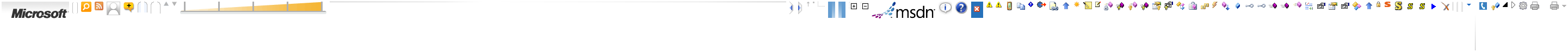
**Expression Examples (Report Builder and SSRS)**

**SQL Server 2012**

[Other Versions](javascript:;)



1 out of 2 rated this helpful [Rate this topic](http://msdn.microsoft.com/en-us/library/ms157328.aspx#feedback)

Expressions are used frequently in reports to control content and report appearance. Expressions are written in Microsoft Visual Basic, and can use built-in functions, custom code, report and group variables, and user-defined variables. Expressions begin with an equal sign (=). For more information about the expression editor and the types of references that you can include, see [Expression Uses in Reports (Report Builder and SSRS)](http://msdn.microsoft.com/en-us/library/ms345237.aspx), and [Add an Expression (Report Builder and SSRS)](http://msdn.microsoft.com/en-us/library/dd255254.aspx).

This topic provides examples of expressions that can be used for common tasks in a report.

* [Visual Basic Functions](http://msdn.microsoft.com/en-us/library/ms157328.aspx#VisualBasicFunctions)   Examples for date, string, conversion and conditional Visual Basic functions.
* [Report Functions](http://msdn.microsoft.com/en-us/library/ms157328.aspx#ReportFunctions)   Examples for aggregates and other built-in report functions.
* [Appearance of Report Data](http://msdn.microsoft.com/en-us/library/ms157328.aspx#AppearanceofReportData)   Examples for changing the appearance of a report.
* [Properties](http://msdn.microsoft.com/en-us/library/ms157328.aspx#Properties)   Examples for setting report item properties to control format or visibility.
* [Parameters](http://msdn.microsoft.com/en-us/library/ms157328.aspx#Parameters)   Examples for using parameters in an expression.
* [Custom Code](http://msdn.microsoft.com/en-us/library/ms157328.aspx#CustomCode)   Examples of embedded custom code.

For expression examples for specific uses, see the following topics:

* [Group Expression Examples (Report Builder and SSRS)](http://msdn.microsoft.com/en-us/library/bb630426.aspx)
* [Filter Equation Examples (Report Builder and SSRS)](http://msdn.microsoft.com/en-us/library/cc627464.aspx)
* [Single-Value and Multivalue Parameters (Report Builder and SSRS)](http://msdn.microsoft.com/en-us/library/aa337292.aspx)
* [Report and Group Variables Collections References (Report Builder and SSRS)](http://msdn.microsoft.com/en-us/library/dd255208.aspx)

For more information about simple and complex expressions, where you can use expressions, and the types of references that you can include in an expression, see topics under [Expressions (Report Builder and SSRS)](http://msdn.microsoft.com/en-us/library/dd220516.aspx). For more information about the context in which expressions are evaluated for calculating aggregates, see [Expression Scope for Totals, Aggregates, and Built-in Collections (Report Builder and SSRS)](http://msdn.microsoft.com/en-us/library/dd255256.aspx).

To learn how to write expressions that use many of the functions and operators also used by expression examples in this topic, but in the context of writing a report, see [Tutorial: Introducing Expressions](http://msdn.microsoft.com/en-us/library/gg720213.aspx).

The expression editor includes a hierarchical view of built-in functions. When you select the function, a code example appears in the Values pane. For more information, see the [Expression Dialog Box](http://msdn.microsoft.com/en-us/library/ms189719.aspx) or [Expression Dialog Box (Report Builder)](http://msdn.microsoft.com/en-us/library/dd220634.aspx).

