([Price] + [Cost]) > 500".
IsNull([Sales])	This expression evaluates whether Sales are null.
Using DateTimeFunctions	**Description**
AddDays([OrderDate], 30)	This expression adds the number of days equivalent to integer value 30 to the OrderDate value of the type DateTime.
DateDiffDay([DateStart], [DateEnd])	This expression counts the number of days between DateStart and DateEnd.
Using Math Functions	**Description**
Sign([Value])	This expression returns the integer value indicating the sign of the provided Value.
Exp([Value])	This expression calculates and returns the base of natural logarithms raised to the power Value.
Using String Functions	**Description**
Mid([Name],0,4)	This expression returns a string of specified number of 4 characters from the beginning of the specified [Name] string.
Remove([Name], 0, 3)	This expression removes specified number of 3 characters from the beginning of the string specified as 0.

Built-in Functions, Operators, and Constants

Expression Editor provides following built-in:

- **Functions**
- **Operators**
- **Constants**

Functions

Aggregate Functions	Syntax	Description
Average(value1....valueN)	Average()	Computes the average of a specified sequence of numbers (or enumerable).
Count(value1....valueN)	Count()	Gets the number of elements actually contained in a specified sequence
First(value1....valueN)	First()	Returns the first element of a specified sequence.
Last(value1....valueN)	Last()	Returns the last element of all the specified elements.
Max(value1....valueN)	Max()	Returns the maximum value in a specified sequence of numbers (or enumerable).
Min(value1....valueN)	Min()	Returns the minimum value in a specified sequence of numbers (or enumerable).
Sum(value1....valueN)	Sum()	Computes the sum of a specified sequence of numbers (or enumerable).
DateTime Functions	**Syntax**	**Description**
Now()	Now()	Gets a System.DateTime object that is set to the current date and time on this computer, expressed as the local time.
Today()	Today()	Gets the current date.
AddDays(DateTime, DaysCount)	AddDays(DateTime date, int days)	Returns a new System.DateTime that adds the specified number of days to the specified System.DateTime value.
AddHours(DateTime, HoursCount)	AddHours(DateTime date, int hours)	Returns a new System.DateTime that adds the specified number of hours to the specified System.DateTime value.
AddMilliSeconds(DateTime, MilliSecondsCount)	AddMilliSeconds(DateTime date, int milliSeconds)	Returns a new System.DateTime that adds the specified number of milliseconds to the specified System.DateTime value.
AddMinutes(DateTime, MinutesCount)	AddMinutes(DateTime date, int minutes	Returns a new System.DateTime that adds the specified number of minutes to the specified System.DateTime value.

	)	
AddMonths(DateTime, MonthsCount)	AddMonths(DateTime date, int months)	Returns a new System.DateTime that adds the specified number of months to the specified System.DateTime value.
AddSeconds(DateTime, SecondsCount)	AddSeconds(DateTime date, int seconds)	Returns a new System.DateTime that adds the specified number of seconds to the specified System.DateTime value.
AddTicks(DateTime, TicksCount)	AddTicks(DateTime date, int ticks)	Returns a new System.DateTime that adds the specified number of ticks to the specified System.DateTime value.
AddTimeSpan(DateTime, TimeSpan)	AddTimeSpan(DateTime date, TimeSpan timeSpan)	Returns a new System.DateTime that adds the specified number of System.TimeSpan to the specified System.DateTime value.
AddYears(DateTime, YearsCount)	AddYears(DateTime date, int years)	Returns a new System.DateTime that adds the specified number of years to the specified System.DateTime value.
DateDiffDay(startDate, endDate)	DateDiffDay(,)	Counts the number of day boundaries between two non-nullable dates.
DateDiffHour(startDate, endDate)	DateDiffHour(,)	Counts the number of hour boundaries between two non-nullable dates.
DateDiffMilliSecond(startDate, endDate)	DateDiffMilliSecond(,)	Counts the number of milliseconds boundaries between two non-nullable dates.
DateDiffMinute(startDate, endDate)	DateDiffMinute(,)	Counts the number of minutes boundaries between two non-nullable dates.
DateDiffSecond(startDate, endDate)	DateDiffSecond(,)	Counts the number of seconds boundaries between two non-nullable dates.
DateDiffTick(startDate, endDate)	DateDiffTick(,)	Counts the number of ticks boundaries between two non-nullable dates.
GetDate(DateTime)	GetDate()	Gets the date component of specified System.DateTime value.
GetDay(DateTime)	GetDay()	Gets the day of the month represented by the specified System.DateTime value.
GetDayOfWeek(DateTime)	GetDayOfWeek()	Gets the day of the week represented by the specified System.DateTime value.
GetDayOfYear(DateTime)	GetDayOfYear()	Gets the day of the year represented by the specified System.DateTime value.
GetHour(DateTime)	GetHour()	Gets the hour component of the specified System.DateTime value.

GetMilliSecond(DateTime)	GetMilliSecond()	Gets the milliSecond component of the specified System.DateTime value.
GetMinute(DateTime)	GetMinute()	Gets the minute component of the specified System.DateTime value.
GetMonth(DateTime)	GetMonth()	Gets the month component of the specified System.DateTime value.
GetSecond(DateTime)	GetSecond()	Gets the seconds component of the specified System.DateTime value.
GetTimeOfDay(DateTime)	GetTimeOfDay()	Gets the time of day for the specified System.DateTime value.
GetYear(DateTime)	GetYear()	Gets the year component of the specified System.DateTime value.
UtcNow()	UtcNow()	Gets a System.DateTime object that is set to the current date and time on this computer, expressed as the Coordinated Universal Time (UTC).
Logical Functions	**Syntax**	**Description**
IsNull(Value)	IsNull(object param)	Returns True if the specified Value is NULL.
Iif(condition, resultTrue,resultFalse)	Iif(bool condition, object expressionTrue, object expressionFalse)	Returns the evaluation of one of two expressions, depending on the condition.
Math Functions	**Syntax**	**Description**
Abs(Value)	Abs()	Returns the absolute value of a number.
Acos(Value)	Acos()	Returns the angle whose cosine is the specified number.
Asin(Value)	Asin()	Returns the angle whose sine is the specified number.
Atan(Value)	Atan()	Returns the angle whose tangent is the specified number.
Atan2(Value1, Value2)	Atan2(,)	Returns the angle whose tangent is the quotient of two specified numbers.
Ceiling(Value)	Ceiling()	Returns the smallest integral value that's greater than or equal to the specified decimal or double.
Cos(Value)	Cos()	Returns the cosine of the specified angle.
Cosh(Value)	Cosh()	Returns the hyperbolic cosine of the specified angle.
Exp(Value)	Exp()	Returns e (the base of natural logarithms) raised to the specified power.
Floor(Value)	Floor()	Returns the largest integer that's less than or equal to the specified decimal or double number.

