Using functions, formulas and calculations in Web Intelligence

- SAP BusinessObjects Business Intelligence Suite 4.0 Feature Pack 3
## Contents

### Chapter 1
- Document History: Web Intelligence Functions, Formulas and Calculations
- About this guide

### Chapter 2
- Using standard and custom calculations
- Standard calculations
- Using formulas to build custom calculations
- Using variables to simplify formulas
- Working with functions
- Including functions in cells
- Function syntax
- Examples of functions
- Function and formula operators

### Chapter 3
- Understanding calculation contexts
- Calculation contexts defined
- The input context
- The output context
- Default calculation contexts
- Default contexts in a vertical table
- Default contexts in a horizontal table
- Default contexts in a crosstab
- Default contexts in a section
- Default contexts in a break
- Modifying the default calculation context with extended syntax
- Extended syntax operators
- Extended syntax keywords

### Chapter 4
- Calculating values with smart measures
- Smart measures defined
- Grouping sets and smart measures

### Chapter 5
- Grouping sets and smart measures
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.1</td>
<td>Management of grouping sets</td>
<td>42</td>
</tr>
<tr>
<td>5.3</td>
<td>Smart measures and the scope of analysis</td>
<td>42</td>
</tr>
<tr>
<td>5.4</td>
<td>Smart measures and SQL</td>
<td>43</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Grouping sets and the UNION operator</td>
<td>43</td>
</tr>
<tr>
<td>5.5</td>
<td>Smart measures and formulas</td>
<td>45</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Smart measures and dimensions containing formulas</td>
<td>45</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Smart measures in formulas</td>
<td>45</td>
</tr>
<tr>
<td>5.6</td>
<td>Smart measures and filters</td>
<td>46</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Smart measures and filters on dimensions</td>
<td>46</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Filtering smart measures</td>
<td>47</td>
</tr>
<tr>
<td>5.6.3</td>
<td>Smart measures and drill filters</td>
<td>48</td>
</tr>
<tr>
<td>5.6.4</td>
<td>Smart measures and nested OR filters</td>
<td>48</td>
</tr>
</tbody>
</table>

**Chapter 6**

**Functions, operators and keywords**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Functions</td>
<td>49</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Aggregate functions</td>
<td>49</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Character functions</td>
<td>72</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Date and Time functions</td>
<td>87</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Data Provider functions</td>
<td>98</td>
</tr>
<tr>
<td>6.1.5</td>
<td>Document functions</td>
<td>108</td>
</tr>
<tr>
<td>6.1.6</td>
<td>Logical functions</td>
<td>115</td>
</tr>
<tr>
<td>6.1.7</td>
<td>Numeric functions</td>
<td>122</td>
</tr>
<tr>
<td>6.1.8</td>
<td>Set functions</td>
<td>141</td>
</tr>
<tr>
<td>6.1.9</td>
<td>Misc functions</td>
<td>150</td>
</tr>
<tr>
<td>6.2</td>
<td>Function and formula operators</td>
<td>167</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Mathematical operators</td>
<td>168</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Conditional operators</td>
<td>168</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Logical operators</td>
<td>168</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Function-specific operators</td>
<td>171</td>
</tr>
<tr>
<td>6.2.5</td>
<td>Extended syntax operators</td>
<td>178</td>
</tr>
<tr>
<td>6.2.6</td>
<td>Set operators</td>
<td>182</td>
</tr>
<tr>
<td>6.3</td>
<td>Extended syntax keywords</td>
<td>183</td>
</tr>
<tr>
<td>6.3.1</td>
<td>The Block keyword</td>
<td>183</td>
</tr>
<tr>
<td>6.3.2</td>
<td>The Body keyword</td>
<td>184</td>
</tr>
<tr>
<td>6.3.3</td>
<td>The Break keyword</td>
<td>185</td>
</tr>
<tr>
<td>6.3.4</td>
<td>The Report keyword</td>
<td>186</td>
</tr>
<tr>
<td>6.3.5</td>
<td>The Section keyword</td>
<td>187</td>
</tr>
<tr>
<td>6.4</td>
<td>Rounding and truncating numbers</td>
<td>188</td>
</tr>
<tr>
<td>6.5</td>
<td>Referring to members and member sets in hierarchies</td>
<td>189</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>Troubleshooting formulas</td>
<td>191</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>7.1</td>
<td>Formula error and information messages</td>
<td>191</td>
</tr>
<tr>
<td>7.1.1</td>
<td>#COMPUTATION</td>
<td>191</td>
</tr>
<tr>
<td>7.1.2</td>
<td>#CONTEXT</td>
<td>191</td>
</tr>
<tr>
<td>7.1.3</td>
<td>#DATASYNC</td>
<td>192</td>
</tr>
<tr>
<td>7.1.4</td>
<td>#DIV/0</td>
<td>192</td>
</tr>
<tr>
<td>7.1.5</td>
<td>#ERROR</td>
<td>192</td>
</tr>
<tr>
<td>7.1.6</td>
<td>#EXTERNAL</td>
<td>193</td>
</tr>
<tr>
<td>7.1.7</td>
<td>#INCOMPATIBLE</td>
<td>193</td>
</tr>
<tr>
<td>7.1.8</td>
<td>#MIX</td>
<td>193</td>
</tr>
<tr>
<td>7.1.9</td>
<td>#MULTIVALUE</td>
<td>193</td>
</tr>
<tr>
<td>7.1.10</td>
<td>#OVERFLOW</td>
<td>194</td>
</tr>
<tr>
<td>7.1.11</td>
<td>#PARTIALRESULT</td>
<td>194</td>
</tr>
<tr>
<td>7.1.12</td>
<td>#RANK</td>
<td>194</td>
</tr>
<tr>
<td>7.1.13</td>
<td>#RECURSIVE</td>
<td>195</td>
</tr>
<tr>
<td>7.1.14</td>
<td>#REFRESH</td>
<td>195</td>
</tr>
<tr>
<td>7.1.15</td>
<td>#REPFORMULA</td>
<td>195</td>
</tr>
<tr>
<td>7.1.16</td>
<td>#SECURITY</td>
<td>195</td>
</tr>
<tr>
<td>7.1.17</td>
<td>#SYNTAX</td>
<td>196</td>
</tr>
<tr>
<td>7.1.18</td>
<td>#TOREFRESH</td>
<td>196</td>
</tr>
<tr>
<td>7.1.19</td>
<td>#UNAVAILABLE</td>
<td>196</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 8</th>
<th>Comparing values using functions</th>
<th>197</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Comparing values using the Previous function</td>
<td>197</td>
</tr>
<tr>
<td>8.2</td>
<td>Comparing values using the RelativeValue function</td>
<td>198</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Slicing dimensions and the RelativeValue function</td>
<td>198</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Slicing dimensions and sections</td>
<td>200</td>
</tr>
<tr>
<td>8.2.3</td>
<td>Order of slicing dimensions</td>
<td>202</td>
</tr>
<tr>
<td>8.2.4</td>
<td>Slicing dimensions and sorts</td>
<td>204</td>
</tr>
<tr>
<td>8.2.5</td>
<td>Using RelativeValue in crosstabs</td>
<td>205</td>
</tr>
</tbody>
</table>

| Appendix A | More Information | 207 |

| Index      |                                | 209 |
The following table provides an overview of the most important document changes.

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAP BusinessObjects Interactive Analysis 4.0</td>
<td>30, November 2011</td>
<td>Initially rebranded Interactive Analysis. First release of this document</td>
</tr>
<tr>
<td>SAP BusinessObjects Interactive Analysis 4.0 Service Pack 1</td>
<td>25, February 2011</td>
<td>Support Package 1</td>
</tr>
<tr>
<td>BusinessObjects Web Intelligence 4.0 Service Pack 2</td>
<td>15, June 2011</td>
<td>Support Package 2. Rebranded back to Web Intelligence</td>
</tr>
<tr>
<td>BusinessObjects Web Intelligence 4.0 Feature Pack 3</td>
<td>20, February 2012</td>
<td>Feature Pack 3</td>
</tr>
</tbody>
</table>
About this guide

The Using Functions, Formulas and Calculations in Web Intelligence guide provides detailed information on the advanced calculation capabilities available in when you perform data analysis. It also provides a syntax reference to the available functions and operators.
Using standard and custom calculations

You can use standard calculation functions to make quick calculations on data. If standard calculations are not sufficient for your needs, you can use the formula language to build custom calculations.

### 3.1 Standard calculations

You can use standard calculation functions to make quick calculations on data. The following standard calculations are available:

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum</td>
<td>Calculates the sum of the selected data.</td>
</tr>
<tr>
<td>Count</td>
<td>Counts all rows for a measure object or count distinct rows for a dimension or detail object.</td>
</tr>
<tr>
<td>Average</td>
<td>Calculates the average of the data.</td>
</tr>
<tr>
<td>Minimum</td>
<td>Displays the minimum value of the selected data.</td>
</tr>
<tr>
<td>Maximum</td>
<td>Display the maximum value of the selected data.</td>
</tr>
<tr>
<td>Percentage</td>
<td>Displays the selected data as a percentage of the total. The results of the percentage are displayed in an additional column or row of the table. <strong>Note:</strong> Percentages are calculated for the selected measure compared to the total results for that measure on the table or break. To calculate the percentage of one measure compared to another measure, you need to build a custom calculation.</td>
</tr>
<tr>
<td>Default</td>
<td>Applies the default aggregation function to a standard measure, or the database aggregation function to a smart measure.</td>
</tr>
</tbody>
</table>
When you apply standard calculations to table columns, the calculation results appear in footers. One footer is added for each calculation.

3.2 Using formulas to build custom calculations

Custom calculations allow you to add additional calculations to your report beyond its base objects and standard calculations.

You add a custom calculation by writing a formula. A formula can consist of base report variables, functions, operators and calculation contexts.

A custom calculation is a formula that can consist of report objects, functions and operators. Formulas have a calculation context that you can specify explicitly if you choose.

Example: Showing average revenue per sale

If you have a report with Sales Revenue and Number Sold objects and you want to add revenue per sale to the report. The calculation \([\text{Sales Revenue}] / [\text{Number Sold}]\) gives this value by dividing the revenue by the number of items sold in order to give the revenue per item.

Related Topics
- Calculation contexts defined

3.2.1 Using variables to simplify formulas

If a formula is complex you can use variables to simplify it. By using variables you break a complex formula down into manageable parts and make it much easier to read, as well as making building formulas much less error-prone.

You can use previously-created variables in a formula in exactly the same way as you use other report objects. Variables appear in the formula editor under the Variables folder.

You can type this variable name into a formula or drag the variable to the Formula toolbar as you would for any report object.

Related Topics
- Simplifying a variance formula with variables
3.3 Working with functions

A custom calculation sometimes contains report objects only, for example \([\text{Sales Revenue}] / [\text{Number of Sales}]\). Calculations can also include functions in addition to report objects.

A function receives zero or more values as input and returns output based on those values. For example, the \text{Sum} function totals all the values in a measure and outputs the result. The formula \text{Sum}([\text{Sales Revenue}]) outputs a total of sales revenues. In this case, the function input is the Sales Revenue measure and the output is the total of all Sales Measures.

Related Topics
• Function and formula operators
• Functions

3.3.1 Including functions in cells

The text in report cells always begins with `=`. Literal text appears in quotation marks, while formulas appear without quotation marks. For example, the formula \text{Average}([\text{Revenue}]) appears in a cell as =\text{Average}([\text{Revenue}]). The text "Average Revenue?" appears as ="Average Revenue?"

You can use text alone in a cell, or mix formulas and text by using the `+` operator. If you want a cell to display the average revenue preceded by the text "Average Revenue: ", the cell text is as follows:

="Average Revenue: " + Average([\text{Revenue}])

Note the space at the end of the text string so that the text and the value are not placed directly side-by-side in the cell.

3.3.2 Function syntax

To use a function you need to know its name, how many input values it requires and the data types of these input values. You also need to know the type of data that the function outputs.

For example, the \text{Sum} function takes a numerical object as input (for example a measure showing sales revenue) and outputs numeric data (the sum of all the values of the measure object).

Here is the syntax of the \text{Abs} function:

\text{num \ Abs(number)}
This syntax tells you that the \texttt{Abs} function takes a single number as input and returns a number as output.

The Formula Editor displays the function syntax when you select the function.

### 3.3.3 Examples of functions

#### Example: Showing prompt input with the \texttt{UserResponse} function

You have a report showing Year, Quarter and Sales revenue. The State object also appears in the report data, although it is not displayed. When the user runs the report they are presented with a prompt and they must choose a state. You want to show the state that they have chosen in the report title. If your data provider is called "eFashion" and the text in the prompt is "Choose a State", the formula for the title is:

\begin{verbatim}
"Quarterly Revenues for " + UserResponse( "eFashion";"Choose a State")
\end{verbatim}

The report is as follows when the user has chosen Illinois as the state when refreshing the data provider:

**Quarterly Revenues for Illinois**

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>$256,454</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>$241,458</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>$107,006</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>$133,306</td>
</tr>
<tr>
<td>2001</td>
<td>Total</td>
<td>$738,223.80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002</td>
<td>Q1</td>
<td>$334,297</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>$264,722</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>$230,573</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>$331,057</td>
</tr>
<tr>
<td>2002</td>
<td>Total</td>
<td>$1,150,650.80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2003</td>
<td>Q1</td>
<td>$256,658</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>$354,724</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>$273,166</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>$250,517</td>
</tr>
<tr>
<td>2003</td>
<td>Total</td>
<td>$1,134,005.40</td>
</tr>
</tbody>
</table>
Example: Calculating a percentage using the Percentage function

The Percentage function calculates percentages. This function calculates the percentage of a number in relation to its surrounding context. For example, the following table shows revenues by year and quarter. The percentage column contains the formula `Percentage ([Sales Revenue])`.

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>$2660706</td>
<td>0.07</td>
</tr>
<tr>
<td>2001</td>
<td>Q2</td>
<td>$2279003</td>
<td>0.06</td>
</tr>
<tr>
<td>2001</td>
<td>Q3</td>
<td>$1367841</td>
<td>0.04</td>
</tr>
<tr>
<td>2001</td>
<td>Q4</td>
<td>$1797586</td>
<td>0.06</td>
</tr>
<tr>
<td>2002</td>
<td>Q1</td>
<td>$3325172</td>
<td>0.05</td>
</tr>
<tr>
<td>2002</td>
<td>Q2</td>
<td>$2840651</td>
<td>0.08</td>
</tr>
<tr>
<td>2002</td>
<td>Q3</td>
<td>$2879303</td>
<td>0.08</td>
</tr>
<tr>
<td>2002</td>
<td>Q4</td>
<td>$4186120</td>
<td>0.12</td>
</tr>
<tr>
<td>2003</td>
<td>Q1</td>
<td>$3742089</td>
<td>0.11</td>
</tr>
<tr>
<td>2003</td>
<td>Q2</td>
<td>$4006718</td>
<td>0.11</td>
</tr>
<tr>
<td>2003</td>
<td>Q3</td>
<td>$3863396</td>
<td>0.11</td>
</tr>
<tr>
<td>2003</td>
<td>Q4</td>
<td>$3352041</td>
<td>0.09</td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td></td>
<td></td>
<td><strong>1</strong></td>
</tr>
</tbody>
</table>

In this case the function calculates each revenue as a percentage of the total revenue. The surrounding context is the total revenue; this is the only revenue figure that is relevant outside the breakdown by year and quarter in the table.

If the report is split into sections by year, the surrounding context outside the table becomes the total revenue in the section.

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>$2586703</td>
<td>0.33</td>
</tr>
<tr>
<td>2001</td>
<td>Q2</td>
<td>$2279003</td>
<td>0.26</td>
</tr>
<tr>
<td>2001</td>
<td>Q3</td>
<td>$1367584</td>
<td>0.17</td>
</tr>
<tr>
<td>2001</td>
<td>Q4</td>
<td>$1768582</td>
<td>0.22</td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td></td>
<td></td>
<td><strong>1</strong></td>
</tr>
</tbody>
</table>

If the Percentage cell is placed outside the table but still inside the section, the surrounding context becomes the total revenue. In this case the Percentage function calculates the total revenue for the section as a percentage of the total overall revenue.
Example: Calculating a percentage using the Sum function

You can gain more control over the context in which a percentage is calculated by using the Sum function rather than the Percentage function. If you divide one figure in a set of figures by the total of those figures, you get its percentage of the total; for example, the formula \( \frac{\text{Sales Revenue}}{\text{Sum(\text{Sales Revenue})}} \) gives the sales revenue as a percentage of the total revenue.

In the following table the Percentage of Total column has the formula:

\[ \frac{\text{Sales revenue}}{\text{Sum(\text{Sales revenue} In Report)}} \]

and the Percentage of Year column has the formula:

\[ \frac{\text{Sales revenue}}{\text{Sum(\text{Sales revenue} In Section)}} \]

These formulas take advantage of the extended syntax keywords Report and Section to instruct the Sum function to calculate the overall total revenue and yearly revenue respectively.

Related Topics

• Modifying the default calculation context with extended syntax
### 3.3.3.1 Simplifying a variance formula with variables

Variance is a statistical term. The variance of a set of values measures the spread of those values around their average. The `Var` function calculates the variance in one step, but manual calculation of variance provides a good example of how to simplify a complex formula using variables. To calculate the variance manually you need to:

- calculate the average number of items sold
- calculate the difference between each number of items sold and the average, then square this value
- add up all these squared differences
- divide this total by the number of values - 1

You have a report showing numbers of items sold by quarter and you want to include the variance. Without the use of variables to simplify it, this complex formula is as follows:

\[
\text{Sum}((\text{[Quantity sold]} - \text{Average([Quantity sold] ForEach [Quarter]) In Report})^2 \text{ [Difference Squared]} \text{ In [Quarter]})/(\text{Count ([Quantity sold] ForEach [Quarter])} - 1)
\]

#### Creating the variance formula

There are several steps involved in creating a variance formula. You encapsulate each of these steps in a variable. The variables you create are:

- average number of items sold
- number of observations (that is, the number of separate values of the number of items sold)
- difference between an observation and the average, squared
- sum of these differences divided by the number of observations - 1

The variable formulas are as follows:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Sold</td>
<td>Average([Quantity Sold] In ([Quarter]) In Report)</td>
</tr>
<tr>
<td>Number of Observations</td>
<td>Count([Quantity Sold] In ([Quarter]) In Report)</td>
</tr>
<tr>
<td>Difference Squared</td>
<td>Power(([Quantity sold] - [Average Sold]);2)</td>
</tr>
<tr>
<td>Variance</td>
<td>Sum([Difference Squared] In ([Quarter])/([Number of Observations] - 1)</td>
</tr>
</tbody>
</table>

The final formula is now

\[
\text{Sum} \left( \frac{\text{[Difference Squared]}}{\text{[Number of Observations] - 1}} \right)
\]

which is much easier to understand. This simplified version of the formula gives you a high-level view of what the formula is doing, rather than plunging you into the confusing details. You can then examine the formulas of the variables referenced in the high-level formula to understand its component parts.
For example, the formula references the variable Difference Squared, which itself references the variable Average Sold. By examining the formulas of Difference Squared and Average sold, you can drill down into the formula to understand the details of what it is doing.

### 3.3.4 Function and formula operators

Operators link the various components in a formula. Formulas can contain mathematical, conditional, logical, function-specific or extended syntax operators.

#### 3.3.4.1 Mathematical operators

Mathematical operators are familiar from everyday arithmetic. There are addition (+), subtraction (-), multiplication (*), division (/) operators that allow you to perform mathematical operations in a formula. The formula \([\text{Sales Revenue}] - [\text{Cost of Sales}]\) contains a mathematical operator, in this case subtraction.

**Note:**
When used with character strings, the `+` operator becomes a string concatenation operator. That is, it joins character strings. For example, the formula "John" + " Smith" returns "John Smith".

#### 3.3.4.2 Conditional operators

Conditional operators determine the type of comparison to be made between values.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>
You use conditional operators with the If function, as in:

```
If [Revenue]>10000 Then "High" Else "Low"
```

which returns "High" for all rows where the revenue is greater than or equal to 10000 and "Low" for all other rows.

### 3.3.4.3 Logical operators

The logical operators are **And**, **Or**, **Not**, **Between** and **Inlist**. Logical operators are used in boolean expressions, which return **True** or **False**.

### 3.3.4.4 Context operators

Context operators form part of extended calculation syntax. Extended syntax allows you to define which dimensions a measure or formula takes into account in a calculation.

### 3.3.4.5 Function-specific operators

Some functions can take specific operators as arguments. For example, the **Previous** function can take the **Self** operator.

All functions use `)` and `,` to enclose function arguments. Functions that accept multiple parameters use `;` to separate the parameters.
Using standard and custom calculations
4.1 Calculation contexts defined

The calculation context is the data that a calculation takes into account to generate a result. This means that the value given by a measure is determined by the dimensions used to calculate the measure.

A report contains two kinds of objects:

- Dimensions represent business data that generate figures. Store outlets, years or regions are examples of dimension data. For example, a store outlet, a year or a region can generate revenue: we can talk about revenue by store, revenue by year or revenue by region.
- Measures are numerical data generated by dimension data. Examples of measure are revenue and number of sales. For example, we can talk about the number of sales made in a particular store.

Measures can also be generated by combinations of dimension data. For example, we can talk about the revenue generated by a particular store in 2005.

The calculation context of a measure has two components:

- the dimension or list of dimensions that determine the measure value
- the part of the dimension data that determines the measure value

The calculation context has two components:

- The input context
- The output context

Related Topics

- The input context
- The output context

4.1.1 The input context

The input context of a measure or formula is the list of dimensions that feed into the calculation.
The list of dimensions in an input context appears inside the parentheses of the function that outputs the value. The list of dimensions must also be enclosed in parentheses (even if it contains only one dimension) and the dimensions must be separated by semicolons.

**Example: Specifying an input context**

In a report with Year sections and a block in each section with Customer and Revenue columns, the input contexts are:

<table>
<thead>
<tr>
<th>Report part</th>
<th>Input context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section header and block footers</td>
<td>Year</td>
</tr>
<tr>
<td>Rows in the block</td>
<td>Year, Customer</td>
</tr>
</tbody>
</table>

In other words, the section headers and block footers show aggregated revenue by Year, and each row in the block shows revenue aggregated by Year and Customer (the revenue generated by that customer in the year in question).

When specified explicitly in a formula, these input contexts are:

\[
\text{Sum ([Revenue] In ([Year]))}
\]

\[
\text{Sum ([Revenue] In ([Year],[Customer]))}
\]

That is, the dimensions in the input context appear inside the parentheses of the function (in this case, Sum) whose input context is specified.

**4.1.2 The output context**

The output context causes the formula to output a value if it is placed in the footer of a block containing a break.

**Example: Specifying an output context**

The following report shows revenue by year and quarter, with a break on year, and the minimum revenue calculated by year:
What if you want to show the minimum revenue by year in a block with no break? You can do this by specifying the output context in a formula. In this case, the formula looks like this:

\[ \text{Min (\{Revenue\}) In (\{Year\})} \]

That is, the output context appears after the parentheses of the function whose output context you are specifying. In this case, the output context calculates the minimum revenue by year.

If you add an additional column containing this formula to the block, the result is as follows:

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
<th>Min by Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>$2660699.50</td>
<td>$1367840.70</td>
</tr>
<tr>
<td>2001</td>
<td>Q2</td>
<td>$2279003.00</td>
<td>$1367840.70</td>
</tr>
<tr>
<td>2001</td>
<td>Q3</td>
<td>$1367840.70</td>
<td>$1367840.70</td>
</tr>
<tr>
<td>2001</td>
<td>Q4</td>
<td>$1786680.40</td>
<td>$1367840.70</td>
</tr>
<tr>
<td>2002</td>
<td>Q1</td>
<td>$3326172.20</td>
<td>$2840650.80</td>
</tr>
<tr>
<td>2002</td>
<td>Q2</td>
<td>$2840650.80</td>
<td>$2840650.80</td>
</tr>
<tr>
<td>2002</td>
<td>Q3</td>
<td>$3356041.10</td>
<td>$3356041.10</td>
</tr>
<tr>
<td>2002</td>
<td>Q4</td>
<td>$3356041.10</td>
<td>$3356041.10</td>
</tr>
</tbody>
</table>

You can see that the Min By Year column contains the minimum revenues that appear in the break footers in the previous report.
Notice that in this example, the input context is not specified because it is the default context (Year, Quarter) for the block. In other words, the output context determines which revenue by year and quarter to output. In full, with both input and output formulas explicitly specified, the formula looks like this:

\[ \text{Min ([Sales Revenue] In ([Year]; [Quarter])) In ([Year])} \]

Explained in words, this formula calculates revenues by year by quarter, then outputs the smallest of these revenues that occurs in each year.