If you are using Report Model Query Designer to design a dataset query that uses a report model as a data source, you will use formulas instead of expressions. These formulas help specify the report data by using custom calculations that are integrated into the query that specifies which data to return from the report model data source. For more information, see [Formulas in Report Model Queries (Report Builder and SSRS)](http://msdn.microsoft.com/en-us/library/ee210538.aspx).

|  |
| --- |
| **Note Note** |
| You can create and modify report definitions (.rdl) in Report Builder and in Report Designer in SQL Server Data Tools. Each authoring environment provides different ways to create, open, and save reports and related items. For more information, see Designing Reports in Report Designer and Report Builder (SSRS) on the Web at microsoft.com. |

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Functions**](javascript:void(0))

Many expressions in a report contain functions. You can format data, apply logic, and access report metadata using these functions. You can write expressions that use functions from the Microsoft Visual Basic run-time library, and from the [System.Convert](http://msdn.microsoft.com/en-us/library/system.convert.aspx) and [System.Math](http://msdn.microsoft.com/en-us/library/system.math.aspx) namespaces. You can add references to functions from other assemblies or custom code. You can also use classes from the Microsoft .NET Framework, including[System.Text.RegularExpressions](http://msdn.microsoft.com/en-us/library/system.text.regularexpressions.aspx).

**Visual Basic Functions**

You can use Visual Basic functions to manipulate the data that is displayed in text boxes or that is used for parameters, properties, or other areas of the report. This section provides examples demonstrating some of these functions. For more information, see [Visual Basic Runtime Library Members](http://go.microsoft.com/fwlink/?LinkId=198941) on MSDN.

The .NET Framework provides many custom format options, for example, for specific date formats. For more information, see [Formatting Types](http://go.microsoft.com/fwlink/?LinkId=112024) on MSDN.

**Math Functions**

* The **Round** function is useful to round numbers to the nearest integer. The following expression rounds a 1.3 to 1:
* = Round(1.3)

You can also write an expression to round a value to a multiple that you specify, similar to the **MRound**function in Excel. Multiply the value by a factor that creates an integer, round the number, and then divide by the same factor. For example, to round 1.3 to the nearest multiple of .2 (1.4), use the following expression:

= Round(1.3\*5)/5

**Date Functions**

* The **Today** function provides the current date. This expression can be used in a text box to display the date on the report, or in a parameter to filter data based on the current date.
* =Today()
* The **DateAdd** function is useful for supplying a range of dates based on a single parameter. The following expression provides a date that is six months after the date from a parameter named *StartDate*.
* =DateAdd(DateInterval.Month, 6, Parameters!StartDate.Value)
* The **Year** function displays the year for a particular date. You can use this to group dates together or to display the year as a label for a set of dates. This expression provides the year for a given group of sales order dates. The **Month** function and other functions can also be used to manipulate dates. For more information, see the Visual Basic documentation.
* =Year(Fields!OrderDate.Value)
* You can combine functions in an expression to customize the format. The following expression changes the format of a date in the form month-day-year to month-week-week number. For example, 12/23/2009 to December Week 3:
* =Format(Fields!MyDate.Value, "MMMM") & " Week " & \_
* (Int(DateDiff("d", DateSerial(Year(Fields!MyDate.Value), \_
* Month(Fields!MyDate.Value),1), Fields!FullDateAlternateKey.Value)/7)+1).ToString

When used as a calculated field in a dataset, you can use this expression on a chart to aggregate values by week within each month.

**String Functions**

* Combine more than one field by using concatenation operators and Visual Basic constants. The following expression returns two fields, each on a separate line in the same text box:
* =Fields!FirstName.Value & vbCrLf & Fields!LastName.Value
* Format dates and numbers in a string with the **Format** function. The following expression displays values of the *StartDate* and *EndDate* parameters in long date format:
* =Format(Parameters!StartDate.Value, "D") & " through " & Format(Parameters!EndDate.Value, "D")

If the text box contains only a date or number, you should use the **Format** property of the text box to apply formatting instead of the **Format** function within the text box.

* The **Right**, **Len**, and **InStr** functions are useful for returning a substring, for example, trimming*DOMAIN*\*username* to just the user name. The following expression returns the part of the string to the right of a backslash (\) character from a parameter named *User*:
* =Right(Parameters!User.Value, Len(Parameters!User.Value) - InStr(Parameters!User.Value, "\"))

The following expression results in the same value as the previous one, using members of the .NET Framework [System.String](http://msdn.microsoft.com/en-us/library/system.string.aspx) class instead of Visual Basic functions:

=Parameters!User.Value.Substring(Parameters!User.Value.IndexOf("\")+1, Parameters!User.Value.Length-Parameters!User.Value.IndexOf("\")-1)