Log(Value)	Log()	Returns the natural (base e) logarithm of a specified number or the logarithm of a specified number in a specified base.
Log(Value, Base)	Log(,)	Returns the natural (base e) logarithm of a specified number or the logarithm of a specified number in a specified base.
Log10(Value)	Log10()	Returns the base 10 logarithm of a specified number.
Pow(Value1, Value2)	Pow()	Returns a specified number raised to the specified power.
Rand(Value)	Rand()	Returns a nonnegative random number.
RandBetween(Value1, Value2)	RandBetween()	Returns a random number within a specified range.
Sign(Value)	Sign()	Returns an integer value indicating the sign of a number.
Sin(Value)	Sin()	Returns the sine of the specified angle.
Sinh(Value)	Sinh()	Returns the hyperbolic sine of the specified angle.
Sqrt(Value)	Sqrt()	Returns the square root of a specified number.
Tan(Value)	Tan()	Returns the tangent of the specified angle.
Tanh(Value)	Tanh()	Returns the hyperbolic tangent of the specified angle.
Convert Functions	**Syntax**	**Description**
CBool(string)	CBool()	Converts the specified string representation of a logical value to its System.Boolean equivalent, or throws an exception if the string is not equivalent to the value of System.Boolean.TrueString or System.Boolean.FalseString.
CByte(string)	CByte()	Converts the string representation of a number to its System.Byte equivalent.
CChar(string)	CChar()	Converts the value of the specified string to its equivalent Unicode character.
CDate(string)	CDate()	Converts the specified string representation of a date and time to its System.DateTime equivalent.
Cdbl(string)	CDbl()	Converts string representation of a number to its double-precision floating-point number equivalent.
CDec(string)	CDec()	Converts the string representation of a number to its System.Decimal equivalent.
CInt(string)	CInt()	Converts the string representation of a number to its 32-bit signed integer equivalent.
CLng(string)	CLng()	Converts the string representation of a number to its 64-bit signed integer equivalent.
CObj(value)	CObj()	Returns the specified elemsnt as System.Object.
CSByte(string)	CSByte()	Converts the string representation of a number to its 8-bit signed integer equivalent.

CShort(string)	CShort()	Converts the string representation of a number to its 16-bit signed integer equivalent.
CSng(string)	CSng()	Converts the string representation of a number to its single-precision floating-point number equivalent.
CStr(value)	CStr()	Tries to evaluate the specified expression and return the result as a string.
CType(value,type)	CType(,)	Returns an object of the specified type and whose value is equivalent to the specified object. value- the specified object for conversion. type- the type name of object to return.
CUint(value)	CUint()	Converts the string representation of a number to its 32-bit unsigned integer equivalent.
CULong(value)	CULong()	Converts the string representation of a number to its 64-bit unsigned integer equivalent.
CUShort(value)	CUShort()	Converts the string representation of a number to its 16-bit unsigned integer equivalent.
Text Functions	**Syntax**	**Description**
Replace(string,oldValue,newValue)	Replace(,,)	Replaces all occurrences of a specified string value with another string value.
Rset(value, length)	RSet(,)	Returns a new string of a specified length in which the end of the current string is padded with spaces or with a specified character.
Rset(value, length,char)	RSet(,,)	Returns a new string of a specified length in which the end of the current string is padded with spaces or with a specified character.
Remove(string,start)	Remove(,)	Deletes all characters from this instance, beginning at a specified position.
Remove(string,start,count)	Remove(,,)	Deletes all characters from this instance, beginning at a specified position.
LSet(string,length)	LSet(,)	Returns a new string of a specified length in which the beginning of the current string is padded with spaces or with a specified character.
LSet(string,length,char)	LSet(,,)	Returns a new string of a specified length in which the beginning of the current string is padded with spaces or with a specified character.
UCase(string)	UCase()	Returns a character expression with lowercase character data converted to uppercase.
LCase(string)	LCase()	Returns a character expression after converting uppercase character data to lowercase.
Insert(string,index,value)	Insert(,,)	Returns a new string in which a specified string is inserted at a specified index position in this instance.
Len(string)	Len()	Returns the number of characters of the specified string

		expression.
Trim(string)	Trim()	Removes the space character char(32) or other specified characters from the start or end of a string.
StartsWith(string,value)	StartsWith(,)	Determines whether the beginning of this string instance matches a specified string.
StrReverse(string)	StrReverse()	Returns the reverse order of a string value.
EndsWith(string,value)	EndsWith(,)	Determines whether the end of this string instance matches a specified string.
Contains(string,value)	Contains(,)	Returns a value indicating whether a specified substring occurs within this string.
InStr(string,value)	InStr(,)	Searches an expression for another expression and returns its starting position if found.
InStr(string,value,start)	InStr(,,)	Searches an expression for another expression and returns its starting position if found.
Mid(string,start)	Mid(,)	Returns a string that contains all the characters starting from a specified position in a string.
Mid(string,start,length)	Mid(,,)	Returns a string that contains a specified number of characters starting from a specified position in a string.
Chr(string)	Chr()	Converts an int ASCII code to a character.
Asc(string)	Acs()	Returns the ASCII code value of the leftmost character of a character expression.
Concat(string,value1.....valueN)	Concat(,)	Returns a string that is the result of concatenating two or more string values.