What would happen if you did not specify the output context in the Min by Year column? In this case, these figures would be identical to the figures in the Sales Revenue column. Why? Remember that the default context in a block includes the dimensions in that block. The minimum revenue by year by quarter is the same as the revenue by year by quarter simply because there is only one revenue for each year/quarter combination.

### 4.2 Default calculation contexts

A measure has a default calculation context depending on its place in the report. The figures returned by a measure depend on the dimensions with which it is associated. This combination of dimensions represents the calculation context.

You can change the default context with extended syntax. In other words, you can determine the set of dimensions used to generate a measure. This is what is meant by defining the calculation context.

**Example: Default contexts in a report**

This example describes the default calculation context of the measures in a simple report. The report shows revenue generated by customers and is split into sections by year.

<table>
<thead>
<tr>
<th>Customer</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harris</td>
<td>1000</td>
</tr>
<tr>
<td>Jones</td>
<td>3000</td>
</tr>
<tr>
<td>Walsh</td>
<td>4000</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>8000</strong></td>
</tr>
</tbody>
</table>

Report total: 8000
The table below lists the calculation context of the measures in this report:

<table>
<thead>
<tr>
<th>Measure</th>
<th>Value</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report total</td>
<td>20000</td>
<td>Total of all revenues in the report</td>
</tr>
<tr>
<td>Section header total</td>
<td>8000</td>
<td>Year</td>
</tr>
<tr>
<td>Customer total</td>
<td>1000, 3000, 4000</td>
<td>Year;Customer</td>
</tr>
<tr>
<td>Block footer total</td>
<td>8000</td>
<td>Year</td>
</tr>
</tbody>
</table>

**Related Topics**
- Calculation contexts defined
- Modifying the default calculation context with extended syntax

### 4.2.1 Default contexts in a vertical table

A vertical table is a standard report table with headers at the top, data going from top to bottom and footers at the bottom. The default contexts in a down table are:

<table>
<thead>
<tr>
<th>When the calculation is in the...</th>
<th>The input context is</th>
<th>The output context is</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>The dimensions and measures used to generate the body of the block</td>
<td>All the data is aggregated then the calculation function returns a single value</td>
</tr>
<tr>
<td>Body of the block</td>
<td>The dimensions and measures used to generate the current row</td>
<td>The same as the input context</td>
</tr>
<tr>
<td>Footer</td>
<td>The dimensions and measures used to generate the body of the block</td>
<td>All the data is aggregated then the calculation function returns a single value</td>
</tr>
</tbody>
</table>

### 4.2.2 Default contexts in a horizontal table
A horizontal table is like a vertical table turned on its side. Headers appear at the left, data goes left to right and footers appear at the right. The default contexts for a horizontal table are the same as those for a vertical table.

### 4.2.3 Default contexts in a crosstab

A crosstab displays data in a matrix with measures appearing at the intersections of dimensions. The default contexts in a crosstab are:

<table>
<thead>
<tr>
<th>The calculation is in the...</th>
<th>The input context is...</th>
<th>The output context is...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>The dimensions and measures used to generate the body of the block.</td>
<td>All the data is aggregated, then the calculation function returns a single value.</td>
</tr>
<tr>
<td>Body of the block</td>
<td>The dimensions and measures used to generate the body of the block.</td>
<td>The same as the input context.</td>
</tr>
<tr>
<td>Footer</td>
<td>The dimensions and measures used to generate the body of the block.</td>
<td>All the data is aggregated, then the calculation function returns a single value.</td>
</tr>
<tr>
<td>VBody footer</td>
<td>The dimensions and measures used to generate the current column.</td>
<td>All the data is aggregated, then the calculation function returns a single value.</td>
</tr>
<tr>
<td>HBody Footer</td>
<td>The dimensions and measures used to generate the current row.</td>
<td>All the data is aggregated, then the calculation function returns a single value.</td>
</tr>
<tr>
<td>VFooter</td>
<td>Same as footer.</td>
<td>All the data is aggregated, then the calculation function returns a single value.</td>
</tr>
<tr>
<td>HFooter</td>
<td>Same as footer.</td>
<td>All the data is aggregated, then the calculation function returns a single value.</td>
</tr>
</tbody>
</table>

**Example: Default contexts in a crosstab**

The following report shows the default contexts in a crosstab:
4.2.4 Default contexts in a section

A section consists of a header, body and footer. The default contexts in a section are:

<table>
<thead>
<tr>
<th>The calculation is in the...</th>
<th>The input context is...</th>
<th>The output context is...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body</td>
<td>The dimensions and measures in the report, filtered to restrict the data to the section data.</td>
<td>All the data is aggregated, then the calculation function returns a single value.</td>
</tr>
</tbody>
</table>

**Example: Default contexts in a section**

The following report shows the default contexts in a crosstab:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
<td>Q2</td>
<td>Q3</td>
<td>Q4</td>
<td></td>
</tr>
<tr>
<td>France</td>
<td>259,170</td>
<td>61,895</td>
<td>78,555</td>
<td>70,180</td>
<td>50,840</td>
</tr>
<tr>
<td>US</td>
<td>856,560</td>
<td>138,381</td>
<td>189,900</td>
<td>234,574</td>
<td>226,709</td>
</tr>
<tr>
<td>Sum</td>
<td>1,115,730</td>
<td>258,276</td>
<td>268,441</td>
<td>304,654</td>
<td>286,909</td>
</tr>
</tbody>
</table>
4.2.5 Default contexts in a break

A break consists of a header, body and footer. The default contexts in a break are:

<table>
<thead>
<tr>
<th>The calculation is in the...</th>
<th>The input context is...</th>
<th>The output context is...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Current instance of the break.</td>
<td>All the data is aggregated, then the calculation function returns a single value.</td>
</tr>
<tr>
<td>Footer</td>
<td>Current instance of the break.</td>
<td>All the data is aggregated, then the calculation function returns a single value.</td>
</tr>
</tbody>
</table>
Example: **Default contexts in a break**

The following report shows the default contexts in a break:

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>$3200700</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>$2279000</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>$1367341</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>$1768500</td>
</tr>
<tr>
<td></td>
<td>Sum:</td>
<td>$8096124</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002</td>
<td>Q1</td>
<td>$3398772</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>$2946651</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>$2679968</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>$4168720</td>
</tr>
<tr>
<td></td>
<td>Sum:</td>
<td>$13232246</td>
</tr>
</tbody>
</table>

### 4.3 Modifying the default calculation context with extended syntax

Extended syntax uses context operators that you add to a formula or measure to specify its calculation context. A measure or formula context consists of its input context and output context.

#### 4.3.1 Extended syntax operators

You specify input and output contexts explicitly with context operators. The following table lists the context operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Specifies an explicit list of dimensions to use in the context.</td>
</tr>
<tr>
<td>ForEach</td>
<td>Adds dimensions to the default context</td>
</tr>
<tr>
<td>ForAll</td>
<td>Removes dimensions from the default context</td>
</tr>
</tbody>
</table>
The ForAll and ForEach operators are useful when you have a default context with many dimensions. It is often easier to add or subtract from the context using ForAll and ForEach than it is to specify the list explicitly using In.

### 4.3.1.1 In context operator

The In context operator specifies dimensions explicitly in a context.

**Example: Using In to specify the dimensions in a context**

In this example you have a report showing Year and Sales Revenue. Your data provider also contains the Quarter object but you do not include this dimension in the block. Instead, you want to include an additional column to show the maximum revenue by quarter in each year. Your report looks like this:

<table>
<thead>
<tr>
<th>Year</th>
<th>Sales revenue</th>
<th>Max Quarterly Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>$8,096,123.60</td>
<td>$2,660,699.50</td>
</tr>
<tr>
<td>2002</td>
<td>$13,232,246.00</td>
<td>$4,186,120.00</td>
</tr>
<tr>
<td>2003</td>
<td>$15,059,142.80</td>
<td>$4,006,717.50</td>
</tr>
</tbody>
</table>

You can see where the values in the Max Quarterly Revenue column come from by examining this block in conjunction with a block that includes the Quarter dimension:
The Max Quarterly Revenue column shows the highest quarterly revenue in each year. For example, Q4 has the highest revenue in 2002, so the Max Quarterly Revenue shows Q4 revenue on the row showing 2002.

Using the In operator, the formula for Max Quarterly Revenue is

\[
\text{Max} \left( \{ \text{Sales Revenue} \} \text{ In } \{ \text{Year} \};\{\text{Quarter}\} \} \text{ In } \{\text{Year}\} \right)
\]

This formula calculates the maximum sales revenue for each (Year,Quarter) combination, then outputs this figure by year.

**Note:**
Because the default output context of the block is Year, you do not need to specify the output context explicitly in this formula.
4.3.1.2 ForEach context operator

The ForEach operator adds dimensions to a context.

Example: Using ForEach to add dimensions to a context

The following table shows the maximum revenue for each Quarter in a report which contains the Quarter dimension but does not include it in the block:

<table>
<thead>
<tr>
<th>Year</th>
<th>Sales revenue</th>
<th>Max Quarterly Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>8096123.60</td>
<td>2660699.50</td>
</tr>
<tr>
<td>2002</td>
<td>13232246.00</td>
<td>4186120.00</td>
</tr>
<tr>
<td>2003</td>
<td>15059142.80</td>
<td>4006717.50</td>
</tr>
</tbody>
</table>

It is possible to create a formula for the Max Quarterly Revenue column that does not include the ForEach operator:

\[
\text{Max ([Sales Revenue] In ([Year];[Quarter])) In ([Year])}
\]

Using the ForEach context operator, you can achieve the same result with the following formula:

\[
\text{Max ([Sales Revenue] ForEach ([Quarter])) In ([Year])}
\]

Why? Because the Year dimension is the default input context in the block. By using the ForEach operator, you add the Quarter dimension to the context, giving an input context of ([Year];[Quarter]).

4.3.1.3 ForAll context operator

The ForAll context operator removes dimensions from a context.

Example: Using ForAll to remove dimensions from a context

You have a report showing Year, Quarter and Sales Revenue and you want to add a column that shows the total revenue in each year, as shown in the following block:
To total revenues by year the input context needs to be (Year); by default it is (Year; Quarter). Therefore, you can remove Quarter from the input context by specifying ForAll ([Quarter]) in the formula, which looks like this:

\[
\text{Sum}([\text{Sales Revenue}] \text{ ForAll} ([\text{Quarter}]))
\]

Note that you can use the In operator to achieve the same thing; in this case the formula is:

\[
\text{Sum}([\text{Sales Revenue}] \text{ In} ([\text{Year}]))
\]

This version of the formula explicitly specifies Year as the context, rather than removing Quarter to leave Year.

### 4.3.2 Extended syntax keywords

Extended syntax keywords are a form of shorthand that allows you to refer to dimensions in extended syntax without specifying those dimensions explicitly. This helps future-proof reports; if formulas do not contain hard-coded references to dimensions, they will continue to work even if dimensions are added to or removed from a report.

There are five extended syntax keywords: Report, Section, Break, Block and Body.

#### 4.3.2.1 The Report keyword

The following table describes the data referenced by the Report keyword depending on where it is placed in a report:

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
<th>Yearly Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>$2680700</td>
<td>$6096124</td>
</tr>
<tr>
<td>2001</td>
<td>Q2</td>
<td>$2279000</td>
<td>$6096124</td>
</tr>
<tr>
<td>2001</td>
<td>Q3</td>
<td>$1367841</td>
<td>$6096124</td>
</tr>
<tr>
<td>2001</td>
<td>Q4</td>
<td>$17895900</td>
<td>$6096124</td>
</tr>
<tr>
<td>2002</td>
<td>Q1</td>
<td>$3326172</td>
<td>$12232248</td>
</tr>
<tr>
<td>2002</td>
<td>Q2</td>
<td>$2040651</td>
<td>$12232248</td>
</tr>
<tr>
<td>2002</td>
<td>Q3</td>
<td>$3893803</td>
<td>$12232248</td>
</tr>
<tr>
<td>2002</td>
<td>Q4</td>
<td>$4186120</td>
<td>$12232248</td>
</tr>
<tr>
<td>2003</td>
<td>Q1</td>
<td>$3742899</td>
<td>$15059143</td>
</tr>
<tr>
<td>2003</td>
<td>Q2</td>
<td>$4006718</td>
<td>$15059143</td>
</tr>
<tr>
<td>2003</td>
<td>Q3</td>
<td>$3953395</td>
<td>$15059143</td>
</tr>
<tr>
<td>2003</td>
<td>Q4</td>
<td>$3356241</td>
<td>$15059143</td>
</tr>
</tbody>
</table>
4.3.2.2 The Section keyword

The following table describes the data referenced by the Section keyword depending on where it is placed in a report.

<table>
<thead>
<tr>
<th>When placed in...</th>
<th>References this data...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A block</td>
<td>All data in the section</td>
</tr>
<tr>
<td>A block break (header or footer)</td>
<td>All data in the section</td>
</tr>
</tbody>
</table>

Example: The Report keyword

You have a report showing Year, Quarter and Sales revenue. The report has a column, Report Total, that shows the total of all revenue in the report.

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
<th>Report Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>$2,860,700</td>
<td>$36,387,512.4</td>
</tr>
<tr>
<td>2001</td>
<td>Q2</td>
<td>$2,273,003</td>
<td>$36,387,512.4</td>
</tr>
<tr>
<td>2001</td>
<td>Q3</td>
<td>$1,367,941</td>
<td>$36,387,512.4</td>
</tr>
<tr>
<td>2001</td>
<td>Q4</td>
<td>$1,700,510</td>
<td>$36,387,512.4</td>
</tr>
<tr>
<td>2002</td>
<td>Q1</td>
<td>$3,255,172</td>
<td>$36,387,512.4</td>
</tr>
<tr>
<td>2002</td>
<td>Q2</td>
<td>$2,840,651</td>
<td>$36,387,512.4</td>
</tr>
<tr>
<td>2002</td>
<td>Q3</td>
<td>$2,573,303</td>
<td>$36,387,512.4</td>
</tr>
<tr>
<td>2002</td>
<td>Q4</td>
<td>$4,186,120</td>
<td>$36,387,512.4</td>
</tr>
<tr>
<td>2003</td>
<td>Q1</td>
<td>$3,742,999</td>
<td>$36,387,512.4</td>
</tr>
<tr>
<td>2003</td>
<td>Q2</td>
<td>$4,006,718</td>
<td>$36,387,512.4</td>
</tr>
<tr>
<td>2003</td>
<td>Q3</td>
<td>$3,953,395</td>
<td>$36,387,512.4</td>
</tr>
<tr>
<td>2003</td>
<td>Q4</td>
<td>$3,366,041</td>
<td>$36,387,512.4</td>
</tr>
</tbody>
</table>

The formula for the Report Total column is `Sum([Sales revenue]) In Report`. Without the Report keyword, this column would duplicate the figures in the Sales Revenue column because it would use the default output context (`([Year];[Quarter])`).
Understanding calculation contexts

<table>
<thead>
<tr>
<th>When placed in...</th>
<th>References this data...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A section (header, footer, or outside a block)</td>
<td>All data in the section</td>
</tr>
<tr>
<td>Outside any blocks or sections</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Example: **The Section keyword**

You have a report showing Year, Quarter, and Sales revenue.

<table>
<thead>
<tr>
<th>Year</th>
<th>Sales revenue</th>
<th>Section Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q1</td>
<td>2,660,700</td>
<td>8,095,814</td>
</tr>
<tr>
<td>Q2</td>
<td>2,278,593</td>
<td>8,095,814</td>
</tr>
<tr>
<td>Q3</td>
<td>1,367,841</td>
<td>8,095,814</td>
</tr>
<tr>
<td>Q4</td>
<td>1,758,280</td>
<td>8,095,814</td>
</tr>
</tbody>
</table>

The report has a section based on Year. The Section Total column has the formula:

\[
\text{Sum ([Sales Revenue]) In Section}
\]

The figure in the Section Total column is the total revenue for 2001, because the section break occurs on the Year object. Without the Section keyword this column would duplicate the figures in the Sales revenue column, because it would use the default output context ([Year];[Quarter]).

---

### 4.3.2.3 The Break keyword

The following table describes the dimensions referenced by the Break keyword depending on where it is placed in a report:

<table>
<thead>
<tr>
<th>When placed in...</th>
<th>References this data...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A block</td>
<td>Data in the part of a block delimited by a break</td>
</tr>
<tr>
<td>A block break (header or footer)</td>
<td>Data in the part of a block delimited by a break</td>
</tr>
<tr>
<td>A section (header, footer, or outside a block)</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Outside any blocks or sections</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Example: **The Break keyword**

You have a report showing Year, Quarter and Sales revenue:
The report has break on Year. The Break Total column has the formula:

\[
\text{Sum ([Sales Revenue]) In Break}
\]

Without the Break keyword this column would duplicate the figures in the Sales revenue column, because it would use the default output context ([Year];[Quarter]).

### 4.3.2.4 The Block keyword

The following table describes the dimensions referenced by the Block keyword depending on where it is placed in a report: the Block keyword often encompasses the same data as the Section keyword. The difference is that Block accounts for filters on a block whereas Section ignores them.

<table>
<thead>
<tr>
<th>When placed in...</th>
<th>References this data...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A block</td>
<td>Data in the whole block, ignoring breaks, respecting filters</td>
</tr>
<tr>
<td>A block break (header or footer)</td>
<td>Data in the whole block, ignoring breaks, respecting filters</td>
</tr>
<tr>
<td>A section (header, footer, or outside a block)</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Outside any blocks or sections</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**Example: The Block keyword**

You have a report showing Year, Quarter and Sales revenue. The report has a section based on Year. The block is filtered to exclude the third and fourth quarters.
The Yearly Average column has the formula

\text{Average([Sales revenue] In Section)}

and the First Half Average column has the formula

\text{Average ([Sales revenue]) In Block}

You can see how the Block keyword takes account of the filter on the block.

### 4.3.2.5 The Body keyword

The following table describes the dimensions referenced by the Body keyword depending on where it is placed in a report:

<table>
<thead>
<tr>
<th>When placed in...</th>
<th>References this data...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A block</td>
<td>Data in the block</td>
</tr>
<tr>
<td>A block break (header or footer)</td>
<td>Data in the block</td>
</tr>
<tr>
<td>A section (header, footer, or outside a block)</td>
<td>Data in the section</td>
</tr>
<tr>
<td>Outside any blocks or sections</td>
<td>Data in the report</td>
</tr>
</tbody>
</table>
**Example: The Body keyword**

You have a report showing Year, Quarter and Sales revenue, with a break on Year. The report has a section based on Year and a break on Quarter.

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
<th>Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>2,660,700</td>
<td>2,660,699.5</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>2,279,003</td>
<td>2,279,003</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>1,367,841</td>
<td>1,367,840.7</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>1,788,580</td>
<td>1,788,580.4</td>
</tr>
<tr>
<td>2001</td>
<td></td>
<td></td>
<td>8,096,123.6</td>
</tr>
</tbody>
</table>

The Body column has the formula

\[
\text{Sum ([Sales Revenue]) In Body}
\]

The totals in the Body column are the same as those in the Sales revenue column because the Body keyword refers to the data in the block. If you were to remove the Month object, the figures in the Block column would change to correspond with the changed figures in the Sales revenue column. If you were to place the formula in the report footer it would return the total revenue for the body.

---

**4.3.2.6 Using keywords to make reports generic**

Extended syntax keywords future-proof your report against changes. If you refer to data explicitly (by specifying dimensions using In, ForEach or ForAll) your reports might return unexpected data if dimensions are added or removed. The following example illustrates this.

**Example: Using the Report keyword to display percentages**

In this example you have a block that contains Year, Quarter and Sales revenue objects. You want to display revenues by year and quarter, and the percentage of the total revenue in the report that each individual revenue represents, as shown:
The formula for the Percentage of Total column is:

\[
\left(\frac{[\text{Sales revenue}]}{\text{Sum}([\text{Sales revenue}] \text{ In Report})}\right) \times 100
\]

In a block, the Report includes all data in a report, so this formula could be written:

\[
\left(\frac{[\text{Sales revenue}]}{\text{Sum}([\text{Sales revenue}] \text{ ForAll } ([\text{Year}];[\text{Quarter}]))}\right) \times 100
\]

This formula removes Year and Quarter from the output context (in other words, calculates a grand total) because there are no other dimensions in the report. The formula then divides each revenue by the grand total to give its percentage of the total.

Although you can use ForAll in this situation, it is much better to use the Report keyword. Why? What if the Month dimension were subsequently added to the report? The version of the formula that uses the Report keyword still calculates each percentage correctly, but the version that explicitly specifies the Year and Quarter dimensions is now wrong:
Why is this? The problem lies in:

$$\text{Sum (Sales Revenue) ForAll ([Year];[Quarter])}$$

When Year and Quarter were the only dimensions in the report, this was equivalent to "a grand total of all revenues". Once you add the Month dimension, this expression removes Year and Quarter from the default output context, but leaves Month.

The formula now has a "break" on month. In other words, on every row where Month is 1, this expression now means "the total revenue of all month 1s". In every row where Month is 2, it means "the total revenue of all month 2s". As a result, the percentages are not the percentages you expect.
Calculating values with smart measures

5.1 Smart measures defined

“Smart measures” are measures whose values are calculated by the database (relational or OLAP) on which a universe is based. They differ from classic measures, which are calculated from the detailed values returned by the database. The data returned by smart measures is aggregated in ways not supported natively by the Web Intelligence component of the SAP BusinessObjects Business Intelligence platform.

Queries that contain smart measures calculate the measures in all the calculation contexts required in a report. These contexts can change as the report changes. As a result, the query changes at each data refresh after the required contexts have changed.

When you edit such a report, automatically the #TOREFRESH message is inserted in the report reminding you that the report should be refreshed in order to reflect the changes.

Smart measures behave differently from classic measures, which support a basic set of aggregation functions (Max, Min, Count, Sum, Average) that can be calculated in all contexts without help from the database. For example, if you build a query containing the [Country] and [Region] dimensions and the [Revenue] measure (which calculates the sum of the revenue), the initial display shows Country, Region and Revenue in a block. If you remove Region from the block, the total revenue for each country can still be calculated without a data refresh by summing the revenues for all the regions in the country. A smart measure requires a data refresh in this situation.

Calculation contexts are represented by “grouping sets” in the generated query.

5.2 Grouping sets and smart measures

A “grouping set” is a set of dimensions that generates a result for a measure. The generated SQL that returns the data in a smart measure includes grouping sets for all the aggregations of that measure that are included in the report.

Example: Grouping sets in a query

A query contains the [Country], [Region], [City] dimensions and the [Revenue] smart measure. These objects imply the following grouping sets to calculate revenue in all possible contexts:
• Total smart measure value
• smart measure value by (Country, Region, City)
• smart measure value by (Country, City)
• smart measure value by (City)
• smart measure value by (Region, City)
• smart measure value by (Region)
• smart measure value by (Country, Region)
• smart measure value by (Country)

If the database supports UNION, each grouping set is represented in a UNION clause in the generated SQL.

The grouping sets are updated according to the calculation contexts required by the report, which can change in response to changes in the report structure.

5.2.1 Management of grouping sets

When you first build and run a query including smart measures, the generated SQL includes the grouping set necessary to calculate the smart measures at the most detailed level implied by the query objects.

For example, if you build a query containing the [Country], [Region] and [City] dimensions and the [Revenue] smart measure, the (Country, Region, City) grouping set appears in the generated SQL. The most detailed grouping set always appears in the SQL. Other grouping sets are added and removed in response to changes in the report.

If you remove the [City] dimension from the block, the (Country, Region) grouping set is required to return the revenue values. This grouping set is not yet available in the query SQL, so #TOREFRESH appears in the [Revenue] cells. When you refresh the data, #TOREFRESH is replaced with the revenue values.

If you then replace the [City] dimension in the block, the (Country, Region) grouping set is no longer needed. It is removed from the query SQL and its values discarded the next time you refresh the data.

Each time you refresh the report data, grouping sets are included or discarded according to the calculation contexts required by the report.

In certain situations, it is not possible to display the value of a smart measure. In this case, #UNAVAILABLE appears in the measure cells.

5.3 Smart measures and the scope of analysis
When you build a query with a scope of analysis, the initial grouping set contains the result objects, but not the scope objects. The query does not generate all the possible grouping sets from the combination of the result objects and the scope objects.

Example: **A query with a scope of analysis and a smart measure**

A query has the result objects [Country] and [Revenue]. The scope of analysis contains the [Region] and [City] dimensions. When you run the query, its SQL contains the (Country) grouping set and it displays [Country] and [Revenue] in a block.

