RowFilter Syntax

When creating 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:
"UnitPrice * Quantity"
When creating an expression for a filter, enclose strings with single quotes:
"LastName = 'Jones'"
The following characters are special characters and must be escaped, as explained below, if they are used in a column name:
\n (newline)
\t (tab)
\r (carriage return)
~
(
)
#
\
/
=
>
<
+
-
*
%
&
|
^
'
"
[
]
If a column name contains one of the above characters, the name must be wrapped in brackets. For example to use a column named "Column#" in an expression, you would write "[Column#]":
Total * [Column#]
Because brackets are special characters, you must use a slash ("\") to escape the bracket, if it is part of a column name. For example, a column named "Column[]" would be written:
Total * [Column[\]]
(Only the second bracket must be escaped.)
USER-DEFINED VALUES
User-defined values may be used within expressions to be compared against column values. String values should be enclosed within single quotes. Date values should be enclosed within pound signs (#). 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"
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 creating 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. Whether string comparisons are case-sensitive or not is determined by the value of the DataSet class's CaseSensitive property. However, you can override that value with the **DataTable** class's CaseSensitive property.
WILDCARD CHARACTERS
Both the * and % can be used interchangeably for wildcards in a LIKE comparison. If the string in a LIKE clause contains a * or %, those characters should be escaped in brackets ([]). If a bracket is in the clause, the bracket characters should be escaped in brackets (for example [[] or []]). A wildcard is allowed at the beginning and end of a pattern, or at the end of a pattern, or at the beginning of a pattern. For example:
"ItemName LIKE '*product*'"
"ItemName LIKE '*product'"
"ItemName LIKE 'product*'"
Wildcards 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**.
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:
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 usually performed along relationships. Create an aggregate expression by using one of the functions listed above and a child table column as detailed in PARENT/CHILD RELATION REFERENCING above. 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 a null reference (**Nothing** in Visual Basic).
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 below.
FUNCTIONS
The following functions are also supported:
CONVERT

Description	Converts given 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 a null reference (**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 begins. *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.

Example

[Visual Basic, C#, C++] The following example creates three coumns 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.

```
[C#]
private void CalcColumns(){
    DataColumn cPrice;
    DataColumn cTax;
    DataColumn cTotal;
    DataTable myTable = new DataTable ();

    // Create the first column.
    cPrice = new DataColumn();
    cPrice.DataType = System.Type.GetType("System.Decimal");
    cPrice.ColumnName = "price";
    cPrice.DefaultValue = 50;

    // Create the second, calculated, column.
    cTax = new DataColumn();
    cTax.DataType = System.Type.GetType("System.Decimal");
    cTax.ColumnName = "tax";
    cTax.Expression = "price * 0.0862";

    // Create third column.
    cTotal = new DataColumn();
    cTotal.DataType = System.Type.GetType("System.Decimal");
    cTotal.ColumnName = "total";
    cTotal.Expression = "price + tax";

    // Add columns to DataTable.
    myTable.Columns.Add(cPrice);
    myTable.Columns.Add(cTax);
    myTable.Columns.Add(cTotal);
    DataRow myRow;
    myRow = myTable.NewRow();
    myTable.Rows.Add(myRow);
    DataView myView = new DataView(myTable);
    dataGrid1.DataSource = myView;
}
```