Back to Top

Operators

Operators	Syntax	Description
Plus	+	Sums two numbers.
Concat	&	Generates a string concatenation of two expressions.
Minus	-	Finds the difference between two numbers or indicates the negative value of a numeric expression.
Multiply	*	Multiplies two numbers.
Divide	/	Divides two numbers and returns a floating-point result.
Modulus	Mod	Divides two numbers and returns only the remainder.
Equal	=	Returns a Boolean value that indicates whether the left and right expressions are equal.
GreaterThan	>	Returns a Boolean value that indicates whether the left expression is greater than right expression.
LessThan	<	Returns a Boolean value that indicates whether the left expression is less than right expression.

NotEqual	<>	Returns a Boolean value that indicates whether the left and right expressions are not equal.
GreaterOrEqua	>=	Returns a Boolean value that indicates whether the left expression is greater than right expression or equal.
LessOrEqual	<=	Returns a Boolean value that indicates whether the left expression is less than right expression or equal.
And	And	Performs a logical conjunction on two expressions.
Or	Or	Performs a logical disjunction on two expressions.
Not	Not	Performs logical negation on an expression.

Back to Top

Constants

Constants	Syntax	Description
True	True	Returns True.
False	False	Returns False.
Nothing	Nothing	Returns Null.

Back to Top

Features

Expression Editor supports following features to enable users create applications using Expression Editor, and work with complex expressions.

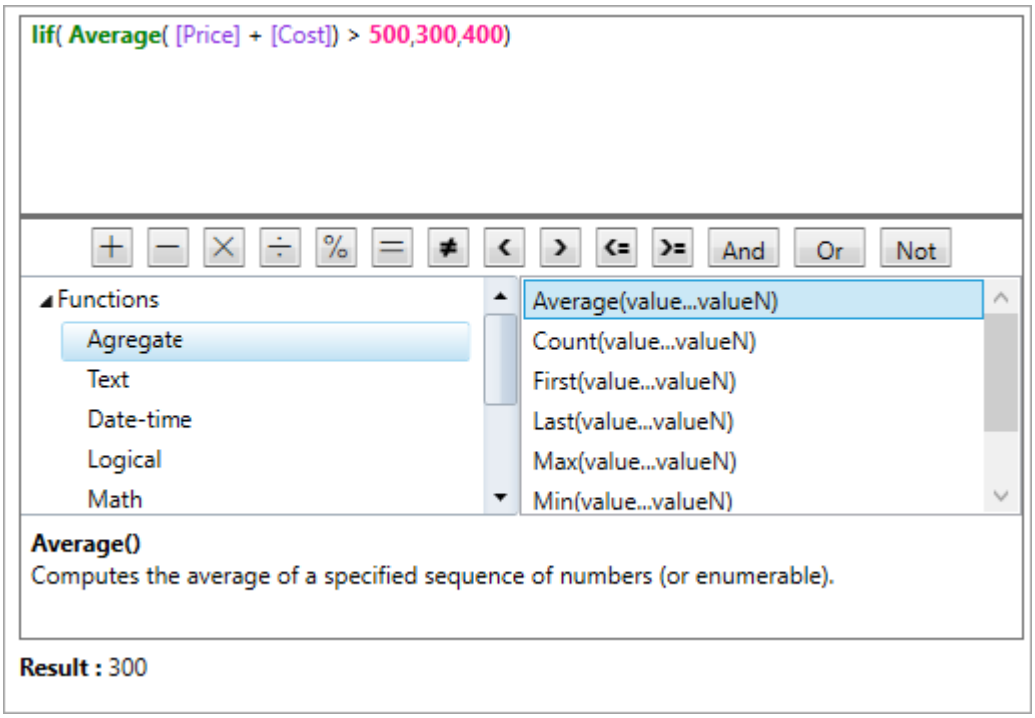
End-User Capabilities
: Learn about some features of Expression Editor, which enable the control to interact with end users and simplify creating and editing expressions.

Appearance and Styling
: Learn how to change the appearance of ExpressionEditor control.

End-User Capabilities

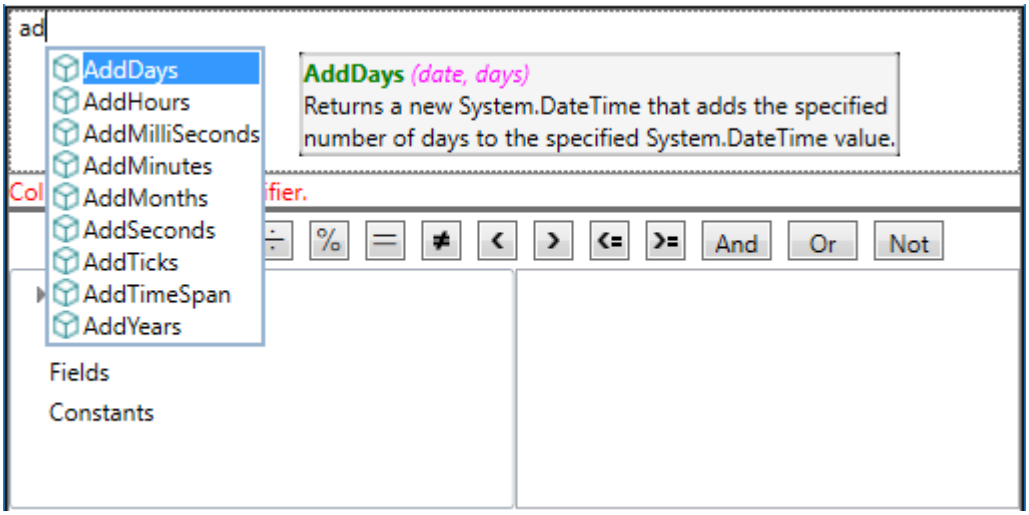
Expression Editor provides various features that enable end-users to interact with the control. These capabilities of the Expression Editor control are as follows:

Syntax Highlighting



Expression Editor uses different colors to display functions and fields, just like the SQL Query editor. As these items are differently colored, it increases the readability of expressions thereby making it easy to differentiate between functions, operators, and fields. SyntaxHighlighting property of C1ExpressionEditor class controls whether the items of expression should be highlighted. Note that operators and constants are not highlighted.

