DataColumn.Expression Property

.NET Framework 4

Gets or sets the expression used to filter rows, calculate the values in a column, or create an aggregate column.

Namespace: System.Data
Assembly: System.Data (in System.Data.dll)

Syntax

VB

```
'Declaration

Public Property Expression As String
        Get
        Set
```

Property Value

Type: System.String
An expression to calculate the value of a column, or create an aggregate column. The return type of an expression is determined by the DataType of the column.

Exceptions

Exception	Condition
ArgumentException	The AutoIncrement or Unique property is set to true.
FormatException	When you are using the CONVERT function, the expression evaluates to a string, but the string does not contain a representation that can be converted to the type parameter.
InvalidCastException	When you are using the CONVERT function, the requested cast is not possible. See the Conversion function in the following section for detailed information about possible casts.
ArgumentOutOfRangeException	When you use the SUBSTRING function, the start argument is out of range. -Or- When you use the SUBSTRING function, the length argument is out of range.

Exception	When you use the LEN function or the TRIM function, the expression does not evaluate to a string. This includes expressions that evaluate to Char.

Remarks

One use of the Expression property is to create calculated columns. For example, to calculate a tax value, the unit price is multiplied by a tax rate of a specific region. Because tax rates vary from region to region, it would be impossible to put a single tax rate in a column; instead, the value is calculated using the Expression property, as shown in the Visual Basic code in the following section:

DataSet1.Tables("Products").Columns("tax").Expression = "UnitPrice * 0.086"

A second use is to create an aggregate column. Similar to a calculated value, an aggregate performs an operation based on the complete set of rows in the DataTable. A simple example is to count the number of rows returned in the set. This is the method you would use to count the number of transactions completed by a particular salesperson, as shown in this Visual Basic code:

```
DataSet1.Tables("Orders").Columns("OrderCount").Expression = "Count(OrderID)"
```

Expression Syntax

When you create an expression, use the ColumnName property to refer to columns. For example, if the ColumnName for one column is "UnitPrice", and another "Quantity", the expression would be as follows:

"UnitPrice * Quantity"

Note

If a column is used in an expression, then the expression is said to have a dependency on that column. If a dependent column is renamed or removed, no exception is thrown. An exception will be thrown when the now-broken expression column is accessed.

When you create an expression for a filter, enclose strings with single quotation marks:

"LastName = 'Jones'"

If a column name contains any non-alphanumeric characters or starts with a digit or matches (case-insensitively) any of the following reserved words, it requires special handling, as described in the following paragraphs.

And

Between

Child

False

In

Is

Like

Not

Null

Or

Parent

True

If a column name satisfies one of the above conditions, it must be wrapped in either square brackets or the "`" (grave accent) quotes. For example, to use a column named "Column#" in an expression, you would write either "[Column#]":

Total * [Column#]

or "`Column#`":

Total * `Column#`

If the column name is enclosed in square brackets then any ']' and '\' characters (but not any other characters) in it must be escaped by prepending them with the backslash ("\") character. If the column name is enclosed in grave accent characters then it must not contain any grave accent characters in it. For example, a column named "Column[]\" would be written:

Total * [Column[\]\\]

or

Total * `Column[]\`

User-Defined Values

User-defined values may be used within expressions to be compared with column values. String values should be enclosed within single quotation marks (and each single quotation character in a string value has to be escaped by prepending it with another single quotation character). Date values should be enclosed within pound signs (#) or single quotes (') based on the data provider. Decimals and scientific notation are permissible for numeric values. For example:

"FirstName = 'John'"

"Price <= 50.00"

"Birthdate < #1/31/82#"

For columns that contain enumeration values, cast the value to an integer data type. For example:

"EnumColumn = 5"

Parsing Literal Expressions

All literal expressions must be expressed in the invariant culture locale. When DataSet parses and converts literal expressions, it always uses the invariant culture, not the current culture.

String literals are identified when there are single quotes surrounding the value. For example:

'John'

Boolean literals are true and false; they are not quoted in expressions.

Integer literals [+-]?[0-9]+ are treated as System.Int32, System.Int64 or System.Double.