* Display the selected values from a multivalue parameter. The following example uses the **Join** function to concatenate the selected values of the parameter *MySelection* into a single string that can be set as an expression for the value of a text box in a report item:
* = Join(Parameters!MySelection.Value)
* The **Regex** functions from the .NET Framework [System.Text.RegularExpressions](http://msdn.microsoft.com/en-us/library/system.text.regularexpressions.aspx) are useful for changing the format of existing strings, for example, formatting a telephone number. The following expression uses the**Replace** function to change the format of a ten-digit telephone number in a field from "*nnn*-*nnn*-*nnnn*" to "(*nnn*) *nnn*-*nnnn*":
* =System.Text.RegularExpressions.Regex.Replace(Fields!Phone.Value, "(\d{3})[ -.]\*(\d{3})[ -.]\*(\d{4})", "($1) $2-$3")

|  |
| --- |
| **Note Note** |
| Verify that the value for Fields!Phone.Value has no extra spaces and is of type [String](http://msdn.microsoft.com/en-us/library/system.string.aspx). |

**Lookup**

* By specifying a key field, you can use the **Lookup** function to retrieve a value from a dataset for a one-to-one relationship, for example, a key-value pair. The following expression displays the product name from a dataset, given the product identifier to match on:
* =Lookup(Fields!PID.Value, Fields!ProductID.Value, Fields.ProductName.Value, "Product")

**LookupSet**

* By specifying a key field, you can use the **LookupSet** function to retrieve a set of values from a dataset for a one-to-many relationship. For example, a person can have multiple telephone numbers. In the following example, assume the dataset PhoneList contains a person identifier and a telephone number in each row.**LookupSet** returns an array of values. The following expression combines the return values into a single string and displays the list of telephone numbers for the person specified by ContactID:
* =Join(LookupSet(Fields!ContactID.Value, Fields!PersonID.Value, Fields!PhoneNumber.Value, "PhoneList"),",")

**Conversion Functions**

You can use Visual Basic functions to convert a field from the one data type to a different data type. Conversion functions can be used to convert a the default data type for a field to the data type needed for calculations or to combine text.

* The following expression converts the constant 500 to type Decimal in order to compare it to a Transact-SQL money data type in the Value field for a filter expression.
* =CDec(500)
* The following expression displays the number of values selected for the multivalue parameter *MySelection*.
* =CStr(Parameters!MySelection.Count)

**Decision Functions**

* The **Iif** function returns one of two values depending on whether the expression is true or not. The following expression uses the **Iif** function to return a Boolean value of **True** if the value of LineTotal exceeds 100. Otherwise it returns **False**:
* =IIF(Fields!LineTotal.Value > 100, True, False)
* Use multiple **IIF** functions (also known as "nested IIFs") to return one of three values depending on the value ofPctComplete. The following expression can be placed in the fill color of a text box to change the background color depending on the value in the text box.
* =IIF(Fields!PctComplete.Value >= 10, "Green", IIF(Fields!PctComplete.Value >= 1, "Blue", "Red"))

Values greater than or equal to 10 display with a green background, between 1 and 9 display with a blue background, and less than 1 display with a red background.

* A different way to get the same functionality uses the **Switch** function. The **Switch** function is useful when you have three or more conditions to test. The **Switch** function returns the value associated with the first expression in a series that evaluates to true:
* =Switch(Fields!PctComplete.Value >= 10, "Green", Fields!PctComplete.Value >= 1, "Blue", Fields!PctComplete.Value = 1, "Yellow", Fields!PctComplete.Value <= 0, "Red",)

Values greater than or equal to 10 display with a green background, between 1 and 9 display with a blue background, equal to 1 display with a yellow background, and 0 or less display with a red background.