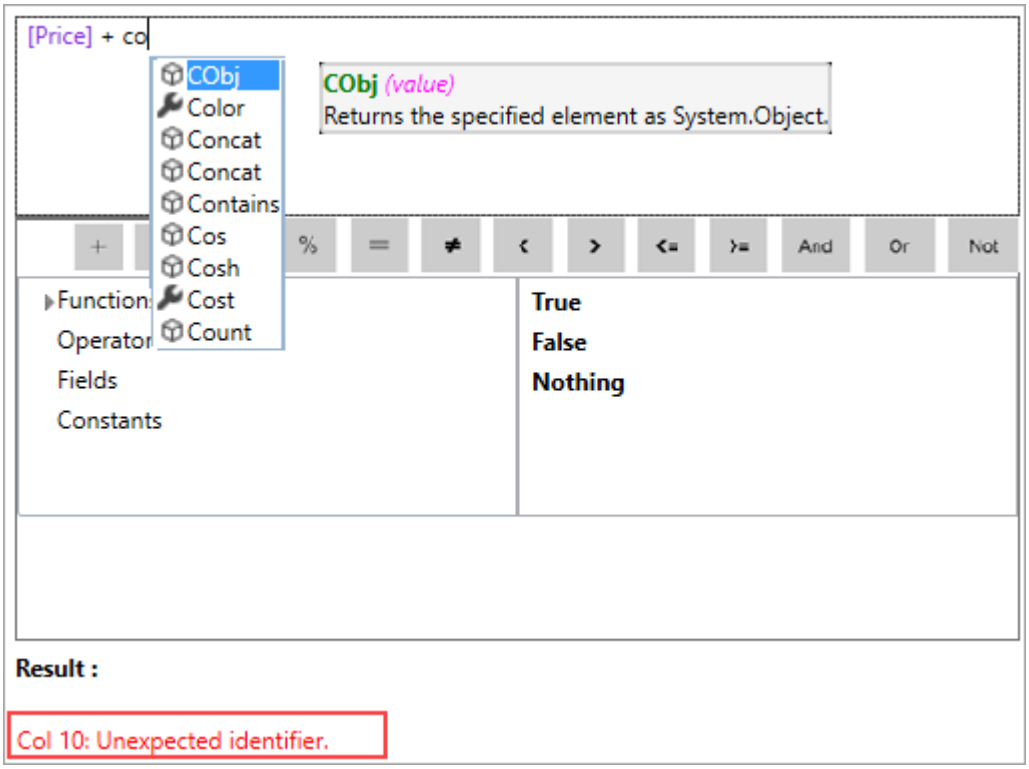
Smart Code Completion



When you type expressions, Expression Editor completes it by providing recommendations of possible functions or fields in a list based on what you type. This feature helps to complete expression quickly and reduces the chance of making typing errors.

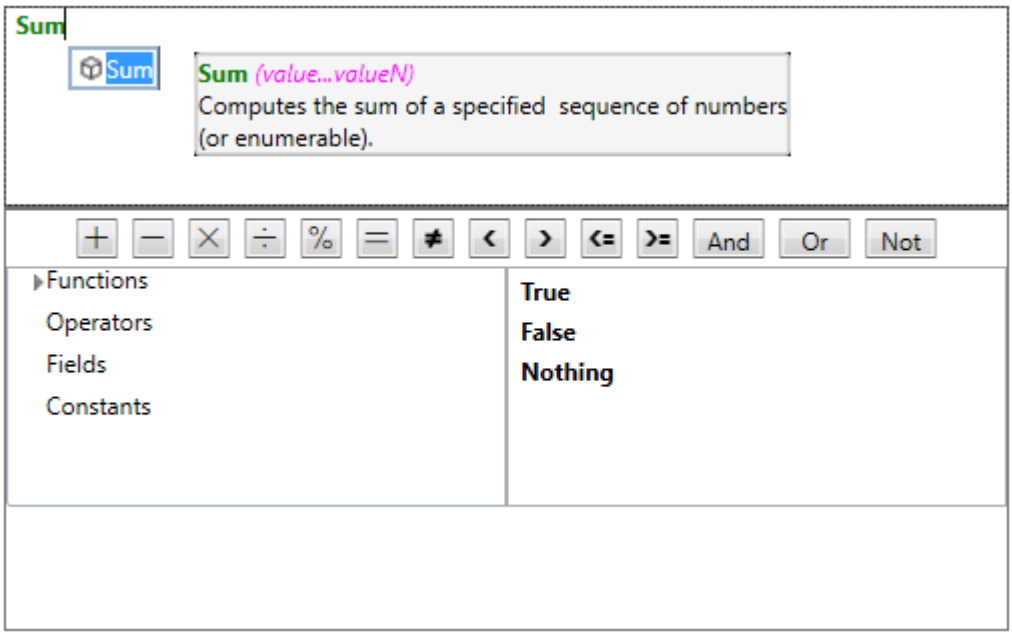
> ⚠ Note that the smart code completion functionality only responds to the keyboard input and does not provide any recommendations when you paste text in the editor.