System.Double can lose precision depending on how large the number is. For example, if the number in the literal is 2147483650, DataSet will first attempt to parse the number as an Int32. This will not succeed because the number is too large. In this case DataSet will parse the number as an Int64, which will succeed. If the literal was a number larger than the maximum value of an Int64, DataSet will parse the literal using Double.

Real literals using scientific notation, such as 4.42372E-30, are parsed using System.Double.

Real literals without scientific notation, but with a decimal point, are treated as System.Decimal. If the number exceeds the maximum or minimum values supported by System.Decimal, then it is parsed as a System.Double. For example:

142526.144524 will be converted to a Decimal.

345262.78036719560925667 will be treated as a Double.

Operators

Concatenation is allowed using Boolean AND, OR, and NOT operators. You can use parentheses to group clauses and force precedence. The AND operator has precedence over other operators. For example:

(LastName = 'Smith' OR LastName = 'Jones') AND FirstName = 'John'

When you create comparison expressions, the following operators are allowed:

<

>

<=

>=

<>

=

IN

LIKE

The following arithmetic operators are also supported in expressions:

+ (addition)

- (subtraction)

* (multiplication)

/ (division)

% (modulus)

String Operators

To concatenate a string, use the + character. The value of the CaseSensitive property of the DataSet class determines whether string comparisons are case-sensitive. However, you can override that value with the CaseSensitive property of the DataTable class.

Wildcard Characters

Both the * and % can be used interchangeably for wildcard characters in a LIKE comparison. If the string in a LIKE clause contains a * or %, those characters should be enclosed in brackets ([]). If a bracket is in the clause, each bracket character should be enclosed in brackets (for example [[] or []]). A wildcard is allowed at the start and end of a pattern, or at the end of a pattern, or at the start of a pattern. For example:

"ItemName LIKE '*product*'"

"ItemName LIKE '*product'"

"ItemName LIKE 'product*'"

Wildcard characters are not allowed in the middle of a string. For example, 'te*xt' is not allowed.

Parent/Child Relation Referencing

A parent table may be referenced in an expression by prepending the column name with Parent. For example, the Parent.Price references the parent table's column named Price.

When a child has more than one parent row, use Parent(RelationName).ColumnName. For example, the Parent(RelationName).Price references the parent table’s column named Price via the relation.

A column in a child table may be referenced in an expression by prepending the column name with Child. However, because child relationships may return multiple rows, you must include the reference to the child column in an aggregate function. For example, Sum(Child.Price) would return the sum of the column named Price in the child table.

If a table has more than one child, the syntax is: Child(RelationName). For example, if a table has two child tables named Customers and Orders, and the DataRelation object is named Customers2Orders, the reference would be as follows:

Avg(Child(Customers2Orders).Quantity)

Aggregates

The following aggregate types are supported:

Sum (Sum)

Avg (Average)

Min (Minimum)

Max (Maximum)

Count (Count)

StDev (Statistical standard deviation)

Var (Statistical variance).

Aggregates are ordinarily performed along relationships. Create an aggregate expression by using one of the functions listed earlier and a child table column as detailed in Parent/Child Relation Referencing that was discussed earlier. For example:

Avg(Child.Price)

Avg(Child(Orders2Details).Price)

An aggregate can also be performed on a single table. For example, to create a summary of figures in a column named "Price":

Sum(Price)

Note

If you use a single table to create an aggregate, there would be no group-by functionality. Instead, all rows would display the same value in the column.

If a table has no rows, the aggregate functions will return Nothing.

Data types can always be determined by examining the DataType property of a column. You can also convert data types using the Convert function, shown in the following section.

Functions

The following functions are also supported:

CONVERT

Description	Converts particular expression to a specified .NET Framework Type.
Syntax	Convert(*expression*, *type*)
Arguments	*expression* -- The expression to convert. *type* -- The .NET Framework type to which the value will be converted.