* Test the value of the ImportantDate field and return "Red" if it is more than a week old, and "Blue" otherwise. This expression can be used to control the Color property of a text box in a report item:
* =IIF(DateDiff("d",Fields!ImportantDate.Value, Now())>7,"Red","Blue")
* Test the value of the PhoneNumber field and return "No Value" if it is **null** (**Nothing** in Visual Basic); otherwise return the phone number value. This expression can be used to control the value of a text box in a report item.
* =IIF(Fields!PhoneNumber.Value Is Nothing,"No Value",Fields!PhoneNumber.Value)
* Test the value of the Department field and return either a subreport name or a **null** (**Nothing** in Visual Basic). This expression can be used for conditional drillthrough subreports.
* =IIF(Fields!Department.Value = "Development", "EmployeeReport", Nothing)
* Test if a field value is null. This expression can be used to control the **Hidden** property of an image report item. In the following example, the image specified by the field [LargePhoto] is displayed only if the value of the field is not null.
* =IIF(IsNothing(Fields!LargePhoto.Value),True,False)

**Report Functions**

In an expression, you can add a reference to additional report functions that manipulate data in a report. This section provides examples for two of these functions. For more information about report functions and examples, see[Aggregate Functions Reference (Report Builder and SSRS)](http://msdn.microsoft.com/en-us/library/dd255275.aspx).

**Sum**

* The **Sum** function can total the values in a group or data region. This function can be useful in the header or footer of a group. The following expression displays the sum of data in the Order group or data region:
* =Sum(Fields!LineTotal.Value, "Order")
* You can also use the **Sum** function for conditional aggregate calculations. For example, if a dataset has a field that is named State with possible values Not Started, Started, Finished, the following expression, when placed in a group header, calculates the aggregate sum for only the value Finished:
* =Sum(IIF(Fields!State.Value = "Finished", 1, 0))

**RowNumber**

* The **RowNumber** function, when used in a text box within a data region, displays the row number for each instance of the text box in which the expression appears. This function can be useful to number rows in a table. It can also be useful for more complex tasks, such as providing page breaks based on number of rows. For more information, see [Page Breaks](http://msdn.microsoft.com/en-us/library/ms157328.aspx#PageBreaks) in this topic.

The scope you specify for **RowNumber** controls when renumbering begins. The **Nothing** keyword indicates that the function will start counting at the first row in the outermost data region. To start counting within nested data regions, use the name of the data region. To start counting within a group, use the name of the group.

=RowNumber(Nothing)

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Appearance of Report Data**](javascript:void(0))

You can use expressions to manipulate how data appears on a report. For example, you can display the values of two fields in a single text box, display information about the report, or affect how page breaks are inserted in the report.

**Page Headers and Footers**

When designing a report, you may want to display the name of the report and page number in the report footer. To do this, you can use the following expressions:

* The following expression provides the name of the report and the time it was run. It can be placed in a text box in the report footer or in the body of the report. The time is formatted with the .NET Framework formatting string for short date:
* =Globals.ReportName & ", dated " & Format(Globals.ExecutionTime, "d")
* The following expression, placed in a text box in the footer of a report, provides page number and total pages in the report:
* =Globals.PageNumber & " of " & Globals.TotalPages

The following examples describe how to display the first and last values from a page in the page header, similar to what you might find in a directory listing. The example assumes a data region that contains a text box namedLastName.

* The following expression, placed in a text box on the left side of the page header, provides the first value of the LastName text box on the page:
* =First(ReportItems("LastName").Value)
* The following expression, placed in a text box on the right side of the page header, provides the last value of the LastName text box on the page:
* =Last(ReportItems("LastName").Value)

The following example describes how to display a page total. The example assumes a data region that contains a text box named Cost.

* The following expression, placed in the page header or footer, provides the sum of the values in the Costtext box for the page:
* =Sum(ReportItems("Cost").Value)

|  |
| --- |
| **Note Note** |
| You can refer to only one report item per expression in a page header or footer. Also, you can refer to the text box name, but not the actual data expression within the text box, in page header and footer expressions. |

**Page Breaks**

In some reports, you may want to place a page break at the end of a specified number of rows instead of, or in addition to, on groups or report items. To do this, create a group that contains the groups or detail records you want, add a page break to the group, and then add a group expression to group by a specified number of rows.

* The following expression, when placed in the group expression, assigns a number to each set of 25 rows. When a page break is defined for the group, this expression results in a page break every 25 rows.
* =Ceiling(RowNumber(Nothing)/25)

To allow the user to set a value for the number of rows per page, create a parameter named RowsPerPageand base the group expression on the parameter, as shown in the following expression:

=Ceiling(RowNumber(Nothing)/Parameters!RowsPerPage.Value)

For more information about setting page breaks for a group, see [Add a Page Break (Report Builder and SSRS)](http://msdn.microsoft.com/en-us/library/dd207058.aspx).