Error Reporting



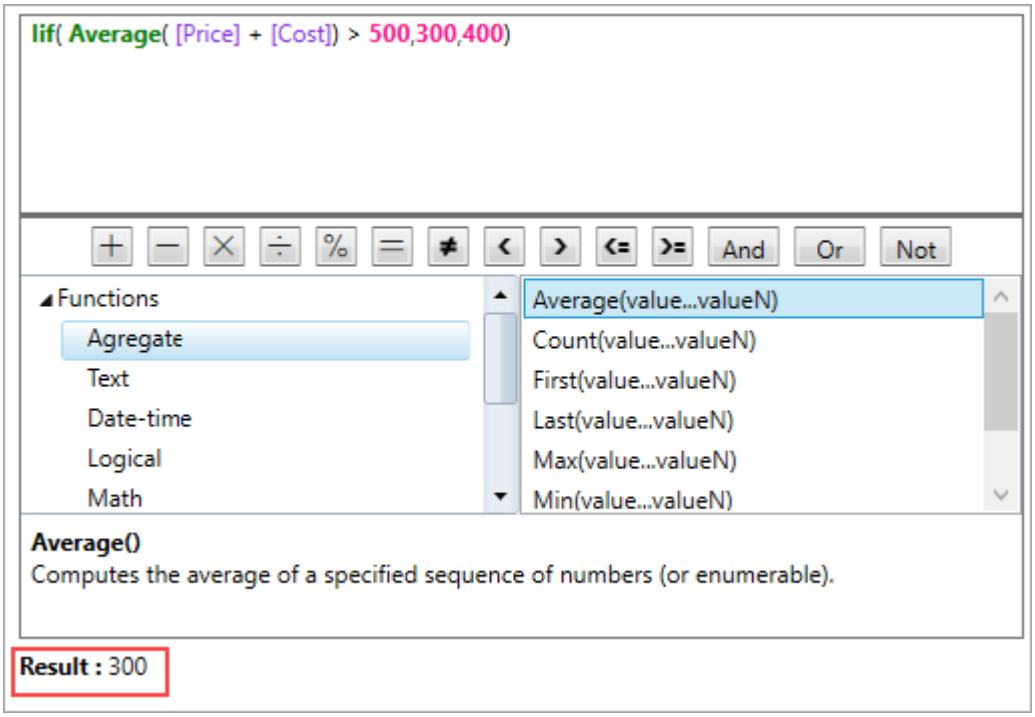
Expression Editor prevents syntax mistakes that might occur while writing expressions. The Editor immediately validates the typed expression and shows the error message as soon as it finds an incorrect syntax. IsValid property of C1ExpressionEditor class is used to control this behavior.

ToolTip Helper



Each time mouse is hovered over a function, Expression Editor displays a tooltip for that function which contains its description and syntax.

Result Preview



Expression Editor allows you to visualize final output and correct the possible errors, before finalizing the expression. While you write a valid expression, it displays the result in the preview. Upon writing an invalid expression, it displays error.

Back to Top

Appearance and Styling

Expression Editor supports customizing its appearance by changing the look of its elements.

C1ExpressionEditorPanel class provides following properties to modify the appearance of Expression Editor's panel.

Properties	Purpose
ButtonBackground	Modifies the background color of buttons inside the control.
ButtonForeground	Modifies the foreground color of buttons inside the control.
MoveOverBrush	Highlights the buttons when mouse is hovered on the buttons inside the control.
Background	Modifies the background color of the control.
Foreground	Modifies the foreground color of the control.
BorderThickness	Adjusts the thickness of the control's border.
BorderBrush	Modifies the color of the control's border.
FontFamily	Modifies the font of the text inside the control.
FontStyle	Modifies the style (Italic, Normal, or Oblique) in which the font is rendered.
FontSize	Modifies the font size of the text inside the control.
FontStretch	Specifies how the font of the control will expand or condense.
FontWeight	Modifies the thickness of control's font.

C1ExpressionEditor class provides following properties to modify the appearance of Expression Editor's text box.

Properties	Purpose
Background	Modifies the background color of the control.
Foreground	Modifies the foreground color of the control.
BorderThickness	Adjusts the thickness of the control's border.
BorderBrush	Modifies the color of the control's border.
FontFamily	Modifies the font of the text inside the control.
FontStyle	Modifies the style (Italic, Normal, or Oblique) in which the font is rendered.
FontSize	Modifies the font size of the text inside the control.
FontStretch	Specifies how the font of the control will expand or condense.
FontWeight	Modifies the thickness of control's font.

The following image illustrates customizing the appearance of Expression Editor.

Result : 16

The following code demonstrates changing the appearance of Expression Editor control.

- **C#**

```
ExpressionEditor.FontFamily = new FontFamily("Bradley Hand ITC");
ExpressionEditor.FontSize = 14;
ExpressionEditor.FontStyle = FontStyles.Italic;
ExpressionEditor.FontWeight = FontWeights.Light;

ExpressionPanel.FontFamily = new FontFamily("Bradley Hand ITC");
ExpressionPanel.ButtonBackground = new SolidColorBrush(Colors.BlanchedAlmond);
ExpressionPanel.ButtonForeground = new SolidColorBrush(Colors.DarkOliveGreen);
ExpressionPanel.MouseOverBrush = new SolidColorBrush(Colors.BurlyWood);
ExpressionPanel.FontSize = 14;
ExpressionPanel.FontStyle = FontStyles.Italic;
ExpressionPanel.FontWeight = FontWeights.Light;
```

Back to Top

Working with Expression Editor

The following topics explore advanced functionalities of the Expression Editor control. Expression Editor aids in creating simple as well as complex expressions. These expressions can then be used for shaping data in grid and chart controls.

Here we assume that the user is familiar with programming in Visual Studio, and has gone through the Quick Start topic.

Integration with FlexGrid
: Learn how to integrate Expression Editor control with FlexGrid, and implementing column calculation for unbound columns in FlexGrid using expressions.

Integration with FlexChart
: Learn how to integrate Expression Editor control with FlexChart, and modifying the chart's view by applying filters using expressions.

