Inhaltsverzeichnis

How to filter your row data

This document briefly explains how to use the data filter syntax.

Column names

If a column name contains any of these special characters **~ ( ) # \ / = > < + - * % & | ^ ' „ [ ]**, you must enclose the column name within square brackets [ ]. If a column name contains right bracket] or backslash \, escape it with backslash (\] or \\).

```
"id = 10";      // no special character in column name "id"
"$id = 10";     // no special character in column name "$id"
"[#id] = 10";   // special character "#" in column name "#id"
"[[id\]] = 10"; // special characters in column name "[id]"
```

Literals

String values are enclosed within single quotes **' '**. If the string contains single quote ', the quote must be doubled.

```
dataView.RowFilter = "Name = 'John'"        // string value
dataView.RowFilter = "Name = 'John ''A'''"  // string with single quotes "John 'A'"
dataView.RowFilter = String.Format("Name = '{0}'", "John 'A'".Replace("'", "''"));
```

Number values are not enclosed within any characters. The values should be the same as is the result of int.ToString() or float.ToString() method for invariant or English culture.

```
dataView.RowFilter = "Year = 2008"          // integer value
dataView.RowFilter = "Price = 1199.9"       // float value
dataView.RowFilter = String.Format(CultureInfo.InvariantCulture.NumberFormat,
                     "Price = {0}", 1199.9f);
```

Date values are enclosed within sharp characters # #. The date format is the same as is the result of DateTime.ToString() method for invariant or English culture.

```
dataView.RowFilter = "Date = #12/31/2008#"          // date value (time is 00:00:00)
dataView.RowFilter = "Date = #2008-12-31#"          // also this format is supported
dataView.RowFilter = "Date = #12/31/2008 16:44:58#" // date and time value
dataView.RowFilter = String.Format(CultureInfo.InvariantCulture.DateTimeFormat,
                     "Date = #{0}#", new DateTime(2008, 12, 31, 16, 44, 58));
```

Alternatively you can enclose all **values** within single quotes **' '**. It means you can use string values for numbers or date time values. In this case the current culture is used to convert the string to the specific value.

```
dataView.RowFilter = "Date = '12/31/2008 16:44:58'" // if current culture is English
dataView.RowFilter = "Date = '31.12.2008 16:44:58'" // if current culture is German
dataView.RowFilter = "Price = '1199.90'"            // if current culture is English
dataView.RowFilter = "Price = '1199,90'"            // if current culture is German
```

Comparison operators

Equal, not equal, less, greater operators are used to include only values that suit to a comparison expression. You can use these operators **= <> < ⇐ > >=** .

Note: **String comparison is culture-sensitive**, it uses CultureInfo from DataTable.Locale property of related table (dataView.Table.Locale). If the property is not explicitly set, its default value is DataSet.Locale (and its default value is current system culture Thread.CurrentThread.CurrentCulture).

```
dataView.RowFilter = "Num = 10"             // number is equal to 10
dataView.RowFilter = "Date < #1/1/2008#"    // date is less than 1/1/2008
dataView.RowFilter = "Name <> 'John'"       // string is not equal to 'John'
dataView.RowFilter = "Name >= 'Jo'"         // string comparison
```

Operator IN is used to include only values from the list. You can use the operator for all data types, such as numbers or strings.

```
dataView.RowFilter = "Id IN (1, 2, 3)"                    // integer values
dataView.RowFilter = "Price IN (1.0, 9.9, 11.5)"          // float values
dataView.RowFilter = "Name IN ('John', 'Jim', 'Tom')"     // string values
dataView.RowFilter = "Date IN (#12/31/2008#, #1/1/2009#)" // date time values
dataView.RowFilter = "Id NOT IN (1, 2, 3)"  // values not from the list
```

Operator LIKE is used to include only values that match a pattern with wildcards. **Wildcard** character is * or %, it can be at the beginning of a pattern '*value', at the end 'value*', or at both '*value*'. Wildcard in the middle of a patern 'va*lue' is not allowed.

```
dataView.RowFilter = "Name LIKE 'j*'"       // values that start with 'j'
dataView.RowFilter = "Name LIKE '%jo%'"     // values that contain 'jo'
dataView.RowFilter = "Name NOT LIKE 'j*'"   // values that don't start with 'j'
```