---

### 5.4 Smart measures and SQL

#### 5.4.1 Grouping sets and the UNION operator

Some databases support grouping sets explicitly with the `GROUPING SETS` operator. When you build a query containing smart measures, the generated SQL uses multiple result sets and the `UNION` operator to simulate the effect of `GROUPING SETS`.

Example: **Grouping sets retrieved with the UNION operator**

This example describes a query containing [Country], [Region], [City] dimensions and the [Revenue] smart measure.

**Note:**
For simplicity, the smart measure calculates a sum. In practice, a smart measure is not needed for this aggregation because sums are supported natively in Web Intelligence.

When the query is first run, the grouping set is (Country, Region, City). The entire SQL query returns this grouping set and there is no need for the `UNION` operator in the SQL.

If you remove the [City] dimension from the table, the (Country, Region) grouping set is required to display the revenue (which appears initially as #TOREFRESH). After data refresh, the SQL is as follows:

```sql
SELECT
  0 AS GID,
  country.country_name,
  region.region_name,
  NULL,
  sum(city.revenue)
FROM
  country,
  region,
  city
WHERE
  (country.country_id=region.country_id )
```
Calculating values with smart measures

Each grouping set is represented by a `SELECT` statement, and each has its own ID (the GID column). Grouping sets that do not contain the full set of dimensions include empty columns (`SELECT ''`) because each `SELECT` statement in a query including `UNION` must have the same number of columns.

If you add a new block containing [Country] and [Revenue] to the report, the (Country) grouping set is required. The generated SQL now includes three grouping sets as follows:

```sql
AND { region.region_id=city.region_id }
GROUP BY
  country.country_name,
  region.region_name
UNION
SELECT
  1 AS GID,
  country.country_name,
  region.region_name,
  city.city_name,
  sum(city.revenue)
FROM
  country,
  region,
  city
WHERE
  ( country.country_id=region.country_id )
AND ( region.region_id=city.region_id )
GROUP BY
  country.country_name,
  region.region_name,
  city.city_name
UNION
SELECT
  1 AS GID,
  country.country_name,
  region.region_name,
  NULL,
  NULL,
  sum(city.revenue)
FROM
  country,
  city,
  region
WHERE
  ( country.country_id=region.country_id )
AND ( region.region_id=city.region_id )
GROUP BY
  country.country_name
UNION
SELECT
  2 AS GID,
  country.country_name,
  region.region_name,
  city.city_name,
  sum(city.revenue)
FROM
  country,
  region,
  city
WHERE
  ( country.country_id=region.country_id )
```
5.5 Smart measures and formulas

5.5.1 Smart measures and dimensions containing formulas

If a formula or variable appears as a dimension in the calculation context of a smart measure, and the formula determines the grouping set required by the measure, the values of the smart measure cannot be displayed.

For example, a report contains a variable, Semester, with the formula

\[
\text{If } [\text{Quarter}] = "Q1" \text{ or } [\text{Quarter}] = "Q2" \text{ Then } "H1" \text{ Else } "H2"
\]

Placed in a block, the Semester variable returns the following result:

<table>
<thead>
<tr>
<th>Semester</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>#UNAVAILABLE</td>
</tr>
<tr>
<td>H2</td>
<td>#UNAVAILABLE</td>
</tr>
</tbody>
</table>

5.5.2 Smart measures in formulas

Smart measures can return values when included in formulas, even when the formula requires a different calculation context from the context implied by the position of the formula.

For example, a report contains a block as follows:

<table>
<thead>
<tr>
<th>Country</th>
<th>Region</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>North</td>
<td>10000</td>
</tr>
</tbody>
</table>
If you include an additional column in this table with the formula

\[\text{Revenue} \text{ ForAll} \{\text{Region}\}\]

the initial value of the column is #TOREFRESH because the formula, which excludes regions from the calculation, requires the grouping set (Country). Refreshing the data adds the (Country) grouping set to the query and displays the values of the measure.

**Related Topics**
- [ForAll context operator](#)

### 5.6 Smart measures and filters

#### 5.6.1 Smart measures and filters on dimensions

If you apply a multi-valued filter to a dimension on which the value of a smart value depends, but the dimension does not appear explicitly in the calculation context of the measure, the smart measure cannot return a value and the cell displays #UNAVAILABLE.

#UNAVAILABLE appears because the measure must be filtered in the report and then aggregated, but a smart measure cannot be aggregated after a report-level filter is applied. Calculating the measure would be possible by adding a query filter to the generated SQL, but this solution carries the risk of impacting other reports based on the same query.

**Note:**
- A multi-valued filter filters on multiple values using operators such as Greater Than, In List or Less Than. You can apply single-valued filters such as Equal To without generating the #UNAVAILABLE error.
- There is a workaround for cases which do not require aggregation: Define the formula as variable whose qualification is a measure and be sure that the used dimension is included in the block with the variable (you can hide that column for a better display).
**Example: A smart measure and a filter on a dimension**

A query contains the Country and Resort dimensions and the Revenue smart measure. Country and Revenue are displayed in a block. If you apply a report filter restricting the values of Resort to "French Riviera" or "Bahamas Beach", #UNAVAILABLE appears in the Revenue cells.

<table>
<thead>
<tr>
<th>Country</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>#UNAVAILABLE</td>
</tr>
<tr>
<td>US</td>
<td>#UNAVAILABLE</td>
</tr>
<tr>
<td>Sum:</td>
<td>#UNAVAILABLE</td>
</tr>
</tbody>
</table>

If you restrict Resort to "Bahamas Beach" only, the values are displayed.

<table>
<thead>
<tr>
<th>Country</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>971,444</td>
</tr>
<tr>
<td>Sum:</td>
<td>971,444</td>
</tr>
</tbody>
</table>

**5.6.2 Filtering smart measures**

If you apply a filter to a smart measure in a context where the smart measure is aggregated at different levels of detail, the smart measure returns the #UNAVAILABLE error when it is aggregated at a less detailed level.

**Example: Filtering a smart measure**

You have a block displaying revenue by country and resort, where Revenue is a smart measure.

<table>
<thead>
<tr>
<th>Country</th>
<th>Resort</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>French Riviera</td>
<td>835,420</td>
</tr>
<tr>
<td>US</td>
<td>Bahamas Beach</td>
<td>971,444</td>
</tr>
<tr>
<td>US</td>
<td>Hawaiian Club</td>
<td>1,479,660</td>
</tr>
<tr>
<td></td>
<td>Sum:</td>
<td>3,286,524</td>
</tr>
</tbody>
</table>

If you apply a filter, Revenue > 900,000, to the block, the total displays #UNAVAILABLE.
Calculating values with smart measures

<table>
<thead>
<tr>
<th>Country</th>
<th>Resort</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>Bahamas Beach</td>
<td>971,444</td>
</tr>
<tr>
<td>US</td>
<td>Hawaiian Club</td>
<td>1,479,660</td>
</tr>
</tbody>
</table>

#UNAVAILABLE appears because the revenue in the block is aggregated by the grouping set (Country, Resort), but the total revenue in the footer, which is also impacted by the filter, is aggregated at a less detailed level than (Country, Resort).

5.6.3 Smart measures and drill filters

In general, it is not possible to return values for smart measures when a filter is applied to a dimension that impacts the calculation of the measure. Dimensions filtered by drill filters are an exception to this rule.

Example: A drill filter that affects a smart measure

A block contains the [Country] and [Revenue] objects. You drill on [Country] to display [Region], [Revenue] in the block and move the filter on [Country] to the drill toolbar.

To do this, the (Country, Region) grouping set is added to the query. The result is then filtered to display only those regions contained in the drilled country. It is not necessary to add a filter at the query level to filter regions based on their country.

5.6.4 Smart measures and nested OR filters

Nested OR filters in which at least one of the filtered dimensions does not appear in a block generate the #UNAVAILBLE error for a smart measure in the block.

Example:

6.1 Functions

Functions are divided into the following categories:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate</td>
<td>Aggregates data (for example by summing or averaging a set of values)</td>
</tr>
<tr>
<td>Character</td>
<td>Manipulates character strings</td>
</tr>
<tr>
<td>Date and Time</td>
<td>Returns date or time data</td>
</tr>
<tr>
<td>Document</td>
<td>Returns data about a document</td>
</tr>
<tr>
<td>Data Provider</td>
<td>Returns data about a document's data provider</td>
</tr>
<tr>
<td>Logical</td>
<td>Returns TRUE or FALSE</td>
</tr>
<tr>
<td>Numeric</td>
<td>Returns numeric data</td>
</tr>
<tr>
<td>Misc</td>
<td>Functions that do not fit into the above categories</td>
</tr>
<tr>
<td>Set</td>
<td>Returns sets of members from hierarchies</td>
</tr>
</tbody>
</table>

6.1.1 Aggregate functions

6.1.1.1 Aggregate
**Description**

Returns the default aggregation of a measure for a given member set

**Function Group**

Aggregate

**Syntax**

```
num Aggregate(measure[;member_set])
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
<tr>
<td>member_set</td>
<td>The member set used to calculate</td>
<td>Member set</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>the aggregation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes**

- You can use extended syntax context operators with `Aggregate`.
- If you include `member_set`, `Aggregate` returns the aggregate value of the measure for all members in the member set.
- `member_set` can include multiple sets separated by semicolons (;).
- The list of member sets must be enclosed in {}.
- All member sets must be from the current hierarchy. If another hierarchy is used, the #MULTIVALUE error occurs.
- If a hierarchy only is specified in a member set, the current member of the hierarchy is used.

**Examples**

If the default aggregation of the `[Sales Revenue]` measure is Sum, and `[California]` is a member in the `[Geography]` hierarchy (Country > State > City), `Aggregate([Sales Revenue];{Descendants([Geography]&[US].[California];1)}` returns the total sales revenue of all cities in California.

**Related Topics**

- [Referring to members and member sets in hierarchies](#)

---

**6.1.2 Average**

**Description**

Returns the average value of a measure
**Function Group**
Aggregate

**Syntax**
\[ \text{num } \text{Average} \left( \text{measure} \left[ ; \text{member}\_\text{set} \right] \left[ ; \text{IncludeEmpty} \right] \right) \]

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
<tr>
<td>member_set</td>
<td>A set of members</td>
<td>Member set</td>
<td>No</td>
</tr>
<tr>
<td>Include_Empty</td>
<td>Includes empty rows in the calculation</td>
<td>Keyword</td>
<td>No</td>
</tr>
</tbody>
</table>

**Notes**
- You can use extended syntax context operators with \textit{Average}.
- If you include \textit{member\_set}, \textit{Average} returns the average value of the measure for all members in the member set.
- \textit{member\_set} can include multiple sets separated by semicolons (;).
- The list of member sets must be enclosed in {}.
- All member sets must be from the current hierarchy. If another hierarchy is used, the \#MULTIVALUE error occurs.
- If a hierarchy only is specified in a member set, the current member of the hierarchy is used.

**Examples**
If the [Sales Revenue] measure has the values 41569, 30500, 40000 and 50138, \textit{Average([Sales Revenue])} returns 40552.

If [California] is a member in the [Geography] hierarchy (Country > State > City), \textit{Average([Sales Revenue];{[Geography]&[US].[California].children}) returns the average sales revenue of all cities in California.

**Related Topics**
- Referring to members and member sets in hierarchies
- Include\_Empty operator

---

6.1.1.3 Count
Description
Returns the number of values in a set of values

Function Group
Aggregate

Syntax
integer Count(aggregated_data|member_set|[IncludeEmpty]|[Distinct|All])

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregated_data</td>
<td>Any dimension, measure, hierarchy, level or member set</td>
<td>Dimension, measure, hierarchy, member set</td>
<td>Yes</td>
</tr>
<tr>
<td>member_set</td>
<td>The member set used to calculate the count</td>
<td>Member set</td>
<td>No</td>
</tr>
<tr>
<td>IncludeEmpty</td>
<td>Includes empty values in the calculation</td>
<td>Keyword</td>
<td>No</td>
</tr>
<tr>
<td>Distinct</td>
<td>Includes distinct values only (default for dimensions) or all values (default for measures) in the calculation</td>
<td>Keyword</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes
• You can use extended syntax context operators with Count.
• If you specify IncludeEmpty as the second argument, the function takes empty (null) values into consideration in the calculation.
• If you do not specify the Distinct|All parameter, the default values are Distinct for dimensions and All for measures.
• If you include member_set, Count restricts the count to the number of values in member_set.
• member_set can include multiple sets separated by semicolons (;).
• The list of member sets must be enclosed in {}.
• All member sets must be from the current hierarchy. If another hierarchy is used, the #MULTIVALUE error occurs.
• If a hierarchy only is specified in a member set, the current member of the hierarchy is used.

Examples
Count("Test") returns 1
Count([City];Distinct) returns 5 if there are 5 different cities in a list of cities, even if there are more than 5 rows in the list due to duplication.
Count([City];All) returns 10 if there are 10 cities in a list of cities, even though some are duplicated.
Count ([City];IncludeEmpty) returns 6 if there are 5 cities and one blank row in a list of cities.
Count ([Product];{[Geography] & [State]}) returns the total number of products at the [State] level in the [Geography] hierarchy.
6.1.4 First

**Description**
Returns the first value in a data set

**Function Group**
Aggregate

**Syntax**
\[ \text{input	extunderscore type} \text{ First(dimension|measure)} \]

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
<td>measure</td>
<td>Any dimension or measure</td>
<td>Dimension or measure</td>
</tr>
</tbody>
</table>

**Notes**
- When placed in a break footer, `First` returns the first value in the break.
- When placed a section footer, `First` returns the first value in the section.

**Examples**
When placed in a table footer, `First([Revenue])` returns the first value of [Revenue] in the table.

6.1.5 Last

**Description**
Returns the last value in a dimension or measure

**Function Group**
Aggregate

**Syntax**
\[ \text{input	extunderscore type} \text{ Last(dimension|measure)} \]
### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
<td>Any dimension or measure</td>
<td>Dimension or measure</td>
<td>Yes</td>
</tr>
<tr>
<td>measure</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Notes
- When placed in a break footer, `Last` returns the last value in the break.
- When placed a section footer, `Last` returns the last value in the section.

#### Examples
When placed in a table footer, `First([Revenue])` returns the first value of [Revenue] in the table.

### 6.1.1.6 Max

#### Description
Returns the largest value in a set of values

#### Function Group
Aggregate

#### Syntax
```
input_type Max(aggregated_data[,member_set])
```

#### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregated_data</td>
<td>Any dimension, measure, hierarchy, level or member set</td>
<td>Dimension, measure, hierarchy, level or member set</td>
<td>Yes</td>
</tr>
<tr>
<td>member_set</td>
<td>A set of members</td>
<td>Member set</td>
<td>No</td>
</tr>
</tbody>
</table>

#### Notes
- You can use extended syntax context operators with `Max`.
- If you include `member_set`, `Max` returns the maximum value of the aggregated data for all members in the member set.
- `member_set` can include multiple sets separated by semicolons (;).
- The list of member sets must be enclosed in {}.
- All member sets must be from the current hierarchy. If another hierarchy is used, the #MULTIVALUE error occurs.
- If a hierarchy only is specified in a member set, the current member of the hierarchy is used.
Examples
If the [Sales Revenue] measure has the values 3000, 60034 and 901234, \( \text{Max}([\text{Sales Revenue}]) \) returns 901234.

If the [City] dimension has the values "Aberdeen" and "London", \( \text{Max}([\text{City}]) \) returns "London".

If [US] is a member in the [Geography] hierarchy (Country > State > City), \( \text{Max}([\text{Sales Revenue}];([\text{Geography}].[US].Children]) \) returns the highest sales revenue for a US state.

6.1.1.7 Median

Description
Returns the median (middle value) of a measure

Function Group
Aggregate

Syntax
\[ \text{num Median(measure)} \]

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Notes
If the set of numbers has an even number of values, \( \text{Median} \) takes the average of the middle two values.

Examples
\( \text{Median}([\text{Revenue}]) \) returns 971,444 if [Revenue] has the values 835420, 971444, and 1479660.

6.1.1.8 Min

Description
Returns the smallest value in a set of values

Function Group
Aggregate
Syntax
\[ \text{any_type Min(aggregated_data[,member_set])} \]

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregated_data</td>
<td>Any dimension, measure, hierarchy, level or member set</td>
<td>Dimension, measure, hierarchy, level or member set</td>
<td>Yes</td>
</tr>
<tr>
<td>member_set</td>
<td>A set of members</td>
<td>Member set</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes
- You can use extended syntax context operators with Min.
- If you include \( \text{member_set} \), Min returns the minimum value of the aggregated data for all members in the member set.
- \( \text{member_set} \) can include multiple sets separated by semicolons (;).
- The list of member sets must be enclosed in {}.
- All member sets must be from the current hierarchy. If another hierarchy is used, the #MULTIVALUE error occurs.
- If a hierarchy only is specified in a member set, the current member of the hierarchy is used.

Examples
If the [Sales revenue] measure has the values 3000, 60034 and 901234, \( \text{Min([Sales Revenue])} \) returns 3000.

If the [City] dimension has the values Aberdeen and London, \( \text{Min([City])} \) returns "Aberdeen".
\( \text{Min([Sales Revenue];{[Geography]&[US].children})} \) returns the lowest sales revenue for a US state if [US] is a member in the [Geography] hierarchy with levels [Country] > [State] > [City].

6.1.1.9 Mode

Description
Returns the most frequently-occurring value in a data set

Function Group
Aggregate

Syntax
\[ \text{input_type Mode(dimension|measure)} \]
### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension\measure</td>
<td>Any dimension or measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Notes

- **Mode** returns null if the data set does not contain one value that occurs more frequently than all the others.

#### Examples

- `Mode([Revenue])` returns 200 if [Revenue] has the values 100, 200, 300, 200.
- `Mode([Country])` returns the most frequently-occurring value of [Country].

### 6.1.1.10 Percentage

#### Description

Expresses a measure value as a percentage of its embedding context

#### Function Group

Aggregate

#### Syntax

```
num Percentage(measure[;Break][;Row|Col])
```

#### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
<tr>
<td>Break</td>
<td>Accounts for table breaks</td>
<td>Keyword</td>
<td>No</td>
</tr>
<tr>
<td>Row</td>
<td>Col</td>
<td>Sets the calculation direction</td>
<td>Keyword</td>
</tr>
</tbody>
</table>

#### Examples

In the following table, the Percentage column has the formula `Percentage([Sales Revenue])`

<table>
<thead>
<tr>
<th>Year</th>
<th>Sales Revenue</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>1000</td>
<td>10</td>
</tr>
<tr>
<td>2002</td>
<td>5000</td>
<td>50</td>
</tr>
<tr>
<td>2003</td>
<td>4000</td>
<td>40</td>
</tr>
</tbody>
</table>
By default the embedding context is the measure total in the table. You can make the function take account of a break in a table by using the optional Break argument. In this case the default embedding context becomes the table section.

In the following table, the Percentage column has the formula `Percentage([Sales Revenue];Break)`

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales Revenue</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>1000</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>2000</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>5000</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>2000</td>
<td>20</td>
</tr>
<tr>
<td>2001</td>
<td>Sum:</td>
<td>10000</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales Revenue</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002</td>
<td>Q1</td>
<td>2000</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>2000</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>5000</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>1000</td>
<td>10</td>
</tr>
<tr>
<td>2002</td>
<td>Sum:</td>
<td>10000</td>
<td>100</td>
</tr>
</tbody>
</table>

You can use the Percentage function across columns or rows; you can specify this explicitly using the optional Row|Col argument. For example, in the following crosstab, the % column has the formula `Percentage([Sales Revenue];Row)`

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>%</th>
<th>Q2</th>
<th>%</th>
<th>Q3</th>
<th>%</th>
<th>Q4</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>1000</td>
<td>10</td>
<td>2000</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2002</td>
<td>2000</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5000</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.1.1.11 Percentile

**Description**

Returns the nth percentile of a measure

**Function Group**

Numeric

**Syntax**

\[ \text{num \ Percentile(measure;percentile)} \]

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
<tr>
<td>percentile</td>
<td>A percentage expressed as a decimal</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

The nth percentile is a number that is greater than or equal to n% of the numbers in a set. You express n% in the form 0.n.

**Examples**

If [measure] has the set of numbers (10;20;30;40;50), \( \text{Percentile([measure];0.3)} \) returns 22, which is greater than or equal to 30% of the numbers in the set.

6.1.1.12 Product

**Description**

Multiplies the values of a measure

**Function Group**

Aggregate

**Syntax**

\[ \text{num \ Product(measure)} \]
Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Examples

\[ \text{Product (}\{\text{Measure}\}) \] returns 30 if \([\text{Measure}]\) has the values 2, 3, 5.

### 6.1.1.13 RunningAverage

**Description**

Returns the running average of a measure

**Function Group**

Aggregate

**Syntax**

\[ \text{num \_ RunningAverage (measure;}\{\text{Row|Col}|\{\text{IncludeEmpty}];\{\text{reset\_dims}\})} \]

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
<tr>
<td>Row</td>
<td>Col</td>
<td>Sets the calculation direction</td>
<td>Keyword</td>
</tr>
<tr>
<td>IncludeEmpty</td>
<td>Includes empty values in the calculation</td>
<td>Keyword</td>
<td>No</td>
</tr>
<tr>
<td>reset_dims</td>
<td>Resets the calculation on the specified dimensions</td>
<td>Dimension list</td>
<td>No</td>
</tr>
</tbody>
</table>

**Notes**

- You can use extended syntax context operators with \text{RunningAverage}.
- You can set the calculation direction with the \text{Row} and \text{Col} operators.
- If you apply a sort on the measure referenced by \text{RunningAverage}, the running average is calculated after the measure is sorted.
- You must always place dimensions in parentheses even if there is only one dimension in the list of reset dimensions.
- When you specify a set of reset dimensions you must separate them with semi-colons.
- \text{RunningAverage} does not automatically reset the average after a block break or new section.

**Examples**

\[ \text{RunningAverage (}\{\text{Revenue}\}) \] returns these results in the following table:
Running Average Revenue

<table>
<thead>
<tr>
<th>Resort</th>
<th>Revenue</th>
<th>Running Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hawaiian Club</td>
<td>1,479,660</td>
<td>835,420</td>
</tr>
<tr>
<td>Bahamas Beach</td>
<td>971,444</td>
<td>1,225,552</td>
</tr>
<tr>
<td>French Riviera</td>
<td>835,420</td>
<td>1,095,508</td>
</tr>
</tbody>
</table>

RunningAverage([Revenue]; ([Country])) returns these results in the following table:

<table>
<thead>
<tr>
<th>Resort</th>
<th>Revenue</th>
<th>Running Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hawaiian Club</td>
<td>1,479,660</td>
<td>835,420</td>
</tr>
<tr>
<td>Bahamas Beach</td>
<td>971,444</td>
<td>1,225,552</td>
</tr>
<tr>
<td>French Riviera</td>
<td>835,420</td>
<td>835,420</td>
</tr>
</tbody>
</table>

Related Topics
- IncludeEmpty operator
- Row/Col operators

6.1.1.14 RunningCount

Description
Returns the running count of a number set

Function Group
Aggregate

Syntax
num RunningCount(dimension|measure[;Row|Col][;IncludeEmpty][; (reset_dims)])
### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
<td>measure</td>
<td>Any dimension or measure</td>
<td>Dimension or measure</td>
</tr>
<tr>
<td>Row</td>
<td>Col</td>
<td>Sets the calculation direction</td>
<td>Keyword</td>
</tr>
<tr>
<td>IncludeEmpty</td>
<td>Includes empty values in the calculation</td>
<td>Keyword</td>
<td>No</td>
</tr>
<tr>
<td>reset_dims</td>
<td>Resets the calculation on the specified dimensions</td>
<td>Dimension list</td>
<td>No</td>
</tr>
</tbody>
</table>

### Notes

- You can use extended syntax context operators with `RunningCount`.
- You can set the calculation direction with the `Row` and `Col` operators.
- If you apply a sort on the measure referenced by `RunningCount`, the running count is calculated after the measure is sorted.
- You must always place dimensions in parentheses even if there is only one dimension in the list of reset dimensions.
- When you specify a set of reset dimensions you must separate them with semi-colons.
- `RunningCount` does not automatically reset the count after a block break or new section.

### Examples

**RunningCount([Revenue])** returns these results in the following table:

<table>
<thead>
<tr>
<th>Country</th>
<th>Resort</th>
<th>Revenue</th>
<th>Running Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>Hawaiian Club</td>
<td>1,479,660</td>
<td>1</td>
</tr>
<tr>
<td>US</td>
<td>Bahamas Beach</td>
<td>971,444</td>
<td>2</td>
</tr>
<tr>
<td>France</td>
<td>French Riviera</td>
<td>835,420</td>
<td>3</td>
</tr>
</tbody>
</table>

**RunningCount([Revenue];([Country]))** returns these results in the following table:

<table>
<thead>
<tr>
<th>Country</th>
<th>Resort</th>
<th>Revenue</th>
<th>Running Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>Hawaiian Club</td>
<td>1,479,660</td>
<td>1</td>
</tr>
<tr>
<td>US</td>
<td>Bahamas Beach</td>
<td>971,444</td>
<td>2</td>
</tr>
<tr>
<td>France</td>
<td>French Riviera</td>
<td>835,420</td>
<td>1</td>
</tr>
</tbody>
</table>

### Related Topics

- IncludeEmpty operator
- Row/Col operators
- IncludeEmpty operator
- IncludeEmpty operator
**6.1.15 RunningMax**

**Description**
Returns the running maximum of a dimension or measure

**Function Group**
Aggregate

**Syntax**
```
input_type RunningMax(dimension|measure|Row|Col|{reset_dims})
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
<td>measure</td>
<td>Any dimension or measure</td>
<td>Dimension or measure</td>
</tr>
<tr>
<td>Row</td>
<td>Col</td>
<td>Sets the calculation direction</td>
<td>Keyword</td>
</tr>
<tr>
<td>reset_dims</td>
<td>Resets the calculation on the specified dimensions</td>
<td>Dimension list</td>
<td>No</td>
</tr>
</tbody>
</table>

**Notes**
- You can use extended syntax context operators with RunningMax.
- You can set the calculation direction with the Row and Col operators.
- If you apply a sort on the measure referenced by RunningMax, the running maximum is calculated after the measure is sorted.
- You must always place dimensions in parentheses even if there is only one dimension in the list of reset dimensions.
- When you specify a set of reset dimensions you must separate them with semi-colons.
- RunningMax does not automatically reset the max after a block break or new section.