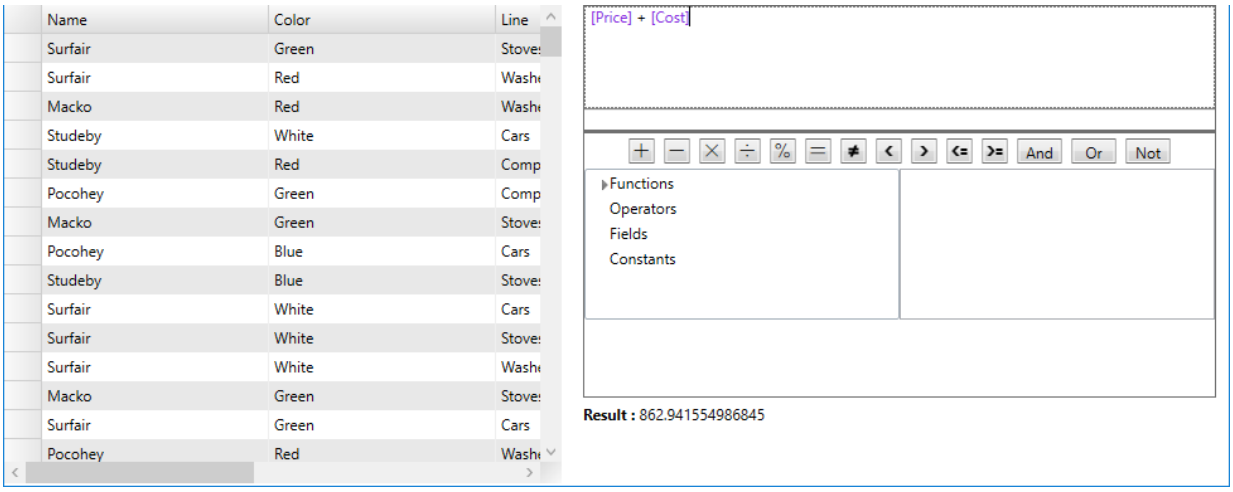
Integration with MSDataGrid
: Learn how to integrate Expression Editor control with MSDataGrid.

Integration with FlexGrid

The Expression Editor supports integration with FlexGrid control. Expression Editor, when integrated with grid, enables using expressions on grid and perform operations such as filtering, grouping, sorting, and column calculation over its data.

To integrate Expression Editor with the FlexGrid, you need to use ItemsSource property of C1FlexGrid class that gets a collection of objects to generate grid data. Once the grid is populated, data source of Expression editor can be bound to data source of FlexGrid using the DataSource property of C1ExpressionEditor class.

The following image exhibits Expression Editor integrated with FlexGrid control. The grid shows feature and cost information for different hardware products.



The following code demonstrates integrating FlexGrid with ExpressionEditor.

Bind the instance of C1CollectionView component of FlexGrid to ExpressionEditor through its DataSource property, as shown in the following code snippet.

- **C#**

```
ICollectionView View;
public MainWindow()
{
    InitializeComponent();

   _flexGrid.ItemsSource = Product.GetData(200);
```

```
    FlexExpEditor.editor.FlexExpressionEditor.DataSource = _flexGrid.CollectionView.CurrentItem;
}
```

For the detailed data refer the sample project **ExpressionEditorSamples** accompanying the installer.

Following topic discuss column calculation on flexGrid using expressions. However, to understand how expression editor helps in filtering, sorting, and grouping refer product sample accompanying the installer and the blog.

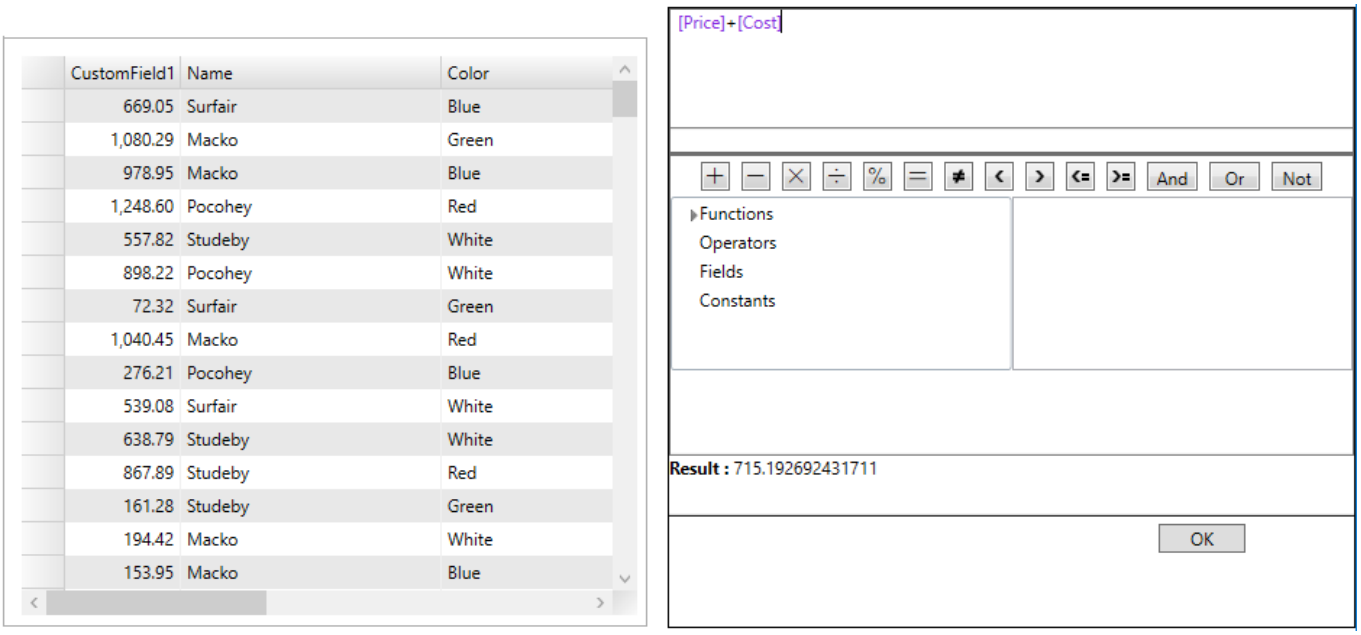
Back to Top

Column Calculation in FlexGrid

Expression Editor, when integrated with grid, allows calculating column data for unbound columns of FlexGrid.