If a pattern in a LIKE clause contains any of these special characters * % [ ], those characters must be escaped in brackets [ ] like this [*], [%],] or [.

```
dataView.RowFilter = "Name LIKE '[*]*'"     // values that starts with '*'
dataView.RowFilter = "Name LIKE '[[]*'"     // values that starts with '['
```

The following method escapes a text value for usage in a LIKE clause.

```
public static string EscapeLikeValue(string valueWithoutWildcards)
{
  StringBuilder sb = new StringBuilder();
  for (int i = 0; i < valueWithoutWildcards.Length; i++)
  {
    char c = valueWithoutWildcards[i];
    if (c == '*' || c == '%' || c == '[' || c == ']')
      sb.Append("[").Append(c).Append("]");
    else if (c == '\'')
      sb.Append("''");
    else
      sb.Append(c);
  }
  return sb.ToString();
}
```

```
// select all that starts with the value string (in this case with "*")
string value = "*";
// the dataView.RowFilter will be: "Name LIKE '[*]*'"
dataView.RowFilter = String.Format("Name LIKE '{0}*'", EscapeLikeValue(value));
```

Boolean operators

Boolean operators AND, OR and NOT are used to concatenate expressions. Operator NOT has precedence over AND operator and it has precedence over OR operator.

```
// operator AND has precedence over OR operator, parenthesis are needed
dataView.RowFilter = "City = 'Tokyo' AND (Age < 20 OR Age > 60)";
// following examples do the same
dataView.RowFilter = "City <> 'Tokyo' AND City <> 'Paris'";
dataView.RowFilter = "NOT City = 'Tokyo' AND NOT City = 'Paris'";
dataView.RowFilter = "NOT (City = 'Tokyo' OR City = 'Paris')";
dataView.RowFilter = "City NOT IN ('Tokyo', 'Paris')";
```

Arithmetic and string operators

Arithmetic operators are addition +, subtraction -, multiplication *, division / and modulus %.

```
dataView.RowFilter = "MotherAge - Age < 20";   // people with young mother
dataView.RowFilter = "Age % 10 = 0";           // people with decennial birthday
```

There is also one **string** operator concatenation **+**.

Parent-Child Relation Referencing

A **parent table** can be referenced in an expression using parent column name with *Parent*. prefix. A column in a **child table** can be referenced using child column name with *Child*. prefix.

The reference to the child column must be in an **aggregate function** because child relationships may return multiple rows. For example expression *SUM(Child.Price)* returns sum of all prices in child table related to the row in parent table.

If a table has more than one child relation, the prefix must contain relation name. For example expression *Child(OrdersToItemsRelation).Price* references to column Price in child table using relation named OrdersToItemsRelation.

Aggregate Functions

There are supported following aggregate functions SUM, COUNT, MIN, MAX, AVG (average), STDEV (statistical standard deviation) and VAR (statistical variance). This example shows aggregate function performed on a single table.

```
// select people with above-average salary
dataView.RowFilter = "Salary > AVG(Salary)";
```

Following example shows aggregate functions performed on two tables which have parent-child relation. Suppose there are tables Orders and Items with the parent-child relation.

```
// select orders which have more than 5 items
dataView.RowFilter = "COUNT(Child.IdOrder) > 5";
// select orders which total price (sum of items prices) is greater or
equal $500
dataView.RowFilter = "SUM(Child.Price) >= 500";
```

Functions

There are also supported following functions. Detailed description can be found here DataColumn.Expression.

- **CONVERT** – converts particular expression to a specified .NET Framework type
- **LEN** – gets the length of a string
- **ISNULL** – checks an expression and either returns the checked expression or a replacement value
- **IIF** – gets one of two values depending on the result of a logical expression
- **TRIM** – removes all leading and trailing blank characters like \r, \n, \t, , ‘
- **SUBSTRING** – gets a sub-string of a specified length, starting at a specified point in the string

```

```