**Examples**

RunningMax([Revenue]) returns these results in the following table:

<table>
<thead>
<tr>
<th>Country</th>
<th>Resort</th>
<th>Revenue</th>
<th>Running Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>French Riviera</td>
<td>835,420</td>
<td>835,420</td>
</tr>
<tr>
<td>US</td>
<td>Bahamas Beach</td>
<td>971,444</td>
<td>971,444</td>
</tr>
<tr>
<td>US</td>
<td>Hawaiian Club</td>
<td>1,479,660</td>
<td>1,479,660</td>
</tr>
</tbody>
</table>
Related Topics

- IncludeEmpty operator
- Row/Col operators

6.1.1.16 RunningMin

Description
Returns the running minimum of a dimension or measure

Function Group
Aggregate

Syntax
input_type RunningMin(dimension|measure;[Row|Col];{reset_dims})

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
<td>Any dimension or measure</td>
<td>Dimension or measure</td>
<td>Yes</td>
</tr>
<tr>
<td>Row</td>
<td>Sets the calculation direction</td>
<td>Keyword</td>
<td>No</td>
</tr>
<tr>
<td>reset_dims</td>
<td>Resets the calculation on the specified dimensions</td>
<td>Dimension list</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes

- You can use extended syntax context operators with RunningMin.
- You can set the calculation direction with the Row and Col operators.
- If you apply a sort on the measure referenced by RunningMin, the running minimum is calculated after the measure is sorted.
- You must always place dimensions in parentheses even if there is only one dimension in the list of reset dimensions.
- When you specify a set of reset dimensions you must separate them with semi-colons.
- RunningMin does not automatically reset the minimum after a block break or new section.

Examples

RunningMin([Revenue]) returns these results in the following table:

<table>
<thead>
<tr>
<th>Country</th>
<th>Resort</th>
<th>Revenue</th>
<th>Running Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>French Riviera</td>
<td>835,420</td>
<td>835,420</td>
</tr>
</tbody>
</table>
### 6.1.1.17 RunningProduct

**Description**

Returns the running product of a measure

**Function Group**

Aggregate

**Syntax**

```
num RunningProduct(measure[;Row|Col][;{reset_dims}])
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
<tr>
<td>Row</td>
<td>Col</td>
<td>Sets the calculation direction</td>
<td>Keyword</td>
</tr>
<tr>
<td>reset_dims</td>
<td>Resets the calculation on the specified dimensions</td>
<td>Dimension list</td>
<td>No</td>
</tr>
</tbody>
</table>

**Notes**

- You can use extended syntax context operators with `RunningProduct`.
- You can set the calculation direction with the `Row` and `Col` operators.
- If you apply a sort on the measure referenced by `RunningProduct`, the running product is calculated after the measure is sorted.
- You must always place dimensions in parentheses even if there is only one dimension in the list of reset dimensions.
- When you specify a set of reset dimensions you must separate them with semi-colons.
- `RunningProduct` does not automatically reset the product after a block break or new section.

**Examples**

`RunningProduct([Number of guests])` returns these results in the following table:

<table>
<thead>
<tr>
<th></th>
<th>Bahamas Beach</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>971,444</td>
<td>835,420</td>
<td></td>
</tr>
<tr>
<td>US</td>
<td>1,479,660</td>
<td>835,420</td>
<td></td>
</tr>
</tbody>
</table>

**Related Topics**

- IncludeEmpty operator
- Row/Col operators
Running Product\([\text{Number of guests}];([\text{Country of origin}])\) returns these results in the following table:

<table>
<thead>
<tr>
<th>Country of origin</th>
<th>City</th>
<th>Number of guests</th>
<th>Running Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Japan</td>
<td>Kobe</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Japan</td>
<td>Osaka</td>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>US</td>
<td>Chicago</td>
<td>241</td>
<td>5,784</td>
</tr>
</tbody>
</table>

Related Topics
- [IncludeEmpty operator](#)
- [Row/Col operators](#)

### 6.1.18 RunningSum

**Description**
Returns the running sum of a measure

**Function Group**
Aggregate

**Syntax**

\[
\text{num RunningSum(measure[;Row|Col][;(reset\_dims)])}
\]

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
<tr>
<td>Row</td>
<td>Col</td>
<td>Sets the calculation direction</td>
<td>Keyword</td>
</tr>
</tbody>
</table>
### RequiredTypeDescriptionParameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset_dms</td>
<td>Resets the calculation on the specified dimensions</td>
<td>Dimension list</td>
<td>No</td>
</tr>
</tbody>
</table>

### Notes
- You can use extended syntax context operators with the RunningSum.
- You can set the calculation direction with the Row and Col operators.
- If you apply a sort on the measure referenced by the RunningSum function, the running sum is calculated after the measure is sorted.
- You must always place dimensions in parentheses even if there is only one dimension in the list of reset dimensions.
- When you specify a set of reset dimensions you must separate them with semi-colons.
- RunningSum does not automatically reset the sum after a block break or new section.

### Example
RunningSum([Revenue]) returns these results in the following table:

<table>
<thead>
<tr>
<th>Country</th>
<th>Resort</th>
<th>Revenue</th>
<th>Running Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>French Riviera</td>
<td>835,420</td>
<td>835,420</td>
</tr>
<tr>
<td>US</td>
<td>Bahamas Beach</td>
<td>971,444</td>
<td>1,806,864</td>
</tr>
<tr>
<td>US</td>
<td>Hawaiian Club</td>
<td>1,479,660</td>
<td>3,286,524</td>
</tr>
</tbody>
</table>

RunningSum([Revenue];([Country])) returns these results in the following table:

<table>
<thead>
<tr>
<th>Country</th>
<th>Resort</th>
<th>Revenue</th>
<th>Running Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>French Riviera</td>
<td>835,420</td>
<td>835,420</td>
</tr>
<tr>
<td>US</td>
<td>Bahamas Beach</td>
<td>971,444</td>
<td>971,444</td>
</tr>
<tr>
<td>US</td>
<td>Hawaiian Club</td>
<td>1,479,660</td>
<td>2,451,104</td>
</tr>
</tbody>
</table>

### Related Topics
- IncludeEmpty operator
- Row/Col operators

### 6.1.1.19 ServerValue
**Description**
Returns the database value of a measure

**Function Group**
Aggregate

**Syntax**

```sql
num ServerValue([measure])
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
- ServerValue ignores all local filters applied to dimensions or hierarchies used to calculate the measure

**Example**

ServerValue([Internet Sales Amount]) returns the database value of the measure [Internet Sales Amount]

---

**6.1.1.20 StdDev**

**Description**
Returns the standard deviation of a measure

**Function Group**
Aggregate

**Syntax**

```sql
num StdDev(measure)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

The standard deviation is a measure of the statistical dispersion in a set of numbers. It is calculated by:
- finding the average of the set of numbers
- subtracting the average from each number in the set and squaring the difference
- summing all these squared differences
- dividing this sum by \((number\ of\ numbers\ in\ the\ set\ - 1)\)
- finding the square root of the result

**Examples**

If `measure` has the set of values \((2, 4, 6, 8)\) then `StdDev([measure])` returns 2.58.

**Related Topics**

- `Var`

---

**6.1.1.21 StdDevP**

**Description**

Returns the population standard deviation of a measure

**Function Group**

Aggregate

**Syntax**

```
num StdDevP(measure)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

The population standard deviation is a measure of the statistical dispersion in a set of numbers. It is calculated by:

- finding the average of the set of numbers;
- subtracting the average from each number in the set and squaring the difference;
- summing all these squared differences;
- dividing this sum by \((number\ of\ numbers\ in\ the\ set)\);
- finding the square root of the result.

You can use extended syntax context operators with `StdDevP`.

**Examples**

If `measure` has the set of values \((2, 4, 6, 8)\) then `StdDevP([measure])` returns 2.24.
6.1.22 Sum

**Description**
Returns the sum of a measure

**Function Group**
Aggregate

**Syntax**
```
num Sum(measure[;member_set])
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
<tr>
<td>member_set</td>
<td>A set of members</td>
<td>Member set</td>
<td>No</td>
</tr>
</tbody>
</table>

**Notes**

- You can use extended syntax context operators with `Sum`.
- If you include `member_set`, `Sum` returns the sum of the measure for all members in the member set.
- `member_set` can include multiple sets separated by semicolons (;).
- The list of member sets must be enclosed in {}.
- All member sets must be from the current hierarchy. If another hierarchy is used, the #MULTIVALUE error occurs.
- If a hierarchy only is specified in a member set, the current member of the hierarchy is used.

**Examples**

If the Sales Revenue measure has the values 2000, 3000, 4000, and 1000, `Sum([Sales Revenue])` returns 10000.

If `[California]` is a member in the [Geography] hierarchy (Country > State > City), `Sum([Sales Revenue];{Descendants([Geography]&[US].[California];1)`) returns the total sales revenue of all cities in California.

6.1.23 Var
**Description**
Returns the variance of a measure

**Function Group**
Aggregate

**Syntax**
```
num Var(measure)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
The variance is a measure of the statistical dispersion in a set of numbers. It is calculated by:
- finding the average of the set of numbers
- subtracting the average from each number in the set and squaring the difference
- summing all these squared differences
- dividing this sum by \((number\ of\ numbers\ in\ the\ set\ - 1)\)

The variance is the square of the standard deviation.

You can use extended syntax context operators with \(\text{Var}\).

**Examples**
If `measure` has the set of values \((2, 4, 6, 8)\) \(\text{Var}([\text{measure}])\) returns 6.67.

**Related Topics**
- StdDev

---

**6.1.1.24 VarP**

**Description**
Returns the population variance of a measure

**Function Group**
Aggregate

**Syntax**
```
num VarP(measure)
```
Notes
The population variance is a measure of the statistical dispersion in a set of numbers. It is calculated by:
• finding the average of the set of numbers
• subtracting the average from each number in the set and squaring the difference
• summing all these squared differences
• dividing this sum by \( \frac{\text{number of numbers in the set}}{} \) 

The population variance is the square of the population standard deviation.

You can use extended syntax context operators with \texttt{VarP}.

Examples
If \texttt{measure} has the set of values \(2, 4, 6, 8\) \texttt{VarP([measure])} returns 5.

Related Topics
• \texttt{StdDevP}

6.1.2 Character functions

6.1.2.1 \texttt{Asc}

Description
Returns the ASCII value of a character

Function Group
Character

Syntax
\[ \text{int Asc(string)} \]
### Asc

**Description**
Returns the ASCII value of the first character in a string.

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Any string</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
If `string` contains more than one character, the function returns the ASCII value of the first character in the string.

**Examples**
- `Asc("A")` returns 65.
- `Asc("ab")` returns 97.
- `Asc([Country])` returns 85 when the value of [Country] is "US".

### Char

**Description**
Returns the character associated with an ASCII code.

**Function Group**
Character

**Syntax**
```
string Char(ascii_code)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascii_code</td>
<td>An ASCII code</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
If `number` is a decimal, the function ignores the decimal part.

**Example**
```
s
Char(123) returns "{".
```
### 6.1.2.3 Concatenation

**Description**
Concatenates (joins) two character strings

**Function Group**
Character

**Syntax**

```
string Concatenation(first_string;second_string)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>first_string</td>
<td>The first string</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>second_string</td>
<td>The second string</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
You can also use the `+` operator to concatenate strings.

"First " + "Second" returns "First Second".

"First " + "Second" + " Third" returns "First Second Third".

You can use concatenation to include multiple dimensions in an aggregation function. For example, Count([Sales Person]+[Quarter]+[Resort]) is equivalent to the syntax Count(<Sales Person>,<Quarter>,<Resort>) that is allowed by Desktop Intelligence.

**Examples**
Concatenation("First ";"Second") returns "First Second".

Concatenation("First ";Concatenation("Second ";"Third")) returns "First Second Third".

### 6.1.2.4 Fill

**Description**
Builds a string by repeating a string \( n \) times
**Function Group**
Character

**Syntax**
```
string Fill(repeating_string;num_repeats)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>repeating_string</td>
<td>The repeating string</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>num_repeats</td>
<td>The number of repeats</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

Fill ("New York ";2) returns "New York New York ".

---

**6.1.2.5 FormatDate**

**Description**
Formats a date according to a specified format

**Function Group**
Character

**Syntax**
```
string FormatDate(date;format_string)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>The date to format</td>
<td>Date</td>
<td>Yes</td>
</tr>
<tr>
<td>format_string</td>
<td>The format to apply</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

- The format of the output is dependent on the date format applied to the cell.
- The color formatting strings (for example: [Red], [Blue] and so on) cannot be applied to FormatDate.

**Examples**

FormatDate (CurrentDate();"dd/MM/yyyy") returns "15/12/2005" if the current date is 15 December 2005.
### 6.1.2.6 FormatNumber

**Description**
Formats a number according to a specified format

**Function Group**
Character

**Syntax**
```
string FormatNumber(number;format_string)
```

**Input**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>The number to format</td>
<td>Number</td>
<td>Yes</td>
</tr>
<tr>
<td>format_string</td>
<td>The format to apply</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
- The format of the output is dependent on the number format applied to the cell.
- The color formatting strings (for example: [Red], [Blue] and so on) cannot be applied to `FormatNumber`.

**Examples**
`FormatNumber([Revenue];"#,##.00")` returns 835,420.00 if [Revenue] is 835,420.

### 6.1.2.7 HTMLEncode

**Description**
Applies HTML encoding rules to a string

**Function Group**
Character

**Syntax**
```
string HTMLEncode(html)
```
### 6.1.2.8 InitCap

**Description**
Capitalizes the first letter of a string

**Function Group**
Character

**Syntax**
```plaintext
string InitCap(string)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>The string to capitalize</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

InitCap("we hold these truths to be self-evident") returns "We hold these truths to be self-evident".

### 6.1.2.9 Left

**Description**
Returns the leftmost characters of a string

**Function Group**
Character

**Syntax**
```plaintext
string Left(string;num_chars)
```
### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>The input string</td>
<td>string</td>
<td>Yes</td>
</tr>
<tr>
<td>num_chars</td>
<td>The number of characters to return from the left</td>
<td>number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

Left([Country];2) returns "Fr" if [Country] is "France".

### 6.1.2.10 LeftPad

**Description**

Pads a string on its left with another string

**Function Group**

Character

**Syntax**

```
string LeftPad(padded_string;length;left_string)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>padded_string</td>
<td>The original string</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>The length of the output string</td>
<td>Number</td>
<td>Yes</td>
</tr>
<tr>
<td>left_string</td>
<td>The string to be added to the left of padded_string</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

- If length is less than the length of left_string and padded_string combined, left_string is truncated.
- If length is less than or equal to the length of padded_string, the function returns padded_string.
- If length is greater than the lengths of padded_string and left_string combined, left_string is repeated or partially repeated enough times to fill out the length.

**Examples**

LeftPad("York";8;"New ") returns "New York"

LeftPad("York";6;"New ") returns "NeYork"
LeftPad("York";11;"New ") returns "New NewYork"

LeftPad("New ";2;"York") returns "New".

### 6.1.2.11 LeftTrim

**Description**
Trims the leading spaces from a string

**Function Group**
Character

**Syntax**

```plaintext
string LeftTrim(trimmed_string)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>trimmed_string</td>
<td>The string to be trimmed</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

LeftTrim([Country]) returns "France" if [Country] is " France".

### 6.1.2.12 Length

**Description**
Returns the number of characters in a string

**Function Group**
Character

**Syntax**

```plaintext
int Length(string)
```
Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>The input string</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Examples
Length([Last Name]) returns 5 if [Last Name] is "Smith".

6.1.2.13 Lower

Description
Converts a string to lower case

Function Group
Character

Syntax
`string Lower(string)`

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>The string to be converted to lower case</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Examples
Lower("New York") returns "new york".

6.1.2.14 Match

Description
Determines whether a string matches a pattern

Function Group
Character

Syntax
`bool Match(test_string;pattern)`
### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_string</td>
<td>The string to be tested against the text pattern</td>
<td>string</td>
<td>Yes</td>
</tr>
<tr>
<td>pattern</td>
<td>The text pattern</td>
<td>string</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### Notes
- The pattern can contain the wildcards "*" (replaces any set of characters) or "?" (replaces any single character).

#### Examples
- `Match([Country]; "F*")` returns True if `[Country]` is "France".
- `Match([Country]; "?S?")` returns True if `[Country]` is "USA".

### 6.1.2.15 Pos

#### Description
Returns the starting position of a text pattern in a string

#### Function Group
Character

#### Syntax
```
int Pos(test_string; pattern)
```

#### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_string</td>
<td>The string to be tested for the text pattern</td>
<td>string</td>
<td>Yes</td>
</tr>
<tr>
<td>pattern</td>
<td>The text pattern</td>
<td>string</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### Notes
- If the pattern occurs more than once, `Pos` returns the position of the first occurrence.

#### Examples
- `Pos("New York"; "Ne")` returns 1.
- `Pos("New York, New York"; "Ne")` returns 1.
6.1.2.16 Replace

Description
Replaces part of a string with another string

Function Group
Character

Syntax
string Replace(replace_in;replaced_string;replace_with)

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>replace_in</td>
<td>The string in which the text is replaced</td>
<td>string</td>
<td>Yes</td>
</tr>
<tr>
<td>replaced_string</td>
<td>The text to be replaced</td>
<td>string</td>
<td>Yes</td>
</tr>
<tr>
<td>replace_with</td>
<td>The text that replaces replaced_string</td>
<td>string</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Examples
Replace("New YORK"; "ORK"; "ork") returns "New York".

6.1.2.17 Right

Description
Returns the rightmost characters of a string

Function Group
Character

Syntax
string Right(string;num_chars)
## Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Any string</td>
<td>string</td>
<td>Yes</td>
</tr>
<tr>
<td>num_chars</td>
<td>The number of characters to return from the right</td>
<td>number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Examples

Right([Country];2) returns "ce" if [Country] is "France".

## 6.1.2.18 RightPad

### Description

Pads a string on its right with another string

### Function Group

Character

### Syntax

```
string RightPad(padded_string;length;right_string)
```

### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>padded_string</td>
<td>The original string</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>The length of the output string</td>
<td>Number</td>
<td>Yes</td>
</tr>
<tr>
<td>right_string</td>
<td>The string to be added to the right of padded_string</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Notes

- If `length` is less than the length of `right_string` and `padded_string` combined, `right_string` is truncated.
- If `length` is less than or equal to the length of `padded_string`, the function returns `padded_string`
- If `length` is greater than the lengths of `padded_string` and `right_string` combined, `right_string` is repeated or partially repeated enough times to fill out the length.

### Examples

RightPad("New ";3;"York") returns "New York"

RightPad("New ";6;"York") returns "New Yo"
RightPad("New ";11;"York") returns "New YorkYor"
RightPad("New ";2;"York") returns "New".

### 6.1.2.19 RightTrim

**Description**
Trims the trailing spaces from a string

**Function Group**
Character

**Syntax**
string RightTrim(trimmed_string)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>trimmed_string</td>
<td>The string to be trimmed</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**
RightTrim([Country]) returns "France" if [Country] is "France ".

### 6.1.2.20 Substr

**Description**
Returns part of a string

**Function Group**
Character

**Syntax**
string SubStr(string;start;length)
### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Any string</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>start</td>
<td>The start position of the extracted string</td>
<td>Number</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>The length of the extracted string</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

SubStr("Great Britain";1;5) returns "Great".

SubStr("Great Britain";7;7) returns "Britain".

### 6.1.2.21 Trim

**Description**

Trims the leading and trailing spaces from a string

**Function Group**

Character

**Syntax**

```plaintext
string Trim(trimmed_string)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>The string to be trimmed</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

Trim(" Great Britain ") returns "Great Britain".

### 6.1.2.22 Upper

**Description**

Converts a string to upper case

**Function Group**

Character
### Syntax

```
string Upper(string)
```

### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>The string to be converted</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Examples

```
Upper("New York") returns "NEW YORK".
```

#### 6.1.2.23 UrlEncode

**Description**

Applies URL encoding rules to a string

**Function Group**

Character

**Syntax**

```
string UrlEncode(html)
```

### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>html</td>
<td>The URL to be encoded</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Examples

```
```

#### 6.1.2.24 WordCap

**Description**

Capitalizes the first letter of all the words in a string

**Function Group**

Character
### Syntax

```plaintext
string WordCap(string)
```

### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>The string to be capitalized</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Examples

`WordCap("Sales revenue for March")` returns "Sales Revenue For March".

---

### 6.1.3 Date and Time functions

#### 6.1.3.1 CurrentDate

**Description**

Returns the current date formatted according to the regional settings

**Function Group**

Date and Time

**Syntax**

```plaintext
date CurrentDate()
```

**Examples**

`CurrentDate()` returns 10 September 2002 if the date is 10 September 2002.

---

#### 6.1.3.2 CurrentTime

**Description**

Returns the current time formatted according to the regional settings

**Function Group**

Date and Time
Syntax
```
time CurrentTime()
```

Examples
CurrentTime returns 11:15 if the current time is 11:15.

### 6.1.3.3 DayName

**Description**
Returns the day name in a date

**Function Group**
Date and Time

**Syntax**
```
string DayName(date)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>The input date</td>
<td>Date</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**
DayName([Reservation Date]) returns "Saturday' when the date in [Reservation Date] is 15 December 2001 (which is a Saturday).

**Note**
The input date must be a variable. You cannot specify the date directly, as in DayName("07/15/2001").

### 6.1.3.4 DayNumberOfMonth

**Description**
Returns the day number in a month

**Function Group**
Date and Time

**Syntax**
```
int DayNumberOfMonth(date)
```
Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>The input date</td>
<td>Date</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Examples

DayNumberOfMonth([Reservation Date]) returns 15 when the date in [Reservation Date] is 15 December 2001.

6.1.3.5 DayNumberOfWeek

Description
Retruns the day number in a week

Function Group
Date and Time

Syntax

```
int DayNumberOfWeek(date)
```

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>The input date</td>
<td>Date</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Notes
The function treats Monday as the first day of the week.

Examples

DayNumberOfWeek([Reservation Date]) returns 1 when the date in [Reservation Date] is 2 May 2005 (which is a Monday).

6.1.3.6 DayNumberOfYear

Description
Returns the day number in a year
**Function Group**
Date and Time

**Syntax**

```plaintext
int DayNumberOfYear(date)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>The input date</td>
<td>Date</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

`DayNumberOfYear([Reservation Date])` returns 349 when the date in [Reservation Date] is 15 December 2001.