To allow expressions to be used for generating data for unbound columns of FlexGrid, you can create a class that inherits C1FlexGrid class and implements iSupportExpressions interface. Then add Expression Editor on unbound columns of the new FlexGrid, where you need to display calculated values.

The following image exhibits FlexGrid control, demonstrating column calculation using expressions.



The following code demonstrates column calculation on FlexGrid columns through expressions.

1. Create a class that inherits C1FlexGrid and implements iSupportExpressions interface, as shown in the following code snippet.
   - **C#**

```
InitializeComponent();
List<Product> flexitems = Product.GetData(200);
List<Product> msGridItems = Product.GetData(200);
flexGrid.ItemsSource = flexitems;
```

2. Now, add expression editor on unbound columns of the new FlexGrid, as shown in the following code snippet.
   - **C#**

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

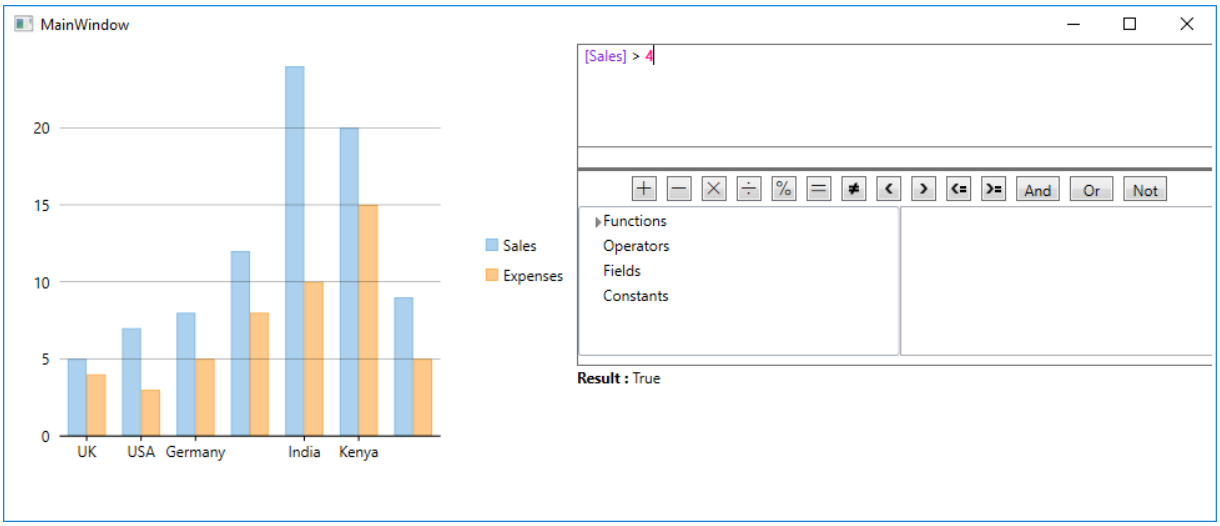
Back to Top

Integration with FlexChart

The Expression Editor supports integration with the FlexChart control. Expression Editor, when integrated with chart, enables manipulating the visualization of chart using expressions.

To integrate Expression Editor with FlexChart, you need to use ItemsSource property of C1FlexChart class that gets a collection of objects containing the series data, from **CollectionViewSource** object. After getting the series data, data source of FlexChart can be bound to data source of Expression Editor using its DataSource property.

The following image exhibits Expression Editor integrated with the FlexChart control. The chart plots country-wise Sales and Expenses for a retail store.



The following code demonstrates integrating FlexChart with ExpressionEditor control.

Create a C1CollectionView type field and bind the instance of C1CollectionView component of FlexChart to ExpressionEditor through its DataSource property.

- **C#**

```
DataSet ds;
ICollectionView View;
public MainWindow()
{
    InitializeComponent();
    FlexExpEditor.editor.OkClick += Editor_OkClick;
    FlexExpEditor.editor.CancelClick += Editor_CancelClick;

    CollectionViewSource view = new CollectionViewSource() { Source = DataCreator.CreateData() };
    View = view.View;
    flexChart.ItemsSource = View;
    flexChart.BindingX = "Country";


    FlexExpEditor.editor.FlexExpressionEditor.DataSource = View.CurrentItem;
```

For the detailed data refer the sample project **ExpressionEditorSamples** accompanying the installer.