Example: myDataColumn.Expression="Convert(total, 'System.Int32')"

All conversions are valid with the following exceptions: Boolean can be coerced to and from Byte, SByte, Int16, Int32, Int64, UInt16, UInt32, UInt64, String and itself only. Char can be coerced to and from Int32, UInt32, String, and itself only. DateTime can be coerced to and from String and itself only. TimeSpan can be coerced to and from String and itself only.

LEN

Description	Gets the length of a string
Syntax	LEN(*expression*)
Arguments	*expression* -- The string to be evaluated.

Example: myDataColumn.Expression="Len(ItemName)"

ISNULL

Description	Checks an expression and either returns the checked expression or a replacement value.
Syntax	ISNULL(*expression*, *replacementvalue*)
Arguments	*expression* -- The expression to check. *replacementvalue* -- If expression is Nothing, *replacementvalue* is returned.

Example: myDataColumn.Expression="IsNull(price, -1)"

IIF

Description	Gets one of two values depending on the result of a logical expression.
Syntax	IIF(*expr*, *truepart*, *falsepart*)
Arguments	*expr* -- The expression to evaluate. *truepart* -- The value to return if the expression is true. *falsepart* -- The value to return if the expression is false.

Example: myDataColumn.Expression = "IIF(total>1000, 'expensive', 'dear')

TRIM

Description	Removes all leading and trailing blank characters like \r, \n, \t, ' '
Syntax	TRIM(*expression*)
Arguments	*expression* -- The expression to trim.

SUBSTRING

Description	Gets a sub-string of a specified length, starting at a specified point in the string.

Syntax	SUBSTRING(*expression*, *start*, *length*)
Arguments	*expression* -- The source string for the substring. *start* -- Integer that specifies where the substring starts. *length* -- Integer that specifies the length of the substring.

Example: myDataColumn.Expression = "SUBSTRING(phone, 7, 8)"

Note
You can reset the Expression property by assigning it a null value or empty string. If a default value is set on the expression column, all previously filled rows are assigned the default value after the Expression property is reset.

Examples

The following example creates three columns in a DataTable. The second and third columns contain expressions; the second calculates tax using a variable tax rate, and the third adds the result of the calculation to the value of the first column. The resulting table is displayed in a DataGrid control.

VB

```
Private Sub CalcColumns()
     Dim rate As Single = .0862
     dim table as DataTable = New DataTable

     ' Create the first column.
     Dim priceColumn As DataColumn = New DataColumn
     With priceColumn
         .DataType = System.Type.GetType("System.Decimal")
         .ColumnName = "price"
         .DefaultValue = 50
     End With

     ' Create the second, calculated, column.
     Dim taxColumn As DataColumn = New DataColumn
     With taxColumn
         .DataType = System.Type.GetType("System.Decimal")
         .ColumnName = "tax"
         .Expression = "price * 0.0862"
     End With

    ' Create third column
     Dim totalColumn As DataColumn = New DataColumn
     With totalColumn
         .DataType = System.Type.GetType("System.Decimal")
         .ColumnName = "total"
         .Expression = "price + tax"
     End With

     ' Add columns to DataTable
```

```
        With table.Columns
            .Add(priceColumn)
            .Add(taxColumn)
            .Add(totalColumn)
        End With

        Dim row As DataRow= table.NewRow
        table.Rows.Add(row)
        Dim view As New DataView
        view.Table = table
        DataGrid1.DataSource = view
    End Sub
```

Version Information

.NET Framework

Supported in: 4, 3.5, 3.0, 2.0, 1.1, 1.0

.NET Framework Client Profile

Supported in: 4, 3.5 SP1

Platforms

Windows 7, Windows Vista SP1 or later, Windows XP SP3, Windows XP SP2 x64 Edition, Windows Server 2008 (Server Core Role not supported), Windows Server 2008 R2 (Server Core Role not supported), Windows Server 2003 SP2

The .NET Framework does not support all versions of every platform. For a list of the supported versions, see .NET Framework System Requirements.

See Also

Reference

DataColumn Class
System.Data Namespace

Other Resources

DataSets, DataTables, and DataViews (ADO.NET)

Community Content

test

test

4/11/2011
sumituppal76