---

**6.1.3.7 DaysBetween**

**Description**

Returns the number of days between two dates

**Function Group**
Date and Time

**Syntax**

```plaintext
int DaysBetween(first_date;last_date)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>first_date</td>
<td>The first date</td>
<td>Date</td>
<td>Yes</td>
</tr>
<tr>
<td>last_date</td>
<td>The last date</td>
<td>Date</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

`DaysBetween([Sale Date];[Invoice Date])` returns 2 if [Sale Date] is 15 December 2001 and [Invoice Date] is 17 December 2001.

---

**6.1.3.8 LastDayOfMonth**
**Description**
Returns the date of the last day in a month

**Function Group**
Date and Time

**Syntax**
date LastOfMonth(date)

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Any date in the month</td>
<td>Date</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**
LastOfMonth([Sale Date]) returns 31 December 2005 if [Sale Date] is 11 December 2005.

### 6.1.3.9 LastOfWeek

**Description**
Returns the date of the last day in a week

**Function Group**
Date and Time

**Syntax**
date LastOfWeek(date)

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Any date in the week</td>
<td>Date</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
The function treats Monday as the first day of the week.

**Examples**
LastOfWeek([Sale Date]) returns 15 May 2005 (a Sunday) if [Sale Date] is 11 May 2005.
6.1.3.10 Month

**Description**
Returns the month name in a date

**Function Group**
Date and Time

**Syntax**
```
string Month(date)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>The input date</td>
<td>Date</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

`Month([Reservation Date])` returns "December" when the date in [Reservation Date] is 15 December 2005.

6.1.3.11 MonthNumberOfYear

**Description**
Returns the month number in a date

**Function Group**
Date and Time

**Syntax**
```
int MonthNumberOfYear(date)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Any date in the year</td>
<td>Date</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example**

`MonthNumberOfYear([Reservation Date])` returns 12 when the date in [Reservation Date] is 15 December 2005.
**6.1.3.12 MonthsBetween**

**Description**
Returns the number of months between two dates

**Function Group**
Date and Time

**Syntax**
```
int MonthsBetween(first_date;last_date)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>first_date</td>
<td>The first date</td>
<td>Date</td>
<td>Yes</td>
</tr>
<tr>
<td>last_date</td>
<td>The last date</td>
<td>Date</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

`MonthsBetween([Sale Date];[Invoice Date])` returns 1 if `[Sale Date]` is 2 December 2005 and `[Invoice Date]` is 2 January 2006.

---

**6.1.3.13 Quarter**

**Description**
Returns the quarter number in a date

**Function Group**
Date and Time

**Syntax**
```
int Quarter(date)
```
Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Any date in the quarter</td>
<td>Date</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Examples
Quarter([Reservation Date]) returns 4 when the date in [Reservation Date] is 15 December 2005.

6.1.3.14 RelativeDate

Description
Returns a date relative to another date

Function Group
Date and Time

Syntax
\[ \text{date RelativeDate(start date;num days)} \]

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_date</td>
<td>The start date</td>
<td>Date</td>
<td>Yes</td>
</tr>
<tr>
<td>num_days</td>
<td>The number of days from the start date</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Notes
The num_days parameter can be negative to return a date earlier than start_date.

Examples
RelativeDate([Reservation Date];2) returns 17 December 2005 when [Reservation Date] is 15 December 2005.

RelativeDate([Reservation Date];-3) returns 9 January 2007 when [Reservation Date] is 12 January 2007.

6.1.3.15 TimeDim
Description
The `TimeDim` time dimension allows you to build a time axis from a date type universe object. `TimeDim` returns the data for the dates given as the first parameter over the time periods given as the second parameter. When there are periods that have no data, the first day of each empty period is returned. This ensures a full axis for the given period. This guarantees:

- That the axis retains the natural time order (oldest objects first, the most recent objects last).
- The axis contains all the periods between the minimum and maximum dates in the current context.

Function Group
Date and Time

Syntax
`TimeDim([Date Type]; Period Type)`

Input
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date Type</td>
<td>The date object for the report, for example, InvoiceDate.</td>
<td>Date</td>
<td>Yes</td>
</tr>
<tr>
<td>Period Type</td>
<td>The period for the results, from the following values:</td>
<td>Pre-defined</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>• DayPeriod</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• MonthPeriod</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• QuarterPeriod</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• YearPeriod</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When no value is selected, the DayPeriod is used by default. This object should be a data provider object, it must be available from report objects, and cannot be a variable.

Use the above function in conjunction with the following functions:

- DayName
- DayNumberOfMonth
- DayNumberOfWeek
- DayNumberOfYear
- Month
- MonthNumberOfYear
- Quarter
- Year
- FormatDate

Example
The first table below contains data that concerns only certain dates. The query examples that follow show how the results are interpreted.
The following formula `DayName(TimeDim([Invoice Date] ; QuarterPeriod)` returns daily values from the above table.

### Invoice Date | Revenue  
---|---  
1/3/00 | 31,607  
1/8/00 | 31,244  
7/3/00 | 38,154

You should format the results of the TimeDim function with the Quarter function to return the results by Quarter (Q1, Q2...) to give you the following result table:

### Invoice Date | Revenue  
---|---  
Q1 | 62,851  
Q2 |   
Q3 | 38,154

### 6.1.3.16 ToDate

**Description**  
Returns a character string formatted according to a date format

**Function Group**  
Date and Time

**Syntax**  
`date ToDate(date_string;format)`
### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>date_string</td>
<td>The date to be formatted</td>
<td>string</td>
<td>Yes</td>
</tr>
<tr>
<td>format</td>
<td>The date format</td>
<td>string</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### Examples

ToDate("15/12/2002";"dd/MM/yyyy") returns 15/12/2002.

### 6.1.3.17 Week

#### Description

Returns the week number in the year

#### Function Group

Date and Time

#### Syntax

```plaintext
int Week(date)
```

#### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>The input date</td>
<td>Date</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### Examples

Week([Reservation Date]) returns 1 when the date in [Reservation Date] is 4 January 2004 (which occurs in the first week of the year 2004).

### 6.1.3.18 Year

#### Description

Returns the year in a date

#### Function Group

Date and Time
**Syntax**

```markdown
int Year(date)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>The input date</td>
<td>Date</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

`Year([Reservation Date])` returns 2005 when the date in `[Reservation Date]` is 15 December 2005.

---

### 6.1.4 Data Provider functions

---

#### 6.1.4.1 Connection

**Description**

Returns the parameters of the database connection used by a data provider

**Function Group**

Data Provider

**Syntax**

```markdown
string Connection(dp)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>The data provider</td>
<td>Data provider</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

- You must enclose the name of the data provider in square brackets.
- For security reasons, the output of the function does not include the database host name, user name and user password.
### 6.1.4.2 DataProvider

**Description**
Returns the name of the data provider containing a report object

**Function Group**
Data Provider

**Syntax**
```plaintext
string DataProvider(obj)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>A report object</td>
<td>Report object</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
DataProvider([Total Revenue]) returns "Sales" if the [Total Revenue] measure is in a data provider called "Sales".
```

### 6.1.4.3 DataProviderKeyDate

**Description**
Returns the keydate of a data provider

**Function Group**
Data Provider

**Syntax**
```plaintext
date DataProviderKeyDate(dp)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>The data provider</td>
<td>Data provider</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

- You must enclose the name of the data provider in square brackets.
- The returned keydate is formatted according to the document locale.
Examples
DataProviderKeyDate([Sales]) returns 3 August 2007 if the keydate for the Sales data provider is 3 August 2007.

6.1.4.4 DataProviderKeyDateCaption

Description
Returns the keydate caption of a data provider

Function Group
Data Provider

Syntax
string DataProviderKeyDateCaption(dp)

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>The data provider</td>
<td>Data provider</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Notes
You must enclose the name of the data provider in square brackets.

Examples
DataProviderKeyDateCaption([Sales]) returns "Current calendar date" if the keydate caption in the Sales data provider is "Current calendar date".

6.1.4.5 DataProviderSQL

Description
Returns the SQL generated by a data provider

Function Group
Data Provider

Syntax
string DataProviderSQL(dp)
### 6.1.4.6 DataProviderType

**Description**
Returns the type of a data provider

**Function Group**
Data Provider

**Syntax**

```
string DataProviderType(dp)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>The data provider</td>
<td>Data provider</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

- `DataProviderType` returns "Universe" for universe data providers or "Personal data" for personal data providers.
- You must enclose the name of the data provider in square brackets.

**Examples**

`DataProviderType([Sales])` returns "Universe" if the "Sales" data provider is based on a universe.

### 6.1.4.7 IsPromptAnswered
**Description**
 Determines whether a prompt has been answered

**Function Group**
 Data Provider

**Syntax**
```plaintext
bool IsPromptAnswered([dp];prompt_string)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>The data provider containing the prompt</td>
<td>Data provider</td>
<td>No</td>
</tr>
<tr>
<td>prompt_string</td>
<td>The prompt text</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
 You must enclose the name of the data provider in square brackets.

**Examples**

IsPromptAnswered("Choose a city") returns true if the prompt identified by the text "Choose a city" has been answered.

IsPromptAnswered([Sales];"Choose a city") returns true if the prompt identified by the text "Choose a city" in the [Sales] data provider has been answered.

---

**6.1.4.8 LastExecutionDate**

**Description**
 Returns the date on which a data provider was last refreshed

**Function Group**
 Data Provider

**Syntax**
```plaintext
date LastExecutionDate(dp)
```
### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>The data provider</td>
<td>Data provider</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### Notes
- If your report has one data provider only you can omit the `dp` parameter.
- You must enclose the name of the data provider in square brackets.
- You can use the `DataProvider` function to provide a reference to a data provider.

#### Examples

**LastExecutionDate([Sales Query])** returns "3/4/2002" if the Sales Query data provider was last refreshed on 4 March 2002.

#### Related Topics
- `DataProvider`

---

### 6.1.4.9 LastExecutionDuration

#### Description

Returns the time in seconds taken by the last refresh of a data provider.

#### Function Group

Data Provider

#### Syntax

```
num LastExecutionDuration(dp)
```

#### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>The data provider</td>
<td>Data provider</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### Notes

You must enclose the name of the data provider in square brackets.

#### Examples

**LastExecutionDuration([Sales])** returns 3 if the "Sales" data provider took 3 second to return its data the last time it was run.
6.1.4.10 LastExecutionTime

**Description**
Returns the time at which a data provider was last refreshed

**Function Group**
Data Provider

**Syntax**
```
time LastExecutionTime(dp)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>The data provider</td>
<td>Data provider</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

- If your report has one data provider only you can omit the dp parameter.
- You can use the DataProvider function to provide a reference to a data provider.
- You must enclose the name of the data provider in square brackets.

**Examples**
```
LastExecutionTime([Sales Query]) returns "2:48:00 PM" if the Sales Query data provider was last refreshed at 2:48:00 PM.
```

**Related Topics**
- DataProvider

6.1.4.11 NumberOfDataProviders

**Description**
Returns the number of data providers in a report

**Function Group**
Data Provider

**Syntax**
```
int NumberOfDataProviders()
```
Examples
NumberOfDataProviders() returns 2 if the report has two data providers.

6.1.4.12 NumberOfRows

Description
Returns the number of rows in a data provider

Function Group
Data Provider

Syntax
int NumberOfRows(dp)

Input
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>The data provider</td>
<td>Data provider</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Notes
• You must enclose the name of the data provider in square brackets.
• You can use the DataProvider function to provide a reference to a data provider.

Examples
NumberOfRows([Query 1]) returns 10 if the "Query 1" data provider has 10 rows.

Related Topics
• DataProvider

6.1.4.13 RefValueDate

Description
Returns the date of the reference data used for data tracking

Function Group
Data Provider
Syntax

\texttt{date RefValueDate()}

Examples

\texttt{RefValueDate()} returns 15 December 2008 if the reference date is 15 December 2008.

\section*{6.1.4.14 \texttt{RefValueUserResponse}}

\subsection*{Description}

Returns the response to a prompt when the reference data was the current data.

\subsection*{Function Group}

Data Provider

\subsection*{Syntax}

\texttt{string RefValueUserResponse([dp;]prompt_string[,Index])}

\subsection*{Input}

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>The data provider</td>
<td>Data provider</td>
<td>No</td>
</tr>
<tr>
<td>prompt_string</td>
<td>The prompt text</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>Index</td>
<td>Tells the function to return the database primary keys of the prompt values</td>
<td>Keyword</td>
<td>No</td>
</tr>
</tbody>
</table>

\subsection*{Notes}

- The function returns an empty string if data tracking is not activated.
- You must enclose the name of the data provider in square brackets.
- You can use the \texttt{DataProvider} function to provide a reference to a data provider.
- If you selected more than one value in answer to a prompt, the function returns a string consisting of a list of values (or primary keys if the \texttt{Index} operator is specified) separated by semi-colons.

\subsection*{Examples}

\texttt{RefValueUserResponse("Which city?")} returns "Los Angeles" if you entered "Los Angeles" in the "Which City?" prompt at the time when the reference data was the current data.

\texttt{RefValueUserResponse([Sales Query];"Which city?")} returns "Los Angeles," if you entered "Los Angeles" in the "Which City?" prompt in the "Sales Query" data provider at the time when the reference data was the current data.
### 6.1.4.15 UniverseName

**Description**
Returns the name of the universe on which a data provider is based.

**Function Group**
Data Provider

**Syntax**
```
string UniverseName(dp)
```

**Input**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>The data provider</td>
<td>Data provider</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
- The value of `dp` in the formula is automatically updated if the name of the data provider changes. If the data provider is renamed to "Q1", the formula becomes `UniverseName([Q1])`.
- You must enclose the name of the data provider in square brackets.
- You can use the `DataProvider` function to provide a reference to a data provider.

**Examples**
`UniverseName([Query 1])` returns "eFashion" if the [Query 1] data provider is based on the eFashion universe.

**Related Topics**
- `DataProvider`

### 6.1.4.16 UserResponse

**Description**
Returns the response to a prompt.

**Function Group**
Data Provider
**Syntax**

```plaintext
string UserResponse([dp;prompt_string[Index]])
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>The data provider</td>
<td>Data provider</td>
<td>No</td>
</tr>
<tr>
<td>prompt_string</td>
<td>The prompt text</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>Index</td>
<td>Tells the function to return the database primary keys of the prompt values</td>
<td>Keyword</td>
<td>No</td>
</tr>
</tbody>
</table>

**Notes**

- You must enclose the name of the data provider in square brackets.
- You can use the `DataProvider` function to provide a reference to a data provider.
- If you select more than one value in answer to a prompt, the function returns a string consisting of a list of values (or primary keys if the `Index` operator is specified) separated by semi-colons.

**Examples**

UserResponse("Which city?") returns "Los Angeles if you entered "Los Angeles" in the "Which City?" prompt.

UserResponse([Sales Query];"Which city?") returns "Los Angeles," if you entered "Los Angeles" in the "Which City?" prompt in the "Sales Query" data provider.

UserResponse([Sales Query];"Which city?";Index) returns 23 if you entered "Los Angeles" in the "Which City?" prompt in the "Sales Query" data provider, and the database primary key of Los Angeles is 23.

### 6.1.5 Document functions

#### 6.1.5.1 DocumentAuthor

**Description**

Returns the InfoView logon of the document creator

**Function Group**

Document
6.1.5.2 DocumentCreationDate

Description
Returns the date on which a document was created

Function Group
Document

Syntax
```markdown
date DocumentCreationDate()
```

Examples
```markdown
DocumentCreationDate() returns 15 December 2008 if the document was created on 15 December 2008.
```

6.1.5.3 DocumentCreationTime

Description
Returns the time when a document was created

Function Group
Document

Syntax
```markdown
time DocumentCreationTime()
```

Examples
```markdown
DocumentCreationTime() returns 11:15 if the document was created at 11:15.
```

6.1.5.4 DocumentDate
**Description**

Returns the date on which a document was last saved

**Function Group**

Document

**Syntax**

`date DocumentDate()`

**Examples**

`DocumentDate()` returns 8 August 2005 if the document was last saved on 8 August 2005.

---

### 6.1.5.5 DocumentName

**Description**

Returns the document name

**Function Group**

Document

**Syntax**

`string DocumentName()`

**Examples**

`DocumentName()` returns "Sales Report" if the document is called "Sales Report".

---

### 6.1.5.6 DocumentOwner

**Description**

Returns the InfoView logon/user name of the owner of the document (the last person who saved the document). (To return the original author/creator of the document, use the DocumentAuthor function.)

**Function Group**

Document

**Syntax**

`string DocumentOwner()`
Examples

DocumentOwner() returns "gkn" if the last person who saved the document has the user name or login "gkn".

6.1.5.7 DocumentPartiallyRefreshed

Description
Determines whether a document is partially refreshed

Function Group
Document

Syntax

bool DocumentPartiallyRefreshed()

Notes
DocumentPartiallyRefreshed returns a boolean value that you can use in the If function.

Examples

DocumentPartiallyRefreshed() returns True if the document is partially refreshed.

6.1.5.8 DocumentTime

Description
Returns the time when a document was last saved

Function Group
Document

Syntax

time DocumentTime()

Notes
The format of the returned time varies depending on the cell format.

Example

DocumentTime() returns 15:45 if the document was last saved at 15:45.
### 6.1.5.9 DrillFilters

**Description**
Returns the drill filters applied to a document or object in drill mode

**Function Group**
Document

**Syntax**

```
string DrillFilters(obj|separator)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>A report object</td>
<td>Report object</td>
<td>Either obj or separator required</td>
</tr>
<tr>
<td>separator</td>
<td>The drill filter separator</td>
<td>String</td>
<td>Either obj or separator required</td>
</tr>
</tbody>
</table>

**Notes**
- You can insert `DrillFilters` directly without the need to enter the formula manually by inserting a DrillFilters cell.
- If you do not specify an object, the function returns all drill filters applied to the document.

**Examples**

DrillFilters() returns "US" if the document has a drill filter restricting the [Country] object to US.


DrillFilters ([Quarter]) returns "Q3" if the document has a drill filter restricting [Quarter] to "Q3".

### 6.1.5.10 PromptSummary

**Description**
Returns the prompt text and user response of all prompts in a document
**Function Group**

**Document**

**Syntax**

```
string PromptSummary()
```

**Examples**

Example output of the `PromptSummary` function appears as follows:

```
Enter Quantity Sold: 5000
Enter value(s) for State (optional): California, Texas, Utah
Enter Customer (optional):
```

### 6.1.5.11 QuerySummary

**Description**

Returns information about the queries in a document

**Function Group**

**Document**

**Syntax**

```
string QuerySummary([dp])
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>A data provider</td>
<td>Data provider</td>
<td>No</td>
</tr>
</tbody>
</table>

**Notes**

- You must enclose the name of the data provider in square brackets.

**Examples**

`QuerySummary()` returns information about all the queries in a document.

`QuerySummary([Query 1])` returns information about the queries based on the [Query 1] data provider.

**Output example:**

```
Query 1:
    Universe: eFashion
    Last execution time: 1s
    NB of rows: 34500
    Result objects: State, Year, Sales Revenue
    Scope of analysis: State, City, Year, Quarter, Month
    Filters: (State inlist("US","France");
    And (Sales Revenue Greater Than 1000000
```

Functions, operators and keywords
6.1.5.12 ReportFilter

**Description**
Returns the report filters applied to an object or report

**Function Group**
Document

**Syntax**
string ReportFilter(obj)

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>A report object</td>
<td>Report object</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**
ReportFilter([Country]) returns "US" if there is a report filter on the Country object that restricts it to "US".

6.1.5.13 ReportFilterSummary

**Description**
Returns a summary of the report filters in a document or report

**Function Group**
Document

**Syntax**
string ReportFilterSummary(report_name)
Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>report_name</td>
<td>The name of the report</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes

If `report_name` is omitted, `ReportFilterSummary` returns a summary of all the report filters in all the reports in the document.

Examples

`ReportFilterSummary()` returns information about all the report filters in a document.


Example output of the `ReportFilterSummary` function appears as follows:

```
Filters on Report1:
(Sales Revenue Greater Than 1000000
 Or (Sales Revenue Less Than 3000))
Filters on Section on City:
(City InList("Los Angeles";"San Diego";))
Ranking Filter:
(Top 10 & Bottom 10 [Customer] Based on [Sales Revenue] (Count))
```

6.1.6 Logical functions

6.1.6.1 Even

Description

Determines whether a number is even

Function Group

Logical

Syntax

```plaintext
bool Even(number)
```
Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Any number</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Notes

- Even returns a boolean value that you can use in the If function.
- If you place Even directly into a column, it returns an integer (1=true; 0=false). You can format this integer using a Boolean number format.

Examples

Even(4) returns True.
Even(3) returns False.
Even(23.2) returns False.
Even(-4) returns True.
Even(-2.2) returns False.

6.1.6.2 IsDate

Description
Determines whether a value is a date

Function Group
Logical

Syntax
bool IsDate(obj)

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>Any report object</td>
<td>Report object</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Notes

- IsDate returns a boolean value that you can use in the If function.
- If you place IsDate directly into a column, it returns an integer (1=true; 0=false). You can format this integer using a Boolean number format.

Examples

IsDate([Reservation Date]) returns True if [Reservation Date] is a date.
If(IsDate([Reservation Date]) Then "Date" Else "Not a date" returns "Date" if [Reservation Date] is a date.

**Related Topics**
- If...Then...Else

### 6.1.6.3 IsError

**Description**
Determines whether an object returns an error

**Function Group**
Logical

**Syntax**
bool IsError(obj)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>Any report object</td>
<td>Report object</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
- IsError returns a boolean value that you can use in the If function.
- If you place IsError directly into a column, it returns an integer (1=true; 0=false). You can format this integer using a Boolean number format.

**Examples**
IsError([Revenue]) returns False if the [Revenue] variable does not return an error.

IsError([Average Guests]) returns True if the [Average Guests] variable returns a division by zero (#DIV/0) error.

If IsError([Average Guests]) Then "Error" Else "No error" returns "Error" if the [Average Guests] variable returns a division by zero (#DIV/0) error.

**Related Topics**
- If...Then...Else
6.1.6.4 IsLogical

**Description**
Determines whether a value is boolean

**Function Group**
Logical

**Syntax**
```
bool IsLogical(obj)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>Any report object</td>
<td>Report object</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

- IsLogical returns a boolean value that you can use in the If function.
- If you place IsLogical directly into a column, it returns an integer (1=true; 0=false). You can format this integer using a Boolean number format.

**Examples**

IsLogical(IsString([Country])) returns True.

IsLogical([Country]) returns False if country returns any data type other than boolean.

If IsLogical(IsDate([Country])) Then "Boolean" Else "Not boolean" returns "Boolean".

**Related Topics**

- If...Then...Else

---

6.1.6.5 IsNull

**Description**
Determines whether a value is null

**Function Group**
Logical
Syntax

```c
bool IsNull(obj)
```

### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>Any report object</td>
<td>Report object</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Notes

- `IsNull` returns a boolean value that you can use in the `If` function.
- If you place `IsNull` directly into a column, it returns an integer (1=true; 0=false). You can format this integer using a Boolean number format.

### Examples

- `IsNull([Revenue])` returns False if the [Revenue] variable is not null.
- `IsNull([Average Guests])` returns True if the [Average Guests] variable is null.