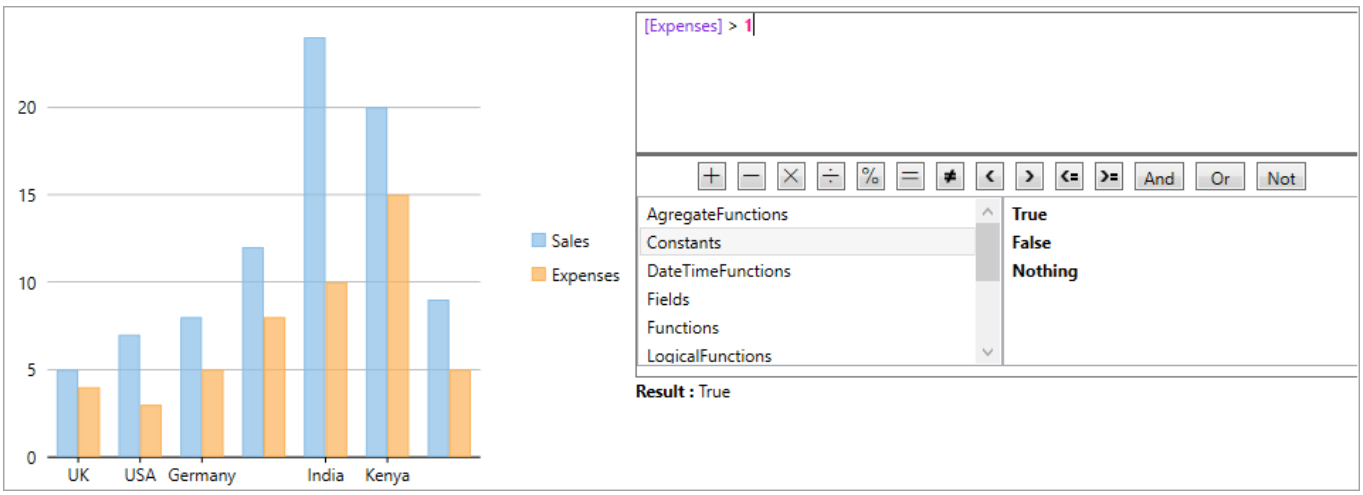
Back to Top

Filtering in FlexChart

Expression Editor, when integrated with FlexChart, enables filtering of its data items using expressions.

To filter the data items of FlexChart based on expressions entered in Expression Editor, you can use Filter predicate of C1CollectionView. Note that, C1Expression Editor's data source should be same as the FlexChart's data source.

The following image exhibits FlexChart control, demonstrating filtering using expression entered in Expression Editor control.



The following code demonstrates filtering the FlexChart's view through expression entered in ExpressionEditor control.

1. Bind the FlexChart control with Expression Editor control, as discussed in Integration with FlexChart.
2. Instantiate the Filter delegate, as shown in the following code snippet.
    - **C#**

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

3. Define the method for delegate, as shown in the following code snippet.
    - **C#**

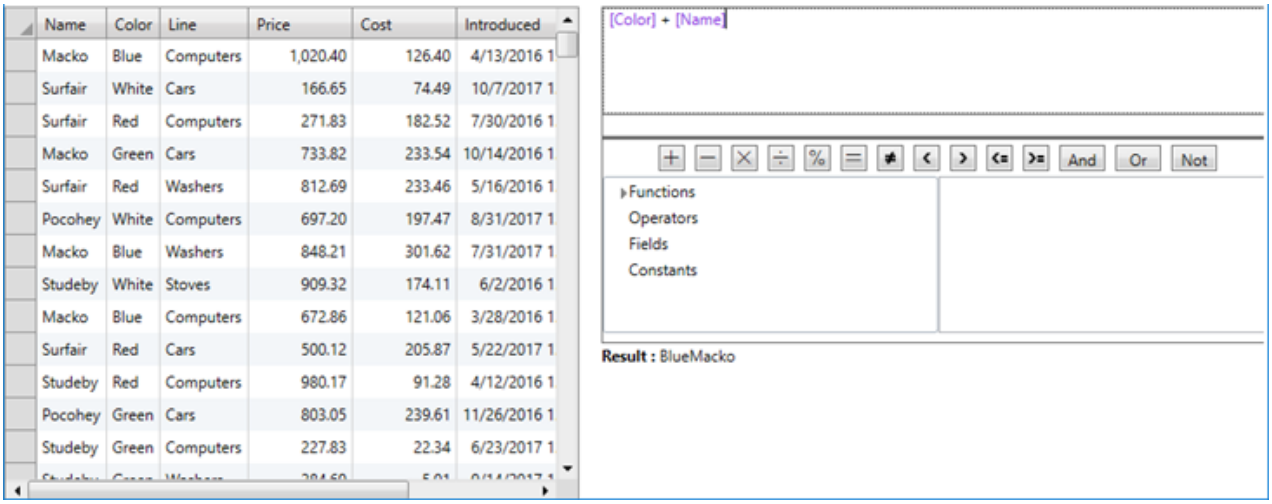
```
private bool Contains(object obj)
{
    C1ExpressionEditor _editor = new C1ExpressionEditor();
    _editor.Expression = FlexExpEditor.editor.FlexExpressionEditor.Expression;
    _editor.DataSource = View.CurrentItem;
    var value = _editor.Evaluate();
    var ret = true;
    if (value != null)
        Boolean.TryParse(value.ToString(), out ret);
    return ret;
}
```

Back to Top

Integration with MSDataGrid

Expression Editor, when integrated with grid, enables using expressions on grid and perform operations such as filtering, grouping, sorting, and column calculation over its data. To integrate Expression Editor with MSDataGrid, you need to use ItemsSource property of MSDataGrid that gets a collection of objects to generate grid data. Once the grid is populated, data source of Expression editor can be bound to data source of FlexGrid using the DataSource property of C1ExpressionEditor class.

The following image exhibits Expression Editor integrated with MSDataGrid control. The grid shows feature and cost information for different hardware products.



The following code demonstrates integrating MSDataGrid with ExpressionEditor.

Bind the instance of C1CollectionView component of MSDataGrid to ExpressionEditor through its DataSource property, as shown in the following code snippet.

- **C#**

```
ICollectionView View;
public MainWindow()
{
    InitializeComponent();

    CollectionViewSource view = new CollectionViewSource() { Source = Product.GetData(200) };
    View = view.View;
    msGrid.ItemsSource = View;
    FlexExpEditor.editor.FlexExpressionEditor.DataSource = View.CurrentItem;
}
```

For the detailed data refer the sample project **ExpressionEditorSamples** accompanying the installer.

Back to Top

Expression Editor Samples

Samples, which come with **C1Studio** installer, help you understand the product and its implementation better. Expression Editor sample is available in the installed folder:

Documents\ComponentOne Samples\WPF\Expression Editor\CS.

Documents\ComponentOne Samples\WPF\Expression Editor\VB.

Sample	Description
ExpressionEditorSamples	This sample demonstrates how to create application with Expression Editor control, bind it to an object and integrate it with FlexChart and FlexGrid.