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Properties**](javascript:void(0))

Expressions are not only used to display data in text boxes. They can also be used to change how properties are applied to report items. You can change style information for a report item, or change its visibility.

**Formatting**

* The following expression, when used in the **Color** property of a text box, changes the color of the text depending on the value of the Profit field:
* =Iif(Fields!Profit.Value < 0, "Red", "Black")

You can also use the Visual Basic object variable Me. This variable is another way of referring to the value of a text box.

=Iif(Me.Value < 0, "Red", "Black")

* The following expression, when used in the **BackgroundColor** property of a report item in a data region, alternates the background color of each row between pale green and white:
* =Iif(RowNumber(Nothing) Mod 2, "PaleGreen", "White")

If you are using an expression for a specified scope, you may have to indicate the dataset for the aggregate function:

=Iif(RowNumber("Employees") Mod 2, "PaleGreen", "White")

|  |
| --- |
| **Note Note** |
| Available colors come from the .NET Framework **KnownColor** enumeration. |

**Chart Colors**

To specify colors for a Shape chart, you can use custom code to control the order that colors are mapped to data point values. This helps you use consistent colors for multiple charts that have the same category groups. For more information, see [Specify Consistent Colors across Multiple Shape Charts (Report Builder and SSRS)](http://msdn.microsoft.com/en-us/library/dd239350.aspx).

**Visibility**

You can show and hide items in a report using the visibility properties for the report item. In a data region such as a table, you can initially hide detail rows based on the value in an expression.

* The following expression, when used for initial visibility of detail rows in a group, shows the detail rows for all sales exceeding 90 percent in the PctQuota field:
* =Iif(Fields!PctQuota.Value>.9, False, True)
* The following expression, when set in the **Hidden** property of a table, shows the table only if it has more than 12 rows:
* =IIF(CountRows()>12,true,false)
* The following expression, when set in the **Hidden** property of a column, shows the column only if the field exists in the report dataset after the data is retrieved from the data source:
* =IIF(Fields!Column\_1.IsMissing, true, false)

**URLs**

You can customize URLs by using report data and also conditionally control whether URLs are added as an action for a text box.

* The following expression, when used as an action on a text box, generates a customized URL that specifies the dataset field EmployeeID as a URL parameter.
* ="http://adventure-works/MyInfo?ID=" & Fields!EmployeeID.Value

For more information, see [Add a Hyperlink to a URL (Report Builder and SSRS)](http://msdn.microsoft.com/en-us/library/dd239346.aspx).

* The following expression conditionally controls whether to add a URL in a text box. This expression depends on a parameter named IncludeURLs that allows a user to decide whether to include active URLs in a report. This expression is set as an action on a text box. By setting the parameter to False and then viewing the report, you can export the report Microsoft Excel without hyperlinks.
* =IIF(Parameters!IncludeURLs.Value,"http://adventure-works.com/productcatalog",Nothing)

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Report Data**](javascript:void(0))

Expressions can be used to manipulate the data that is used in the report. You can refer to parameters and other report information. You can even change the query that is used to retrieve data for the report.

**Parameters**

You can use expressions in a parameter to vary the default value for the parameter. For example, you can use a parameter to filter data to a particular user based on the user ID that is used to run the report.

* The following expression, when used as the default value for a parameter, collects the user ID of the person running the report:
* =User!UserID
* To refer to a parameter in a query parameter, filter expression, text box, or other area of the report, use the**Parameters** global collection. This example assumes that the parameter is named *Department*:
* =Parameters!Department.Value
* Parameters can be created in a report but set to hidden. When the report runs on the report server, the parameter does not appear in the toolbar and the report reader cannot change the default value. You can use a hidden parameter set to a default value as custom constant. You can use this value in any expression, including a field expression. The following expression identifies the field specified by the default parameter value for the parameter named *ParameterField*:
* =Fields(Parameters!ParameterField.Value).Value

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**Custom Code**](javascript:void(0))

You can use custom code in a report. Custom code is either embedded in a report or stored in a custom assembly which is used in the report. For more information about custom code, see [Custom Code and Assembly References in Expressions in Report Designer (SSRS)](http://msdn.microsoft.com/en-us/library/ms159238.aspx).