### Related Topics

- `If...Then...Else`

---

### 6.1.6.6 IsNumber

#### Description

Determines whether a value is a number

#### Function Group

Logical

#### Syntax

```c
bool IsNumber(obj)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>Any report object</td>
<td>Report object</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### Notes

- `IsNumber` returns a boolean value that you can use in the `If` function.
- If you place `IsNumber` directly into a column, it returns an integer (1=true; 0=false). You can format this integer using a Boolean number format.
Examples
IsNumber([Revenue]) returns True if the [Revenue] variable is a number.
IsNumber([Customer Name]) returns False if the [Customer Name] variable is not a number.
If IsNumber([Customer Name]) Then "Number" Else "Not a number" returns "Not a number" if the [Customer Name] variable is not a number.

Related Topics
• If...Then...Else

6.1.6.7 IsString

Description
Determines whether a value is a string

Function Group
Logical

Syntax
bool IsString(obj)

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>Any report object</td>
<td>Report object</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Notes
• IsString returns a boolean value that you can use in the If function.
• If you place IsString directly into a column, it returns an integer (1=true; 0=false). You can format this integer using a Boolean number format.

Examples
IsString([Revenue]) returns false if the [Revenue] variable is not a string.
IsString([Customer Name]) returns true if the [Customer Name] variable is a string.
If IsString([Customer Name]) Then "String" Else "Not a string" returns "String" if the [Customer Name] variable is a string.

Related Topics
• If...Then...Else
### 6.1.6.8 IsTime

**Description**
Determines whether a variable is a time variable

**Function Group**
Logical

**Syntax**
```
bool IsTime(obj)
```

**Input**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>Any report object</td>
<td>Report object</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
- `IsTime` returns a boolean value that you can use in the If function.
- If you place `IsTime` directly into a column, it returns an integer (1=true; 0=false). You can format this integer using a Boolean number format.

**Examples**
- `IsTime([Reservation Time])` returns true if the [Reservation Time] variable is a time variable.
- `IsTime([Average Guests])` returns false if the [Average Guests] variable is not a time variable.
- If `IsTime([Average Guests])` Then "Time" Else "Not time" returns "Not time" if the [Average Guests] variable is not a time variable.

**Related Topics**
- If...Then...Else

### 6.1.6.9 Odd

**Description**
Determines whether a number is odd

**Function Group**
Logical
Syntax

bool Odd(number)

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Any number</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Notes

- Odd returns a boolean value that you can use in the If function.
- If you place Odd directly into a column, it returns an integer (1=true; 0=false). You can format this integer using a Boolean number format.
- Odd ignores the fractional parts of decimal numbers.

Examples

Odd(5) returns True.

Odd(4) returns False.

Odd(23.2) returns True.

Odd(24.2) returns True.

Odd(-23.2) returns True.

Odd(-24.2) returns True.

Related Topics

- If...Then...Else

6.1.7 Numeric functions

6.1.7.1 Abs

Description

Returns the absolute value of a number

Function Group

Numeric
### Syntax

```
num Abs(number)
```

### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Any number</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Examples

Abs(25) returns 25.

Abs(-11) returns 11.

---

### 6.1.7.2 Ceil

#### Description

Returns a number rounded up to the nearest integer

#### Function Group

Numeric

#### Syntax

```
num Ceil(number)
```

### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Any number</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Examples

Ceil(2.4) returns 3.

Ceil(3.1) returns 4.

Ceil(-3.1) returns -3.

---

### 6.1.7.3 Cos

#### Description

Returns the cosine of an angle
**Function Group**
Numeric

**Syntax**

\[ \text{num } \cos(\text{angle}) \]

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>An angle in radians</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example**

\[ \cos(180) \text{ returns } -0.6. \]

6.1.7.4 **EuroConvertFrom**

**Description**

Converts a Euro amount to another currency

**Function Group**
Numeric

**Syntax**

\[ \text{num } \text{EuroConvertFrom}(\text{euro\_amount};\text{curr\_code};\text{round\_level}) \]

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>euro_amount</td>
<td>The amount in Euros</td>
<td>Number</td>
<td>Yes</td>
</tr>
<tr>
<td>curr_code</td>
<td>The ISO code of the target currency</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>round_level</td>
<td>The number of decimal places to which the result is rounded</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

The currency code must be the code of one of the 12 EU currencies whose values were fixed in relation to the Euro prior to their abolition in January 2002. If it is not, the function returns #ERROR. The currencies are:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEF</td>
<td>Belgian franc</td>
</tr>
<tr>
<td>DEM</td>
<td>German mark</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>----------------------</td>
</tr>
<tr>
<td>GRD</td>
<td>Greek drachma</td>
</tr>
<tr>
<td>ESP</td>
<td>Spanish peseta</td>
</tr>
<tr>
<td>FRF</td>
<td>French franc</td>
</tr>
<tr>
<td>IEP</td>
<td>Irish punt</td>
</tr>
<tr>
<td>ITL</td>
<td>Italian lira</td>
</tr>
<tr>
<td>LUF</td>
<td>Luxembourg franc</td>
</tr>
<tr>
<td>NLG</td>
<td>Dutch guilder</td>
</tr>
<tr>
<td>ATS</td>
<td>Austrian schilling</td>
</tr>
<tr>
<td>PTS</td>
<td>Portuguese escudo</td>
</tr>
<tr>
<td>FIM</td>
<td>Finnish mark</td>
</tr>
</tbody>
</table>

**Examples**

EuroConvertFrom(1000;"FRF";2) returns 6559.57.

EuroConvertFrom(1000;"FRF";1) returns 6559.60.

EuroConvertFrom(1000.04;"DEM";2) returns 1955.83.

EuroConvertFrom(1000.04;"DEM";1) returns 1955.80.

**Related Topics**

- Rounding and truncating numbers

---

**6.1.7.5 EuroConvertTo**

**Description**

Converts an amount to Euros

**Function Group**

Numeric

**Syntax**

num EuroConvertTo(noneuro_amount;curr_code;round_level)
**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>euro_amount</td>
<td>The amount in the non-euro currency</td>
<td>Number</td>
<td>Yes</td>
</tr>
<tr>
<td>curr_code</td>
<td>The ISO code of the non-euro currency</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>round_level</td>
<td>The number of decimal places to which the result is rounded</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example**

EuroConvertTo(6559;"FRF";2) returns 999.91.

EuroConvertTo(6559;"FRF";1) returns 999.90.

EuroConvertTo(1955;"DEM";2) returns 999.58.

EuroConvertTo(1955;"DEM";1) returns 999.60.

**Note**

The currency code must be the code of one of the 12 EU currencies whose values were fixed in relation to the Euro prior to their abolition in January 2002. If it is not, the function returns #ERROR. The currencies are:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEF</td>
<td>Belgian franc</td>
</tr>
<tr>
<td>DEM</td>
<td>German mark</td>
</tr>
<tr>
<td>GRD</td>
<td>Greek drachma</td>
</tr>
<tr>
<td>ESP</td>
<td>Spanish peseta</td>
</tr>
<tr>
<td>FRF</td>
<td>French franc</td>
</tr>
<tr>
<td>IEP</td>
<td>Irish punt</td>
</tr>
<tr>
<td>ITL</td>
<td>Italian lira</td>
</tr>
<tr>
<td>LUF</td>
<td>Luxembourg franc</td>
</tr>
<tr>
<td>NLG</td>
<td>Dutch guilder</td>
</tr>
<tr>
<td>ATS</td>
<td>Austrian schilling</td>
</tr>
<tr>
<td>PTS</td>
<td>Portuguese escudo</td>
</tr>
<tr>
<td>FIM</td>
<td>Finnish mark</td>
</tr>
</tbody>
</table>

**Related Topics**

- Rounding and truncating numbers
6.1.7.6 EuroFromRoundError

Description
Returns the rounding error in a conversion from Euros

Function Group
Numeric

Syntax
num EuroFromRoundError(euro_amount;curr_code;round_level)

Input
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>euro_amount</td>
<td>The amount in Euros</td>
<td>Number</td>
<td>Yes</td>
</tr>
<tr>
<td>curr_code</td>
<td>The ISO code of the target currency</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>round_level</td>
<td>The number of decimal places to which the result is rounded</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Output
The rounding error in the calculation

Examples
EuroFromRoundErr(1000;"FRF";2) returns 0. (There is no difference between the unrounded conversion and the conversion rounded to 2 decimal places.)

EuroFromRoundErr(1000;"FRF";1) returns 0.03. (The unrounded conversion is 6559.57. The conversion rounded to 1 decimal place is 6559.60. The rounding error is 0.03.)

EuroFromRoundErr(1000;"DEM";2) returns 0. (There is no difference between the unrounded conversion and the conversion rounded to 2 decimal places.)

EuroFromRoundErr(1000;"DEM";1) returns -0.01. (The unrounded conversion is 1955.83. The conversion rounded to 1 decimal place is 1955.80. The rounding error is -0.03.)

Note
The currency code must be the code of one of the 12 EU currencies whose values were fixed in relation to the Euro prior to their abolition in January 2002. If it is not, the function returns #ERROR. The currencies are:

<table>
<thead>
<tr>
<th>Currency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEF</td>
<td>Belgian franc</td>
</tr>
<tr>
<td>DEM</td>
<td>German mark</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------------</td>
</tr>
<tr>
<td>GRD</td>
<td>Greek drachma</td>
</tr>
<tr>
<td>ESP</td>
<td>Spanish peseta</td>
</tr>
<tr>
<td>FRF</td>
<td>French franc</td>
</tr>
<tr>
<td>IEP</td>
<td>Irish punt</td>
</tr>
<tr>
<td>ITL</td>
<td>Italian lira</td>
</tr>
<tr>
<td>LUF</td>
<td>Luxembourg franc</td>
</tr>
<tr>
<td>NLG</td>
<td>Dutch guilder</td>
</tr>
<tr>
<td>ATS</td>
<td>Austrian schilling</td>
</tr>
<tr>
<td>PTS</td>
<td>Portuguese escudo</td>
</tr>
<tr>
<td>FIM</td>
<td>Finnish mark</td>
</tr>
</tbody>
</table>

**Related Topics**

- [Rounding and truncating numbers](#)

### 6.1.7.7 EuroToRoundError

**Description**

Returns the rounding error in a conversion to Euros

**Function Group**

Numeric

**Syntax**

```plaintext
um EuroToRoundError(noneuro_amount;curr_code;round_level)
```
Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>euro_amount</td>
<td>The amount in the non-euro currency</td>
<td>Number</td>
<td>Yes</td>
</tr>
<tr>
<td>curr_code</td>
<td>The ISO code of the non-euro currency</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>round_level</td>
<td>The number of decimal places to which the result is rounded</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Examples

EuroToRoundErr(6559;"FRF";2) returns 0. (There is no difference between the unrounded conversion and the conversion rounded to 2 decimal places.)

EuroToRoundErr(6559;"FRF";1) returns -0.01. (The unrounded conversion is 999.91. The conversion rounded to 1 decimal place is 999.90. The rounding error is -0.01.)

EuroToRoundErr(1955;"DEM";2) returns 0. (There is no difference between the unrounded conversion and the conversion rounded to 2 decimal places.)

EuroToRoundErr(1955;"DEM";1) returns 0.02. (The unrounded conversion is 999.58. The conversion rounded to 1 decimal place is 999.60. The rounding error is 0.02.)

Note

The currency code must be the code of one of the 12 EU currencies whose values were fixed in relation to the Euro prior to their abolition in January 2002. If it is not, the function returns #ERROR. The currencies are:

<table>
<thead>
<tr>
<th>BEF</th>
<th>Belgian franc</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEM</td>
<td>German mark</td>
</tr>
<tr>
<td>GRD</td>
<td>Greek drachma</td>
</tr>
<tr>
<td>ESP</td>
<td>Spanish peseta</td>
</tr>
<tr>
<td>FRF</td>
<td>French franc</td>
</tr>
<tr>
<td>IEP</td>
<td>Irish punt</td>
</tr>
<tr>
<td>ITL</td>
<td>Italian lira</td>
</tr>
<tr>
<td>LUF</td>
<td>Luxembourg franc</td>
</tr>
<tr>
<td>NLG</td>
<td>Dutch guilder</td>
</tr>
<tr>
<td>ATS</td>
<td>Austrian schilling</td>
</tr>
<tr>
<td>PTS</td>
<td>Portuguese escudo</td>
</tr>
<tr>
<td>FIM</td>
<td>Finnish mark</td>
</tr>
</tbody>
</table>
Related Topics
• Rounding and truncating numbers

6.1.7.8 Exp

Description
Returns an exponential (e raised to a power)

Function Group
Numeric

Syntax
num Exp(power)

Input
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>power</td>
<td>The power</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Notes
An exponential is the constant e (2.718...) raised to a power.

Examples
Exp(2.2) returns 9.03.

6.1.7.9 Fact

Description
Returns the factorial of a number

Function Group
Numeric

Syntax
int Fact(number)
**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Any number</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

The factorial of \( \text{number} \) is the product of all the integers from 1 to \( \text{number} \).

**Examples**

- \( \text{Fact(4)} \) returns 24.
- \( \text{Fact(5.9)} \) returns 120.

---

### 6.1.7.10 Floor

**Description**

Returns a number rounded down to the nearest integer

**Function Group**

Numeric

**Syntax**

```plaintext
int Floor(number)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Any number</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example**

- \( \text{Floor(24.4)} \) returns 24.

---

### 6.1.7.11 Interpolation

**Description**

Calculates empty measure values by interpolation

**Function Group**

Numeric
Syntax

```plaintext
num Interpolation(measure[;PointToPoint|Linear]
[;NotOnBreak|reset_dims][;Row|Col])
```

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
</tbody>
</table>
| PointToPoint|Linear               | The interpolation method:  
• PointToPoint - point-to-point interpolation  
• Linear - linear regression with least squares interpolation | Keyword    | No         |
| NotOnBreak|reset_dims       |  
• NotOnBreak - prevents the function from resetting the calculation on block and section breaks  
• reset_dims - the list of dimensions used to reset the interpolation | Keyword| dimension list | No |
| Row|Col               | Sets the calculation direction                    | Keyword    | (Row is default) |

Notes

- **Interpolation** is particularly useful when you create a line graph on a measure that contains missing values. By using the function you ensure that the graph plots a continuous line rather than disconnected lines and points.
- Linear regression with least squares interpolation calculates missing values by calculating a line equation in the form \( f(x) = ax + b \) that passes as closely as possible through all the available values of the measure.
- Point-to-point interpolation calculates missing values by calculating a line equation in the form \( f(x) = ax + b \) that passes through the two adjacent values of the missing value.
- The sort order of the measure impacts the values returned by **Interpolation**.
- You cannot apply a sort or a ranking to a formula containing **Interpolation**.
- If there is only one value in the list of values, **Interpolation** uses this value to supply all the missing values.
- Filters applied to an interpolated measure can change the values returned by **Interpolation** depending on which values the filter impacts.

Examples

**Interpolation([Value])** supplies the following missing values using the default point-to-point interpolation method:

<table>
<thead>
<tr>
<th>Day</th>
<th>Value</th>
<th>Interpolation([Value])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Day</td>
<td>Value</td>
<td>Interpolation([Value])</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Tuesday</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Wednesday</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Thursday</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Friday</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>Saturday</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Sunday</td>
<td>19</td>
<td>19</td>
</tr>
</tbody>
</table>

Related Topics

- Linear operator
- PointToPoint operator

### 6.1.7.12 Ln

**Description**

Returns the natural logarithm of a number

**Function Group**

Numeric

**Syntax**

```
num Ln(number)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Any number</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

\[
\text{Ln}(10) \text{ returns } 2.
\]

### 6.1.7.13 Log
**Description**
Returns the logarithm of a number in a specified base

**Function Group**
Numeric

**Syntax**
\[\text{num } \text{Log}(\text{number};\text{base})\]

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Any number</td>
<td>Number</td>
<td>Yes</td>
</tr>
<tr>
<td>base</td>
<td>The base of the logarithm</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**
\( \text{Log}(125;5) \) returns 3.

---

**6.1.7.14 Log10**

**Description**
Returns the base 10 logarithm of a number

**Function Group**
Numeric

**Syntax**
\[\text{num } \text{Log10}(\text{number})\]

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Any number</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**
\( \text{Log10}(100) \) returns 2.
**Description**
Returns the remainder from the division of two numbers

**Function Group**
Numeric

**Syntax**
```
num Mod(dividend;divisor)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dividend</td>
<td>The dividend</td>
<td>Number</td>
<td>Yes</td>
</tr>
<tr>
<td>divisor</td>
<td>The divisor</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

\[ \text{Mod}(10;4) \text{ returns } 2. \]

\[ \text{Mod}(10.2;4.2) \text{ returns } 1.8. \]

---

**6.1.7.16 Power**

**Description**
Returns a number raised to a power

**Function Group**
Numeric

**Syntax**
```
num Power(number;power)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>The number to raise to a power</td>
<td>Number</td>
<td>Yes</td>
</tr>
<tr>
<td>power</td>
<td>The power</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example**

\[ \text{Power}(10;2) \text{ returns } 100. \]
**6.1.7.17 Rank**

**Description**
Ranks a measure by dimensions

**Function Group**
Numeric

**Syntax**
```
int Rank(measure;[ranking_dims];[Top|Bottom];(reset_dims))
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>The measure to be ranked</td>
<td>Measure</td>
<td>Yes</td>
</tr>
<tr>
<td>ranking_dims</td>
<td>The dimensions used to rank the measure</td>
<td>Dimension list</td>
<td>No</td>
</tr>
<tr>
<td>Top</td>
<td>Bottom</td>
<td>Sets the ranking order:</td>
<td>Keyword</td>
</tr>
<tr>
<td>reset_dims</td>
<td>The dimensions that reset the ranking</td>
<td>Dimension list</td>
<td>No</td>
</tr>
</tbody>
</table>

**Notes**
- The function uses the default calculation context to calculate the ranking if you do not specify ranking dimensions.
- You must always place dimensions in parentheses even if there is only one dimension in the list of ranking or reset dimensions.
- When you specify a set of ranking or reset dimensions you must separate them with semi-colons.
- By default the ranking is reset over a section or block break.

**Examples**
In the following table the rank is given by `Rank([Revenue];([Country]));`:

<table>
<thead>
<tr>
<th>Country</th>
<th>Revenue</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>835,420</td>
<td>2</td>
</tr>
<tr>
<td>US</td>
<td>2,451,104</td>
<td>1</td>
</tr>
</tbody>
</table>

In the following table the rank is given by `Rank([Revenue];([Country]);Bottom).` The `Bottom` argument means that the measures are ranked in descending order.
### Functions, operators and keywords

In the following table the rank is given by $\text{Rank}([\text{Revenue}];([\text{Country}];[\text{Resort}]))$:

<table>
<thead>
<tr>
<th>Country</th>
<th>Revenue</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>835,420</td>
<td>1</td>
</tr>
<tr>
<td>US</td>
<td>2,451,104</td>
<td>2</td>
</tr>
</tbody>
</table>

In the following table the rank is given by $\text{Rank}([\text{Revenue}];([\text{Country}];[\text{Year}]);([\text{Country}]))$. The rank is reset on the Country dimension.

<table>
<thead>
<tr>
<th>Country</th>
<th>Year</th>
<th>Revenue</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>FY1998</td>
<td>295,940</td>
<td>1</td>
</tr>
<tr>
<td>France</td>
<td>FY1999</td>
<td>280,310</td>
<td>2</td>
</tr>
<tr>
<td>France</td>
<td>FY2000</td>
<td>259,170</td>
<td>3</td>
</tr>
<tr>
<td>US</td>
<td>FY1998</td>
<td>767,614</td>
<td>3</td>
</tr>
<tr>
<td>US</td>
<td>FY1999</td>
<td>826,930</td>
<td>2</td>
</tr>
<tr>
<td>US</td>
<td>FY2000</td>
<td>856,560</td>
<td>1</td>
</tr>
</tbody>
</table>

**Related Topics**
- Bottom/Top operators

---

### 6.1.7.18 Round

**Description**
Rounds a number

**Function Group**
Numeric
**Syntax**

```
num Round (number;round_level)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>The number to be rounded</td>
<td>Number</td>
<td>Yes</td>
</tr>
<tr>
<td>round_level</td>
<td>The number of decimal places to which the number is rounded</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

- `Round(9.44;1)` returns 9.4.
- `Round(9.45;1)` returns 9.5.
- `Round(9.45;0)` returns 9.
- `Round(9.45;-1)` returns 10.
- `Round(4.45;-1)` returns 0.

**Related Topics**

- [Rounding and truncating numbers](#)

---

**6.1.7.19 Sign**

**Description**

Returns the sign of a number

**Function Group**

Numeric

**Syntax**

```
int Sign(number)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Any number</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

Sign returns -1 if number is negative, 0 if number is zero and 1 if number is positive.
Examples
Sign(3) returns 1.
Sign(-27.5) returns -1.

6.1.7.20 Sin

Description
Returns the sine of an angle

Function Group
Numeric

Syntax
num Sin(angle)

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>An angle in radians</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example
Sin(234542) returns -0.116992.

6.1.7.21 Sqrt

Description
Returns the square root of a number

Function Group
Numeric

Syntax
num Sqrt(number)
Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Any number</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example

Sqrt(25) returns 5.

6.1.7.22 Tan

Description

Returns the tangent of an angle

Function Group

Numeric

Syntax

num Tan(angle)

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>An angle in radians</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Examples

Tan(90) returns -2.

6.1.7.23 ToNumber

Description

Returns a string as a number

Function Group

Numeric

Syntax

num ToNumber(string)
### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A number as a string</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### Notes

If `string` is not a number, `ToNumber` returns `#ERROR`.

#### Examples

`ToNumber("45")` returns 45.

---

### 6.1.7.24 Truncate

#### Description

Truncates a number

#### Function Group

Numeric

#### Syntax

```
num Truncate(number;truncate_level)
```

#### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>The number to be rounded</td>
<td>Number</td>
<td>Yes</td>
</tr>
<tr>
<td>truncate_level</td>
<td>The number of decimal places to which the number is truncated</td>
<td>Number</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### Notes

#### Example

`Truncate(3.423;2)` returns 3.42.

#### Related Topics

- Rounding and truncating numbers

---

### 6.1.8 Set functions
6.1.8.1 Children

**Description**
Returns the child members of a member

**Function Group**
Set

**Syntax**

```
member_set member.Children
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>member</td>
<td>Any member</td>
<td>member</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
- *Children* is not used as a standalone function. It is used in the input parameter in aggregate functions that specifies the member set for aggregation.
- *member* is either specified explicitly or is the current member of a hierarchy. If you specify the member explicitly you must specify the full path in the hierarchy. If you specify a hierarchy and the hierarchy is not in the context of the formula, the formula returns an empty value.

**Examples**

```
[Geography].[US].[California].Children
```

returns [Los Angeles], [San Francisco], [San Diego].

```
[Geography].Children
```

returns [Los Angeles], [San Francisco], [San Diego] if [California] is the current member in the [Geography] hierarchy.

**Related Topics**
- Aggregate
- Average
- Count
- Max
- Min
- Sum

6.1.8.2 Depth
**Description**
Returns the depth of a member in a hierarchy

**Function Group**
Set

**Syntax**
```plaintext
int member.Depth
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>member</td>
<td>Any member</td>
<td>member</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
- The depth is the distance of the member from top level of the hierarchy.
- The top level of the hierarchy is level 0.
- `member` is either specified explicitly or is the current member of a hierarchy. If you specify the member explicitly you must specify the full path in the hierarchy. If you specify a hierarchy and the hierarchy is not in the context of the formula, the formula returns an empty value.

**Examples**

```
```

### 6.1.8.3 Descendants

**Description**
Returns descendant members of a member

**Function Group**
Set

**Syntax**
```plaintext
member_set Descendants(member[,level|distance][;desc_flag])
```
## Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>member</td>
<td>Any member</td>
<td>member</td>
<td>Yes</td>
</tr>
<tr>
<td>level</td>
<td>The level of the descendants</td>
<td>level</td>
<td>No (the level of member is the default)</td>
</tr>
<tr>
<td>distance</td>
<td>The distance of the descendant level from the current level</td>
<td>int</td>
<td>No (the level of member is the default)</td>
</tr>
<tr>
<td>desc_flag</td>
<td>Determines which descendant members are returned</td>
<td>keyword</td>
<td>No (default is Self)</td>
</tr>
</tbody>
</table>

## Notes

- **Descendants** is not used as a standalone function. It is used in the input parameter in aggregate functions that specifies the member set for aggregation.
- **member** is either specified explicitly or is the current member of a hierarchy. If you specify the member explicitly you must specify the full path in the hierarchy. If you specify a hierarchy and the hierarchy is not in the context of the formula, the formula returns an empty value.
- **Self** in desc_flag refers to the level specified by the level|distance parameter.
- **Before** in desc_flag refers to all levels above the level specified by the level|distance parameter.
- **After** in desc_flag refers to all levels below the level specified by the level|distance parameter.
- The values of desc_flag are as follows:

<table>
<thead>
<tr>
<th>desc_flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self</td>
<td>Returns the descendants at the level specified by the level</td>
</tr>
<tr>
<td>Before</td>
<td>Returns the current member and all descendants above the level specified by the level</td>
</tr>
<tr>
<td>After</td>
<td>Returns the descendants below the level specified by the level</td>
</tr>
<tr>
<td>Self_Before</td>
<td>Returns the current member and all descendants above and including the level specified by the level</td>
</tr>
<tr>
<td>Self_After</td>
<td>Returns the current member and all descendants at and below the level specified by the level</td>
</tr>
<tr>
<td>Before_After</td>
<td>Returns the current member and all descendants except those at the level specified by the level</td>
</tr>
<tr>
<td>Self_Before_After</td>
<td>Returns the current member and all descendants.</td>
</tr>
<tr>
<td>Leaves</td>
<td>Returns all members between the current member and the level specified by the level</td>
</tr>
</tbody>
</table>

- **distance** must be positive.
Examples

All examples are based on the following data:

<table>
<thead>
<tr>
<th></th>
<th>US</th>
<th>Nevada</th>
</tr>
</thead>
<tbody>
<tr>
<td>California</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Los Angeles</td>
<td>San Diego</td>
<td>San Francis-City</td>
</tr>
</tbody>
</table>

Descendants([Geography].[US].[California];[Geography].[City]) returns [San Francisco], [San Diego], [Los Angeles].

Descendants([Geography];1) returns [California], [Nevada] if the current member is [US].

Descendants([Geography].[US];2;Before) returns [US], [California], [Nevada].

Descendants([Geography].[US];[Geography].[City],Self_Before) returns [US], [California], [Nevada], [San Francisco], [San Diego], [Los Angeles], [Las Vegas], [Reno], [Carson City].

Descendants([Geography].[Geography].[State];After) returns [San Francisco], [San Diego], [Los Angeles], [Las Vegas], [Reno], [Carson City] if the current member is [US].

Descendants([Geography];1;Self_After) returns [US], [California], [Nevada], [San Francisco], [San Diego], [Los Angeles], [Las Vegas], [Reno], [Carson City] if the current member is [US].

Descendants([Geography].[US];[Geography].[State];Before_After) returns [US], [San Francisco], [San Diego], [Los Angeles], [Las Vegas], [Reno], [Carson City].

Descendants([Geography].[US];[Geography].[State];Self_Before_After) returns [US], [California], [Nevada], [San Francisco], [San Diego], [Los Angeles], [Las Vegas], [Reno], [Carson City].

Descendants([Geography].[US];[Geography].[City];Leaves) returns [San Francisco], [San Diego], [Los Angeles], [Las Vegas], [Reno], [Carson City].

Related Topics
- Aggregate
- Average
- Count
- Max
- Min
- Sum
### 6.1.8.4 IsLeaf

**Description**
Determines whether a member is a leaf member

**Function Group**
Misc

**Syntax**
```
bool member.IsLeaf
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>member</td>
<td>Any member</td>
<td>member</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
- A leaf member is a member that does not have any child members.
- `member` is either specified explicitly or is the current member of a hierarchy. If you specify the member explicitly you must specify the full path in the hierarchy. If you specify a hierarchy and the hierarchy is not in the context of the formula, the formula returns an empty value.

**Examples**
```
[Geography].[US].[California].IsLeaf returns False if [California] has the child members [Los Angeles], [San Diego], [San Francisco].
```
```
[Geography].IsLeaf returns True if the current member of the [Geography] hierarchy is [Nevada] and [Nevada] has no child members.
```

### 6.1.8.5 Key

**Description**
Returns the key of a member

**Syntax**
```
string member.Key
```
**Function Group**

Set

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>member</td>
<td>Any member</td>
<td>member</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

- The key is the internal identifier of a member.
- *member* is either specified explicitly or is the current member of a hierarchy. If you specify the member explicitly you must specify the full path in the hierarchy. If you specify a hierarchy and the hierarchy is not in the context of the formula, the formula returns an empty value.

**Example**

[Geography].[US].Key returns "XYZ" if the key of the [US] member is "XYZ".

---

**6.1.8.6 Lag**

**Description**

Returns a member at the same level as the current member and a given distance before it.

**Syntax**

`member member.Lag(distance)`

**Function Group**

Set

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>member</td>
<td>Any member</td>
<td>member</td>
<td>Yes</td>
</tr>
<tr>
<td>distance</td>
<td>The distance of the member from the current member</td>
<td>int</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

- Lag is not used as a standalone function. It is used in the input parameter in aggregate functions that specifies the member set for aggregation.
- If `distance` is positive, Lag returns the member `distance` places before `member`. If `distance` is negative, Lag returns the member `distance` places after `member`. 
• member is either specified explicitly or is the current member of a hierarchy. If you specify the member explicitly you must specify the full path in the hierarchy. If you specify a hierarchy and the hierarchy is not in the context of the formula, the formula returns an empty value.

• Lag uses the member order in the hierarchy and query to return the related member.

Examples
[Geography].[US].[California].[San Francisco].Lag(1) returns [San Diego] if [San Diego] is the previous member to [San Francisco] at the City level.

Related Topics
• Aggregate
• Average
• Count
• Max
• Min
• Sum

6.1.8.7 Parent

Description
Returns the parent member of a member

Function Group
Set

Syntax
member member.Parent

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>member</td>
<td>Any member</td>
<td>member</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Notes
• Parent is not used as a standalone function. It is used in the input parameter in aggregate functions that specifies the member set for aggregation.
• member is either specified explicitly or is the current member of a hierarchy. If you specify the member explicitly you must specify the full path in the hierarchy. If you specify a hierarchy and the hierarchy is not in the context of the formula, the formula returns an empty value.
6.1.8.8 Siblings

Description
Returns the member and the sibling members of that member.

Function Group
Set

Syntax

\[ \text{member\_set member.Siblings} \]

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>member</td>
<td>Any member</td>
<td>member</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Notes

- `Siblings` is not used as a standalone function. It is used in the input parameter in aggregate functions that specifies the member set for aggregation.
- `member` is either specified explicitly or is the current member of a hierarchy. If you specify the member explicitly you must specify the full path in the hierarchy. If you specify a hierarchy and the hierarchy is not in the context of the formula, the formula returns an empty value.
- Sibling members are members from the same level and with the same parent as `member`.

Examples

\[ \text{[Geography].[US].[California].Siblings returns [Nevada], [Arizona] if [Nevada] and [Arizona] are at the same level as [California].} \]

Related Topics

- Aggregate
• Average
• Count
• Max
• Min
• Sum

6.1.9 Misc functions

6.1.9.1 Ancestor

Description
Returns an ancestor member of a member

Function Group
Misc

Syntax
member Ancestor(member;level|distance)

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>member</td>
<td>Any member</td>
<td>member</td>
<td>Yes</td>
</tr>
<tr>
<td>level</td>
<td>The level of the ancestor</td>
<td>level</td>
<td>Either level or distance is required</td>
</tr>
<tr>
<td>distance</td>
<td>The distance of the ancestor</td>
<td>int</td>
<td>Either level or distance is required</td>
</tr>
<tr>
<td></td>
<td>level from the current level</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes
• Ancestor is not used as a standalone function. It is used in the input parameter in aggregate functions that specifies the member set for aggregation.
• member is either specified explicitly or is the current member of a hierarchy. If you specify the member explicitly you must specify the full path in the hierarchy. If you specify a hierarchy and the hierarchy is not in the context of the formula, the formula returns an empty value.
• distance must be positive.
**Examples**

All examples are based on the [Geography] hierarchy (Country > State > City). The following table shows a subset of the data in the hierarchy.

<table>
<thead>
<tr>
<th>Country</th>
<th>US</th>
<th>State</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>California</td>
<td>Nevada</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Los Angeles</td>
<td>San Diego</td>
<td>San Francisco</td>
<td>Las Vegas</td>
</tr>
</tbody>
</table>

Ancestor([Geography].[US].[California].[Los Angeles];1) returns [California].

Ancestor([Geography];[Geography].[State]) returns [California] if the current member is [San Diego].

**Related Topics**

- Aggregate
- Average
- Count
- Max
- Min
- Sum

---

### 6.1.9.2 BlockName

**Description**

Returns the block name

**Function Group**

Misc

**Syntax**

```
string BlockName()
```

**Examples**

BlockName() returns "Block1" if it is placed in a block called "Block1".
6.1.9.3 ColumnNumber

**Description**
Returns the column number

**Function Group**
Misc

**Syntax**
```java
int ColumnNumber()
```

**Examples**
`ColumnNumber()` returns 2 if the formula is placed in the second column of a table.

6.1.9.4 CurrentUser

**Description**
Returns the BI launch pad login of the current user

**Function Group**
Misc

**Syntax**
```java
string CurrentUser()
```

**Examples**
`CurrentUser()` returns "gkn" if the current user's login is "gkn".

6.1.9.5 ForceMerge

**Description**
Includes synchronized dimensions in measure calculations when the dimensions are not in the measure's calculation context
**Function Group**
Misc

**Syntax**
num ForceMerge(measure)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>Any measure</td>
<td>Measure</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Input**
The result of the calculation with the synchronized dimensions taken into account

**Notes**
- ForceMerge returns #MULTIVALE if applied to a smart measure because the grouping set necessary to calculate the smart measure does not exist.
- ForceMerge is the equivalent of the BusinessObjects/Desktop Intelligence Multicube function.

**Examples**
ForceMerge([Revenue]) returns the value of [Revenue], taking into account any synchronized dimensions that do not appear in the same block as the [Revenue] measure.

---

**6.1.9.6 GetContentLocale**

**Description**
Returns the locale of the data contained in the document (the Document Locale)

**Function Group**
Misc

**Syntax**
string GetContentLocale()  

**Notes**
The Document Locale is used to format the data in a document.

**Examples**
GetContentLocale() returns "fr_FR" if the Document Locale is "French (France)".
6.1.9.7 GetDominantPreferredViewingLocale

**Description**
Returns the dominant locale in the user's Preferred Viewing Locale group

**Function Group**
Misc

**Syntax**
```csharp
string GetDominantPreferredViewingLocale()
```

**Notes**
- Each group of related locales has a dominant locale, used as a base for all the other locales in the group. For example, US English ("en_US") is the dominant locale in the English locales group. New Zealand English ("en_NZ") is also a member of this group.
- The *Translation Manager Guide* lists all the Dominant Preferred Viewing Locales.

**Examples**
GetDominantPreferredViewingLocale returns "en_US" when the Preferred Viewing Locale is "English (New Zealand)".

**Related Topics**
- GetPreferredViewingLocale

6.1.9.8 GetLocale

**Description**
Returns the user's locale used to format the user interface (the Product Locale)

**Function Group**
Misc

**Syntax**
```csharp
string GetLocale()
```

**Notes**
The Product Locale is the locale of the user interface (for example, menu items and button text).
Examples

GetLocale() returns "en_US" if the user's Product Locale is "English (US)".

6.1.9.9 GetLocalized

Description

Returns a string localized according to the user's Preferred Viewing Locale

Syntax

string GetLocalized(string[,comment])

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>The string to be translated</td>
<td>string</td>
<td>Yes</td>
</tr>
<tr>
<td>comment</td>
<td>A comment to aid translators</td>
<td>string</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes

- The string parameter can be a string in any formula (for example, in a cell, an alerter message or a variable definition).
- When designing a report, you can use the comment parameter to provide further information to help translators translate the string. The comment appears with the string in the Translation Manager tool which translators use to translate reports.
- Each string + comment pair generates a separate string to be translated in the Translation Manager tool. As a result, GetLocalized("Product Total";"Max 20 characters") and GetLocalized("Product Total";"Use no more than 20 characters") might return different translations.

Examples

GetLocalized("Total for all products") returns the French translation of "Total for all products" if the Preferred Viewing Locale is "fr_FR".

GetLocalized("Total for all products";"Try not to use more than 20 characters") returns the German translation of "Total for all products" if the Preferred Viewing Locale is "de_DE". The function also tells the translator of the report not to use more than 20 characters if possible when translating the string.

Related Topics

- GetPreferredViewingLocale
6.1.9.10 GetPreferredViewingLocale

**Description**
Returns the user's preferred locale for viewing document data (the Preferred Viewing Locale)

**Function Group**
Misc

**Syntax**
```csharp
string GetPreferredViewingLocale()
```

**Examples**
GetPreferredViewingLocale returns "en_US" if the Preferred Viewing Locale is "English (US)".

**Related Topics**
- GetLocalized
- GetDominantPreferredViewingLocale

6.1.9.11 If...Then...Else

**Description**
Returns a value based on whether an expression is true or false

**Function Group**
Misc

**Syntax**
```csharp
If bool_value Then true_value [Else false_value]
```
**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool_value</td>
<td>A boolean value</td>
<td>Boolean</td>
<td>Yes</td>
</tr>
<tr>
<td>true_value</td>
<td>The value to return if bool_value is true</td>
<td>Any</td>
<td>Yes</td>
</tr>
<tr>
<td>false_value</td>
<td>The value to return if bool_value is false</td>
<td>Any</td>
<td>Yes if Else is included</td>
</tr>
</tbody>
</table>

**Notes**

- `true_value` and `false_value` can mix datatypes.
- You can use the boolean operators `And`, `Between`, `InList`, `Or` and `Not` with `If`.
- You can nest `If` conditions by replacing any `Else` clause with an `ElseIf` clause. This syntax describes one level of nesting:

  ```
  If bool_value Then true_value [ElseIf bool_value Then true_value Else false_value ...]
  ```

- The original syntax of the `If` function, `If(bool_value;true_value;false_value)`, is also supported.

**Examples**

If `[Sales Revenue]>1000000` Then "High Revenue" returns "High Revenue" for all rows whose revenue is larger than 1,000,000 and nothing for all other rows.

If `[Sales Revenue] >1000000` Then "High Revenue" Else `[Revenue]` returns "High Revenue" for all rows whose revenue is larger than 1,000,000 and the revenue value for all other rows.

If `[Sales Revenue]>1000000` Then "High Revenue" Else "Low Revenue" returns "High Revenue" for all rows whose revenue is larger than 1,000,000 and "Low Revenue" for all rows whose revenue is less than 1,000,000.

If `[Sales Revenue]>1000000` Then "High Revenue" ElseIf `[Sales Revenue] > 800000` Then "Medium Revenue" Else "Low Revenue" returns "High Revenue" for all rows whose revenue is larger than 1000000, "Medium Revenue" for all rows whose revenue is between 800000 and 1000000, and "Low Revenue" for all other rows.

**Related Topics**

- `If`
- `And operator`
- `Between operator`
- `Inlist operator`
- `Or operator`
- `Not operator`
6.1.9.12 If

**Description**
Returns a value based on whether an expression is true or false

**Function Group**
Misc

**Syntax**
If(bool_value;true_value;false_value)

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool_value</td>
<td>A boolean value</td>
<td>Boolean</td>
<td>Yes</td>
</tr>
<tr>
<td>true_value</td>
<td>The value to return if bool_value is true</td>
<td>Any</td>
<td>Yes</td>
</tr>
<tr>
<td>false_value</td>
<td>The value to return if bool_value is false</td>
<td>Any</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

- `true_value` and `false_value` can mix datatypes.
- You can nest If conditions by replacing `false_value` with additional If conditions. This syntax shows one level of nesting:
  
  ```
  If(bool_value;true_value;If(bool_value;true_value;false_value);false_value)
  ```

- The If...Then...Else syntax is also supported.

**Examples**

If([Sales Revenue]>1000000;"High Revenue";"Low Revenue") returns "High Revenue" for all rows whose revenue is larger than 1,000,000 and "Low Revenue" for all rows whose revenue is less than 1,000,000.

If([Sales Revenue]>1000000;"High Revenue";[Revenue]) returns "High Revenue" for all rows whose revenue is larger than 1,000,000 and the revenue value for all other rows.

**Related Topics**

- If...Then...Else
6.1.9.13 LineNumber

**Description**
Returns the line number in a table

**Function Group**
Misc

**Syntax**
```c
int LineNumber()
```

**Notes**
Numbering of the lines in a table starts with the header, which is line 1.

**Examples**
LineNumber() returns 2 when the function appears at the second line in a table.

6.1.9.14 NameOf

**Description**
Returns the name of an object

**Function Group**
Misc

**Syntax**
```c
string NameOf(obj)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>Any report object</td>
<td>Report object</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**
The NameOf function appears in column and row headers in reports.

**Examples**
NameOf([Reservation Date]) returns "Reservation Date".
### 6.1.9.15 NoFilter

**Description**
Ignores filters when calculating a value

**Function Group**
Misc

**Syntax**
```plaintext
input_type NoFilter(obj|;All|Drill)
```

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>Any report object</td>
<td>Report object</td>
<td>Yes</td>
</tr>
</tbody>
</table>
| All|Drill | • No keyword specified - ignore report and block filters  
    • All - ignore all filters  
    • Drill - ignore report and drill filters | Keyword | No       |

**Notes**
- **NoFilter(obj;Drill)** does not work in query drill mode because the drill filters are added to the query rather than applied to the report data.
- If you end drill mode with drill filters applied, the drill filters become report filters and can change the value of any objects to which **NoFilter(obj;Drill)** is applied.

**Examples**
- When placed in a block footer, **NoFilter(Sum([Sales Revenue]))** returns the total sales revenue of all possible rows in the block, even when rows are filtered out of the block.
- **NoFilter(Sum([Sales Revenue]);All)** returns the sum of the sales revenue for all countries including France, even though there is a filter that excludes France from the report.
- **NoFilter(Sum([Sales Revenue]);Drill)** returns the sum of the sales revenue for all countries, even when there is a drill filter on the [Country] dimension.

### 6.1.9.16 NumberOfPages
**Description**
Returns the number of pages in a report

**Function Group**
Misc

**Syntax**
integer NumberOfPages()

**Examples**
NumberOfDataPages() returns 2 if the report has two pages.

### 6.1.9.17 Page

**Description**
Returns the current page number in a report

**Function Group**
Misc

**Syntax**
integer Page()

**Example**
Page() returns 2 if it appears in the second page of the report.

### 6.1.9.18 Previous

**Description**
Returns a previous value of an object

**Function Group**
Misc

**Syntax**
input_type Previous(dimension|measure|Self[/{reset_dims}]][;offset][;NoNull])
**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
<td>The dimension or measure whose previous value the function</td>
<td>Dimension,</td>
<td>Yes</td>
</tr>
<tr>
<td>measure</td>
<td>returns, or the Self keyword</td>
<td>measure or</td>
<td></td>
</tr>
<tr>
<td>Self</td>
<td></td>
<td>keyword</td>
<td></td>
</tr>
<tr>
<td>reset_dims</td>
<td>The list of dimensions used to reset the calculation</td>
<td>Dimension list</td>
<td>No</td>
</tr>
<tr>
<td>offset</td>
<td>Specifies the value of dimension or measure that is</td>
<td>Integer</td>
<td>No (default is 1)</td>
</tr>
<tr>
<td></td>
<td>offset rows previous to the current row</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NoNull</td>
<td>Tells the function to return the first non-null value starting</td>
<td>Keyword</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>from the offset</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes**

- The default value of offset is 1. Previous([Revenue];1) and Previous([Revenue]) are functionally the same.
- When you include the NoNull argument, the function returns the first non-null value of the object beginning from the cell offset rows before the current row and counting backwards.
- You can use extended syntax context operators with Previous.
- The Self operator allows you to refer to the previous value of a cell when it contains content other than one report object.
- You must always place dimensions in parentheses even if there is only one dimension in the list of reset dimensions.
- When you specify a set of reset dimensions you must separate them with semi-colons.
- Previous is applied after all report, section and block filters, and all sorts, are applied.
- You cannot apply sorts or filters on formulas that use Previous.
- If Previous is applied on a measure and the measure returns an undefined value, Previous returns an undefined value even if the previous line returned a value.
- Previous ignores breaks when placed outside a break header or footer.
- Previous returns the value in the previous instance of the footer when placed in a break footer.
- Previous is reset in each report section.
- When used in a crosstab, Previous does not treat the last value in a row as the previous value of the first value of the next row.

**Examples**

Previous([Country];1) returns the following values in the following table:

<table>
<thead>
<tr>
<th>Country</th>
<th>Revenue</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>5,000,000</td>
<td></td>
</tr>
<tr>
<td>UK</td>
<td>2,000,000</td>
<td>US</td>
</tr>
<tr>
<td>France</td>
<td>2,100,000</td>
<td>UK</td>
</tr>
</tbody>
</table>
**Previous**([Revenue]) returns the following values in the following table:

<table>
<thead>
<tr>
<th>Country</th>
<th>Revenue</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>5,000,000</td>
<td></td>
</tr>
<tr>
<td>UK</td>
<td>2,000,000</td>
<td>5,000,000</td>
</tr>
<tr>
<td>France</td>
<td>2,100,000</td>
<td>2,000,000</td>
</tr>
</tbody>
</table>

**Previous**([Revenue];([Country])) returns the following values in the following table:

<table>
<thead>
<tr>
<th>Country</th>
<th>Region</th>
<th>Revenue</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>North</td>
<td>5,000,000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>South</td>
<td>7,000,000</td>
<td>5,000,000</td>
</tr>
<tr>
<td>UK</td>
<td>North</td>
<td>3,000,000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>South</td>
<td>4,000,000</td>
<td>3,000,000</td>
</tr>
</tbody>
</table>

**Previous**([Revenue]) returns the following values in the following crosstab:

<table>
<thead>
<tr>
<th></th>
<th>2004</th>
<th>Previous</th>
<th>2005</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>5,000,000</td>
<td></td>
<td>6,000,000</td>
<td>5,000,000</td>
</tr>
<tr>
<td>UK</td>
<td>2,000,000</td>
<td></td>
<td>2,500,000</td>
<td>2,000,000</td>
</tr>
<tr>
<td>France</td>
<td>3,000,000</td>
<td></td>
<td>2,000,000</td>
<td>3,000,000</td>
</tr>
</tbody>
</table>

**Previous**([Revenue]) returns the following values in the following table with a break on [Country]:

<table>
<thead>
<tr>
<th>Country</th>
<th>Region</th>
<th>Revenue</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>North</td>
<td>5,000,000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>South</td>
<td>7,000,000</td>
<td>5,000,000</td>
</tr>
<tr>
<td>US</td>
<td></td>
<td>12,000,000</td>
<td></td>
</tr>
<tr>
<td>UK</td>
<td>North</td>
<td>3,000,000</td>
<td>7,000,000</td>
</tr>
<tr>
<td></td>
<td>South</td>
<td>4,000,000</td>
<td>3,000,000</td>
</tr>
</tbody>
</table>
Previous([Revenue];2;NoNull) returns the following values in the following table:

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Revenue</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>Q1</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Q2</td>
<td>400</td>
<td>500</td>
</tr>
<tr>
<td>2008</td>
<td>Q3</td>
<td>400</td>
<td>500</td>
</tr>
<tr>
<td>2008</td>
<td>Q4</td>
<td>700</td>
<td>500</td>
</tr>
<tr>
<td>2008</td>
<td>Q1</td>
<td>300</td>
<td>400</td>
</tr>
<tr>
<td>2008</td>
<td>Q2</td>
<td>300</td>
<td>700</td>
</tr>
<tr>
<td>2008</td>
<td>Q3</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>2008</td>
<td>Q4</td>
<td>200</td>
<td>300</td>
</tr>
</tbody>
</table>

2*Previous(Self) returns the sequence 2, 4, 6, 8, 10...

**Related Topics**
- Comparing values using the Previous function
- Self operator

### 6.1.9.19 RefValue

**Description**
Returns the reference value of a report object when data tracking is activated

**Function Group**
Misc

**Syntax**
```
input_type RefValue(obj)
```

**Examples**
RefValue([Top Performing Region]) returns "South West" if the value of the [Top Performing Region] variable is "South West" in the reference data.
RefValue([Revenue]) returns 1000 if the value of the [Revenue] measure is 1000 in the reference data.

### 6.1.9.20 RelativeValue

**Description**

Returns previous or subsequent values of an object

**Function Group**

Misc

**Syntax**

\[ \text{input_type RelativeValue(measure|detail;slicing_dims;offset)} \]

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>detail</td>
<td>Any measure or a detail of a dimension in the block</td>
<td>Measure or detail</td>
</tr>
<tr>
<td>slicing_dims</td>
<td>The dimensions that provide the calculation context</td>
<td>Dimension list</td>
<td>Yes</td>
</tr>
<tr>
<td>offset</td>
<td>Specifies the value of measure or detail that is offset rows removed from the current row</td>
<td>Integer</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes**

- The object must be a measure or a detail of a dimension available in the block.
- The sort order of the list of values of the slicing dimensions is used to determine the output of the function.
  
The sort order is determined by two factors: sorts applied to the slicing dimensions, and the order in which the slicing dimensions are listed in the function.

- A dimension used as a section master can be specified as a slicing dimension.
- All the slicing dimensions must be present in the block or section header of the block in which the function is placed. If a slicing dimension is later removed from the block, the function returns the #COMPUTATION error.
- If the offset exceeds the number of rows in the list of values of the slicing dimension, the function returns null.
- RelativeValue cannot be used recursively.
- You must always place dimensions in parentheses even if there is only one dimension in the list of slicing dimensions.
Examples

The RelativeValue column in the table below contains the following formula:

\[
\text{RelativeValue}([\text{Revenue}] ; ([\text{Year}] ) ; -1)
\]

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales Person</th>
<th>Revenue</th>
<th>RelativeValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>Q1</td>
<td>Smith</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q2</td>
<td>Jones</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q3</td>
<td>Wilson</td>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q4</td>
<td>Harris</td>
<td>3000</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Q1</td>
<td>Smith</td>
<td>4000</td>
<td>1000</td>
</tr>
<tr>
<td>2008</td>
<td>Q2</td>
<td>Jones</td>
<td>3400</td>
<td>2000</td>
</tr>
<tr>
<td>2008</td>
<td>Q3</td>
<td>Wilson</td>
<td>2000</td>
<td>1500</td>
</tr>
<tr>
<td>2008</td>
<td>Q4</td>
<td>Harris</td>
<td>1700</td>
<td>3000</td>
</tr>
</tbody>
</table>

Related Topics

- #COMPUTATION
- Comparing values using the RelativeValue function

6.1.9.21 ReportName

Description

Returns the name of a report

Function Group

Misc

Syntax

\[
\text{string ReportName()}
\]

Examples

ReportName() returns "Sales Report" if it is placed in a report called "Sales Report".