* The following example calls an embedded code method called **ToUSD**, which converts the StandardCostfield value to a dollar value:
* =Code.ToUSD(Fields!StandardCost.Value)
* The following example shows how to define some custom constants and variables.

[Visual Basic]

Public Const MyNote = "Authored by Bob"

Public Const NCopies As Int32 = 2

Public Dim MyVersion As String = "123.456"

Public Dim MyDoubleVersion As Double = 123.456

Although custom constants and variables do not appear in the Expression Editor Constants view (which only displays built-in constants), you can add references to them from any expression, as shown in the following examples. These are treated as Variants.

[Visual Basic]

=Code.MyNote

=Code.NCopies

=Code.MyVersion

=Code.MyDoubleVersion

**Using Group Variables for Custom Aggregation**

You can initialize the value for a group variable that is local to a particular group scope and then include a reference to that variable in expressions. One of the ways that you can use a group variable with custom code is to implement a custom aggregate. For more information, see [Using Group Variables in Reporting Services 2008 for Custom Aggregation](http://go.microsoft.com/fwlink/?LinkId=128714).

For more information about variables, see [Report and Group Variables Collections References (Report Builder and SSRS)](http://msdn.microsoft.com/en-us/library/dd255208.aspx).

**Replacing Text in a Field at Run Time**

The following example calls an embedded method called **FixSpelling**, which substitutes "Bicycle" for all occurrences of the text "Bike" in SubCategory.Value.

=Code.FixSpelling(Fields!SubCategory.Value)

The following code, when embedded in a report definition, shows an implementation of the **FixSpelling** method. This example shows you how to refer to the .NET Framework [System.Text.StringBuilder](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.aspx) class. For more information, see[Add Code to a Report (SSRS)](http://msdn.microsoft.com/en-us/library/ms156028.aspx).

**VB**

Public Function FixSpelling(ByVal s As String) As String

Dim strBuilder As New System.Text.StringBuilder(s)

If s.Contains("Bike") Then

strBuilder.Replace("Bike", "Bicycle")

Return strBuilder.ToString()

Else : Return s

End If

End Function

**Suppressing Null or Zero Values at Run Time**

Some values in an expression can evaluate to null or undefined at report processing time. This can create run-time errors that result in **#Error** displaying in the text box instead of the evaluated expression. The **IIF** function is particularly sensitive to this behavior because, unlike an If-Then-Else statement, each part of the **IIF** statement is evaluated (including function calls) before being passed to the routine that tests for **true** or **false**. The statement=IIF(Fields!Sales.Value is NOTHING, 0, Fields!Sales.Value) generates**#Error** in the rendered report if Fields!Sales.Value is NOTHING.

To avoid this condition, use one of the following strategies:

* Set the numerator to 0 and the denominator to 1 if the value for field B is 0 or undefined; otherwise, set the numerator to the value for field A and the denominator to the value for field B.
* =IIF(Field!B.Value=0, 0, Field!A.Value / IIF(Field!B.Value =0, 1, Field!B.Value))
* Use a custom code function to return the value for the expression. The following example returns the percentage difference between a current value and a previous value. This can be used to calculate the difference between any two successive values and it handles the edge case of the first comparison (when there is no previous value) and cases whether either the previous value or the current value is **null** (**Nothing** in Visual Basic).
* Public Function GetDeltaPercentage(ByVal PreviousValue, ByVal CurrentValue) As Object
* If IsNothing(PreviousValue) OR IsNothing(CurrentValue) Then
* Return Nothing
* Else if PreviousValue = 0 OR CurrentValue = 0 Then
* Return Nothing
* Else
* Return (CurrentValue - PreviousValue) / CurrentValue
* End If
* End Function

The following expression shows how to call this custom code from a text box:

=Code.GetDeltaPercentage(Previous(Sum(Fields!Sales.Value),"ColumnGroupByYear"), Sum(Fields!Sales.Value))

This helps to avoid run-time exceptions. You can now use an expression like =IIF(Me.Value < 0, "red", "black") in the **Color** property of the text box to conditionally the display text based on whether the values are greater than or less than 0.

[[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))**See Also**](javascript:void(0))