6.1.9.22 RowIndex
**Description**

Returns the number of a row

**Function Group**

Misc

**Syntax**

integer RowIndex()

**Notes**

- Row numbering starts at 0.
- RowIndex returns #MULTIVALUE when placed in a table header or footer.

**Examples**

RowIndex returns 0 when it appears on the first row of a table.

---

### 6.1.9.23 UniqueNameOf

**Description**

Returns the unique name of an object

**Function Group**

Misc

**Syntax**

string UniqueNameOf(obj)

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>Any report object</td>
<td>Report object</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**

UniqueNameOf([Reservation Date]) returns "Reservation Date".

---

### 6.2 Function and formula operators

Operators link the various components in a formula. Formulas can contain mathematical, conditional, logical, function-specific or extended syntax operators.
6.2.1 Mathematical operators

Mathematical operators are familiar from everyday arithmetic. There are addition (+), subtraction (-), multiplication (*), division (/) operators that allow you to perform mathematical operations in a formula. The formula \([\text{Sales Revenue}] - [\text{Cost of Sales}]\) contains a mathematical operator, in this case subtraction.

**Note:**
When used with character strings, the ‘+’ operator becomes a string concatenation operator. That is, it joins character strings. For example, the formula "John" + " Smith" returns "John Smith".

6.2.2 Conditional operators

Conditional operators determine the type of comparison to be made between values.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>

You use conditional operators with the If function, as in:

\[\text{If } [\text{Revenue}] > 10000 \text{ Then } \text{"High" Else } \text{"Low"}\]

which returns "High" for all rows where the revenue is greater than or equal to 10000 and "Low" for all other rows.

6.2.3 Logical operators
The logical operators are *And*, *Or*, *Not*, *Between* and *Inlist*. Logical operators are used in boolean expressions, which return *True* or *False*.

### 6.2.3.1 And operator

**Description**
The *And* operator links boolean values. If all the boolean values linked by *And* return true, the combination of all the values also returns true.

**Syntax**
```
bool_value And bool_value [And bool_value...]
```

**Examples**
If `[Resort]` = "Bahamas Beach" And `[Revenue]`>100000 Then "High Bahamas Revenue" returns "High Bahamas Revenue" if `[Resort]` = "Bahamas Beach" And `[Revenue]`>100000.

### 6.2.3.2 Or operator

**Description**
The *Or* operator links boolean values. If any one boolean value linked by *Or* returns true, the combination of all the values also returns true.

**Syntax**
```
bool_value Or bool_value [Or bool_value...]
```

**Examples**
If `[Resort]` = "Bahamas Beach" Or `[Resort]"Hawaiian Club" Then "US" Else "France" returns "US" if `[Resort]="Bahamas Beach" or "Hawaiian Club", or "France" otherwise.

### 6.2.3.3 Not operator

**Description**
The *Not* operator returns the opposite of a boolean value.

**Syntax**
```
bool Not(bool_value)
```
Examples
If Not([Country] = "US") Then "Not US" returns "Not US" if [Country] has any value other than "US".

6.2.3.4 Between operator

Description
The Between operator determines if a variable is between two values.

Syntax
bool Between(first_value;second_value)

Notes
• You use Between with the If function and the Where operator.
• Changing the document locale can impact the result returned by the Between operator.

Examples
If [Revenue] Between(800000;900000) Then "Medium Revenue" returns "Medium Revenue" if [Revenue] is between 800000 and 900000.
[Sales Revenue] Between (10000;20000) returns true if the sales revenue is between 10000 and 20000.
If ([Sales Revenue] Between (200000;500000);"Medium Revenue";"Low/High Revenue") returns "Medium Revenue" if [Sales Revenue] is 300000.

Related Topics
• If...Then...Else
• Where operator

6.2.3.5 Inlist operator

Description
The Inlist operator determines if a value is in a list of values.

Syntax
bool test_value Inlist(value_list)
Notes
It is the combination of test_value + InList that returns a boolean value, not InList alone.

Examples
If Not ([Country] InList("England";"Scotland";"Wales")) Then "Not Britain"
Else "Britain" returns "Not Britain" if [Country] is not equal to "England", "Scotland" or "Wales", or "Britain" otherwise.

If [Resort] InList("Bahamas Beach";"Hawaiian Club") Then "US Resort" returns "US Resort" if [Resort] is equal to "Bahamas Beach" or "Hawaiian Club".

Related Topics
• If...Then...Else
• Where operator

6.2.4 Function-specific operators

Some functions can take specific operators as arguments. For example, the Previous function can take the Self operator.

All functions use ) and ( to enclose function arguments. Functions that accept multiple parameters use ; to separate the parameters.

6.2.4.1 All operator

The All operator tells the NoFilter function to ignore all filters, or tells the Count function to count all values, including duplicates.

Related Topics
• Count
• Distinct/All operators
• NoFilter
• All/Drill operators
6.2.4.2 All/Drill operators

**Description**
The All/Drill operators determine which filters the NoFilter function ignores.

- Not specified - NoFilter ignores report and block filters
- All - NoFilter ignores all filters
- Drill - NoFilter ignores report filters and drill filters

6.2.4.3 Bottom/Top operators

**Description**
The Bottom/Top operators tell the Rank function to rank in descending or ascending order.

- Top - ranks in descending order
- Bottom - ranks in ascending order

**Examples**
Rank([Revenue];([Country]);Top ranks countries by revenue from highest to lowest.

**Related Topics**
- Rank

6.2.4.4 Break operator

**Description**
The Break operator tells Percentage function to account for table breaks.

**Examples**
The formula Percentage([Revenue]) gives the following result in the following table (percentages are calculated on the total revenue in the block):

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Revenue</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>Q1</td>
<td>10000</td>
<td>10%</td>
</tr>
</tbody>
</table>
The formula \( \text{Percentage}([\text{Revenue}]; \text{Break}) \) gives the following result in the following table (percentages are calculated on the total revenue in each part of the block):

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Revenue</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>Q1</td>
<td>10000</td>
<td>33.3%</td>
</tr>
<tr>
<td>2005</td>
<td>Q2</td>
<td>20000</td>
<td>66.6%</td>
</tr>
<tr>
<td>2006</td>
<td>Q1</td>
<td>30000</td>
<td>42.9%</td>
</tr>
<tr>
<td>2006</td>
<td>Q2</td>
<td>40000</td>
<td>57.1%</td>
</tr>
</tbody>
</table>

**6.2.4.5 Distinct/All operators**

The \text{Distinct/All} operators tell the \text{Count} function to count distinct values only, or all values.

**Examples**

- \text{Count}([\text{Revenue}]; \text{Distinct}) returns 3 if \([\text{Revenue}]\) has the values (5;5;6;4).
- \text{Count}([\text{Revenue}]; \text{All}) returns 4 if \([\text{Revenue}]\) has the values (5;5;6;4).

**Related Topics**
- \text{Percentage}
- \text{Count}
6.2.4.6 IncludeEmpty operator

**Description**
The IncludeEmpty operator tells some aggregate functions to include empty values in calculations.

**Examples**
Average([Revenue];IncludeEmpty) returns 3 if [Revenue] has the values (5;3;<empty>;4).

**Related Topics**
• Average
• Count
• RunningAverage
• RunningCount

6.2.4.7 Index operator

**Description**
The Index operator tells the UserResponse and RefValueUserResponse functions to return the database primary key of the prompt response.

**Related Topics**
• UserResponse
• RefValueUserReponse

6.2.4.8 Linear operator

**Description**
The Linear operator tells the Interpolation function to use linear regression with least squares interpolation to supply missing measure values.

Linear regression with least squares interpolation calculates missing values by calculating a line equation in the form f(x) = ax + b that passes as closely as possible through all the available values of the measure.
6.2.4.9 NoNull operator

Description
The NoNull operator tells the Previous function to ignore null values.

When used with NoNull, Previous returns the first non-null value of the object, beginning from the cell offset rows before the current row and counting backwards.

Related Topics
• Previous

6.2.4.10 NotOnBreak operator

Description
The NotOnBreak operator tells the Interpolation function to ignore section and block breaks.

Related Topics
• Interpolation

6.2.4.11 PointToPoint operator

Description
The PointToPoint operator tells the Interpolation function to use point-to-point interpolation to supply missing measure values.

Point-to-point interpolation calculates missing values by calculating a line equation in the form \( f(x) = ax + b \) that passes through the two adjacent values of the missing value.

Related Topics
• Interpolation
6.2.4.12 Row/Col operators

Description
The Row/Col operators set the calculation direction of the following functions: Percentage, RunningAverage, RunningCount, RunningMax, RunningMin, RunningProduct, RunningSum.

Notes
The Row operator calculates each value in the row as a percentage of the total value of all the rows in the embedding context. The Col operator calculates each value in the column as a percentage of the total value of all the columns in the embedding context.

In a crosstab, the value in each cell is calculated by default as a percentage of the total value in the crosstab. The Row operator calculates the values in the rows as percentages of the total value for the row. The Col operator calculates the values in the columns as percentages of the total value in the column.

Examples
In a crosstab, Percentage([Measure]) gives the following result:

<table>
<thead>
<tr>
<th>Measure</th>
<th>Percentage</th>
<th>Measure</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>10%</td>
<td>500</td>
<td>50%</td>
</tr>
<tr>
<td>200</td>
<td>20%</td>
<td>200</td>
<td>20%</td>
</tr>
</tbody>
</table>

Percentage([Measure];Row) gives the following result:

<table>
<thead>
<tr>
<th>Measure</th>
<th>Percentage</th>
<th>Measure</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>16.7%</td>
<td>500</td>
<td>83.3%</td>
</tr>
<tr>
<td>200</td>
<td>50%</td>
<td>200</td>
<td>50%</td>
</tr>
</tbody>
</table>

Percentage([Measure];Col) gives the following result:

<table>
<thead>
<tr>
<th>Measure</th>
<th>Percentage</th>
<th>Measure</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>33.3%</td>
<td>500</td>
<td>83.3%</td>
</tr>
<tr>
<td>200</td>
<td>66.6%</td>
<td>200</td>
<td>16.7%</td>
</tr>
</tbody>
</table>
The Row operator calculates the running aggregate by row. The Col operator calculates the running aggregate by column.

In a crosstab, \( \text{RunningSum([Measure])} \) or \( \text{RunningSum([Measure];Row)} \) gives the following result:

<table>
<thead>
<tr>
<th>Measure</th>
<th>RunningSum</th>
<th>Measure</th>
<th>RunningSum</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>400</td>
<td>700</td>
<td>250</td>
<td>950</td>
</tr>
</tbody>
</table>

In a crosstab, \( \text{RunningSum([Measure];Col)} \) gives the following result:

<table>
<thead>
<tr>
<th>Measure</th>
<th>RunningSum</th>
<th>Measure</th>
<th>RunningSum</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
<td>200</td>
<td>700</td>
</tr>
<tr>
<td>400</td>
<td>500</td>
<td>250</td>
<td>950</td>
</tr>
</tbody>
</table>

**Related Topics**
- Percentage
- RunningAverage
- RunningCount
- RunningMax
- RunningMin
- RunningProduct
- RunningSum

### 6.2.4.13 Self operator

**Description**
Refers the Previous function to the previous cell when it does not contain a report object.

**Examples**
\[ 5 + \text{Previous(Self)} \] returns the sequence 5, 10, 15, 20, 25, 30...
\[ 1 + 0.5 \times \text{Previous(Self)} \] returns the sequence 1, 1.5, 1.75, 1.88...

**Related Topics**
- Previous
6.2.4.14 Where operator

Description
The Where operator restricts the data used to calculate a measure.

Examples
The formula `Average ([Sales Revenue]) Where ([Country] = "US")` calculates the average sales where the country is "US".

The formula `Average ([Sales Revenue]) Where ([Country] = "US" Or [Country] = "France")` calculates the average sales where the country is "US" or "France".

The formula `[Revenue] Where (Not ([Country] Inlist ("US"; "France")))` calculates the revenue for the countries other than US and France.

The variable [High Revenue] has the formula `[Revenue] Where [Revenue > 500000]`. When placed in a block, [High Revenue] displays either the revenue when its value is greater than 500000, or nothing. When placed in a footer at the bottom of the [High Revenue] column, the formula `Average ([High Revenue])` returns the average of all the revenues greater than 500000.

Related Topics
- And operator
- Between operator
- Inlist operator
- Or operator
- Not operator

6.2.5 Extended syntax operators

You specify input and output contexts explicitly with context operators. The following table lists the context operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Specifies an explicit list of dimensions to use in the context.</td>
</tr>
<tr>
<td>ForEach</td>
<td>Adds dimensions to the default context</td>
</tr>
</tbody>
</table>
The ForAll and ForEach operators are useful when you have a default context with many dimensions. It is often easier to add or subtract from the context using ForAll and ForEach than it is to specify the list explicitly using In.

**6.2.5.1 In context operator**

The In context operator specifies dimensions explicitly in a context.

**Example: Using In to specify the dimensions in a context**

In this example you have a report showing Year and Sales Revenue. Your data provider also contains the Quarter object but you do not include this dimension in the block. Instead, you want to include an additional column to show the maximum revenue by quarter in each year. Your report looks like this:

<table>
<thead>
<tr>
<th>Year</th>
<th>Sales revenue</th>
<th>Max Quarterly Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>$8,096,123.60</td>
<td>$2,660,699.50</td>
</tr>
<tr>
<td>2002</td>
<td>$13,232,246.00</td>
<td>$4,186,120.00</td>
</tr>
<tr>
<td>2003</td>
<td>$15,059,142.80</td>
<td>$4,006,717.50</td>
</tr>
</tbody>
</table>

You can see where the values in the Max Quarterly Revenue column come from by examining this block in conjunction with a block that includes the Quarter dimension:
<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>$2,660,699.50</td>
</tr>
<tr>
<td>2001</td>
<td>Q2</td>
<td>$2,279,003.00</td>
</tr>
<tr>
<td>2001</td>
<td>Q3</td>
<td>$1,367,841.00</td>
</tr>
<tr>
<td>2001</td>
<td>Q4</td>
<td>$1,788,580.00</td>
</tr>
<tr>
<td>Max:</td>
<td></td>
<td>$2,660,699.50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td></td>
<td>$3,326,172.00</td>
</tr>
<tr>
<td>Q2</td>
<td></td>
<td>$2,840,651.00</td>
</tr>
<tr>
<td>Q3</td>
<td></td>
<td>$2,879,303.00</td>
</tr>
<tr>
<td>Q4</td>
<td></td>
<td>$4,186,120.00</td>
</tr>
<tr>
<td>Max:</td>
<td></td>
<td>$4,186,120.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td></td>
<td>$3,742,989.00</td>
</tr>
<tr>
<td>Q2</td>
<td></td>
<td>$4,006,717.50</td>
</tr>
<tr>
<td>Q3</td>
<td></td>
<td>$3,953,395.00</td>
</tr>
<tr>
<td>Q4</td>
<td></td>
<td>$3,356,041.00</td>
</tr>
<tr>
<td>Max:</td>
<td></td>
<td>$4,006,717.50</td>
</tr>
</tbody>
</table>

The Max Quarterly Revenue column shows the highest quarterly revenue in each year. For example, Q4 has the highest revenue in 2002, so the Max Quarterly Revenue shows Q4 revenue on the row showing 2002.

Using the In operator, the formula for Max Quarterly Revenue is

\[
\text{Max} \left( \{ \text{Sales Revenue} \} \mid \{ \text{Year} \}, \{ \text{Quarter} \} \right) \mid \{ \text{Year} \}
\]

This formula calculates the maximum sales revenue for each (Year,Quarter) combination, then outputs this figure by year.

**Note:**
Because the default output context of the block is Year, you do not need to specify the output context explicitly in this formula.
6.2.5.2 ForEach context operator

The ForEach operator adds dimensions to a context.

Example: Using ForEach to add dimensions to a context

The following table shows the maximum revenue for each Quarter in a report which contains the Quarter dimension but does not include it in the block:

<table>
<thead>
<tr>
<th>Year</th>
<th>Sales revenue</th>
<th>Max Quarterly Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>8096123.60</td>
<td>2660699.50</td>
</tr>
<tr>
<td>2002</td>
<td>13232246.00</td>
<td>4186120.00</td>
</tr>
<tr>
<td>2003</td>
<td>15059142.80</td>
<td>4006717.50</td>
</tr>
</tbody>
</table>

It is possible to create a formula for the Max Quarterly Revenue column that does not include the ForEach operator:

Max ([Sales Revenue] In ([Year];[Quarter])) In ([Year])

Using the ForEach context operator, you can achieve the same result with the following formula:

Max ([Sales Revenue] ForEach ([Quarter])) In ([Year])

Why? Because the Year dimension is the default input context in the block. By using the ForEach operator, you add the Quarter dimension to the context, giving an input context of ([Year];[Quarter]).

6.2.5.3 ForAll context operator

The ForAll context operator removes dimensions from a context.

Example: Using ForAll to remove dimensions from a context

You have a report showing Year, Quarter and Sales Revenue and you want to add a column that shows the total revenue in each year, as shown in the following block:
To total revenues by year the input context needs to be (Year); by default it is (Year; Quarter). Therefore, you can remove Quarter from the input context by specifying ForAll ([Quarter]) in the formula, which looks like this:

```
Sum([Sales Revenue] ForAll ([Quarter]))
```

Note that you can use the In operator to achieve the same thing; in this case the formula is:

```
Sum([Sales Revenue] In ([Year]))
```

This version of the formula explicitly specifies Year as the context, rather than removing Quarter to leave Year.

### 6.2.6 Set operators

Set operators work on members in hierarchical data.

#### 6.2.6.1 Range operator

**Description**
The range operator (:) returns a set of members between and including two members at the same level.

**Syntax**
```
first_member:last_member
```
Examples

[Geography]\&[US]\&[California]\&[Los Angeles] returns [Los Angeles], [San Diego], [San Francisco] if the members at the level are in the order...[Los Angeles], [San Diego], San Francisco... 

\(\text{Sum}([\text{Revenue}]\&[\text{Geography}]\&[\text{US}]\&[\text{California}]\&[\text{Los Angeles}]\&[\text{Geography}]\&[\text{US}]\&[\text{California}]\&[\text{San Francisco}])\) returns the total revenue for Los Angeles, San Diego and San Francisco.

6.3 Extended syntax keywords

Extended syntax keywords are a form of shorthand that allows you to refer to dimensions in extended syntax without specifying those dimensions explicitly. This helps future-proof reports; if formulas do not contain hard-coded references to dimensions, they will continue to work even if dimensions are added to or removed from a report.

There are five extended syntax keywords: Report, Section, Break, Block and Body.

6.3.1 The Block keyword

The following table describes the dimensions referenced by the Block keyword depending on where it is placed in a report: the Block keyword often encompasses the same data as the Section keyword. The difference is that Block accounts for filters on a block whereas Section ignores them.

<table>
<thead>
<tr>
<th>When placed in...</th>
<th>References this data...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A block</td>
<td>Data in the whole block, ignoring breaks, respecting filters</td>
</tr>
<tr>
<td>A block break (header or footer)</td>
<td>Data in the whole block, ignoring breaks, respecting filters</td>
</tr>
<tr>
<td>A section (header, footer, or outside a block)</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Outside any blocks or sections</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Example: The Block keyword

You have a report showing Year, Quarter and Sales revenue. The report has a section based on Year. The block is filtered to exclude the third and fourth quarters.
The Yearly Average column has the formula

\[ \text{Average}([\text{Sales revenue}] \text{ In Section}) \]

and the First Half Average column has the formula

\[ \text{Average} ([\text{Sales revenue}] \text{ In Block}) \]

You can see how the Block keyword takes account of the filter on the block.

### 6.3.2 The Body keyword

The following table describes the dimensions referenced by the Body keyword depending on where it is placed in a report:

<table>
<thead>
<tr>
<th>When placed in...</th>
<th>References this data...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A block</td>
<td>Data in the block</td>
</tr>
<tr>
<td>A block break (header or footer)</td>
<td>Data in the block</td>
</tr>
<tr>
<td>A section (header, footer, or outside a block)</td>
<td>Data in the section</td>
</tr>
<tr>
<td>Outside any blocks or sections</td>
<td>Data in the report</td>
</tr>
</tbody>
</table>
Example: **The Body keyword**

You have a report showing Year, Quarter and Sales revenue, with a break on Year. The report has a section based on Year and a break on Quarter.

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
<th>Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>2,660,700</td>
<td>2,660,699.5</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>2,279,003</td>
<td>2,279,003</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>1,367,841</td>
<td>1,367,840.7</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>1,788,580</td>
<td>1,788,580.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8,096,123.6</td>
<td></td>
</tr>
</tbody>
</table>

The Body column has the formula

\[ \text{Sum ([Sales Revenue]) In Body} \]

The totals in the Body column are the same as those in the Sales revenue column because the Body keyword refers to the data in the block. If you were to remove the Month object, the figures in the Block column would change to correspond with the changed figures in the Sales revenue column. If you were to place the formula in the report footer it would return the total revenue for the body.

---

**6.3.3 The Break keyword**

The following table describes the dimensions referenced by the Break keyword depending on where it is placed in a report:

<table>
<thead>
<tr>
<th>When placed in...</th>
<th>References this data...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A block</td>
<td>Data in the part of a block delimited by a break</td>
</tr>
<tr>
<td>A block break (header or footer)</td>
<td>Data in the part of a block delimited by a break</td>
</tr>
<tr>
<td>A section (header, footer, or outside a block)</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Outside any blocks or sections</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Example: **The Break keyword**

You have a report showing Year, Quarter and Sales revenue:
The report has break on Year. The Break Total column has the formula:

```
Sum ([Sales Revenue]) In Break
```

Without the Break keyword this column would duplicate the figures in the Sales revenue column, because it would use the default output context ([Year];[Quarter]).

### 6.3.4 The Report keyword

The following table describes the data referenced by the Report keyword depending on where it is placed in a report:

<table>
<thead>
<tr>
<th>When placed in...</th>
<th>References this data...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A block</td>
<td>All data in the report</td>
</tr>
<tr>
<td>A block break (header or footer)</td>
<td>All data in the report</td>
</tr>
<tr>
<td>A section (header, footer, or outside a block)</td>
<td>All data in the report</td>
</tr>
<tr>
<td>Outside any blocks or sections</td>
<td>All data in the report</td>
</tr>
</tbody>
</table>

Example: **The Report keyword**

You have a report showing Year, Quarter and Sales revenue. The report has a column, Report Total, that shows the total of all revenue in the report.
The formula for the Report Total column is `Sum([Sales revenue]) In Report`. Without the Report keyword, this column would duplicate the figures in the Sales Revenue column because it would use the default output context `([Year];[Quarter])`.

### 6.3.5 The Section keyword

The following table describes the data referenced by the Section keyword depending on where it is placed in a report.

<table>
<thead>
<tr>
<th>When placed in...</th>
<th>References this data...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A block</td>
<td>All data in the section</td>
</tr>
<tr>
<td>A block break (header or footer)</td>
<td>All data in the section</td>
</tr>
<tr>
<td>A section (header, footer, or outside a block)</td>
<td>All data in the section</td>
</tr>
<tr>
<td>Outside any blocks or sections</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**Example: The Section keyword**

You have a report showing Year, Quarter, and Sales revenue.

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
<th>Section Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>$2,860,700</td>
<td>$3,367,512.4</td>
</tr>
<tr>
<td>2001</td>
<td>Q2</td>
<td>$2,279,003</td>
<td>$3,367,512.4</td>
</tr>
<tr>
<td>2001</td>
<td>Q3</td>
<td>$1,357,841</td>
<td>$3,367,512.4</td>
</tr>
<tr>
<td>2001</td>
<td>Q4</td>
<td>$1,766,680</td>
<td>$3,367,512.4</td>
</tr>
<tr>
<td>2002</td>
<td>Q1</td>
<td>$3,325,172</td>
<td>$3,367,512.4</td>
</tr>
<tr>
<td>2002</td>
<td>Q2</td>
<td>$2,840,651</td>
<td>$3,367,512.4</td>
</tr>
<tr>
<td>2002</td>
<td>Q3</td>
<td>$2,673,303</td>
<td>$3,367,512.4</td>
</tr>
<tr>
<td>2002</td>
<td>Q4</td>
<td>$4,186,120</td>
<td>$3,367,512.4</td>
</tr>
<tr>
<td>2003</td>
<td>Q1</td>
<td>$3,742,989</td>
<td>$3,367,512.4</td>
</tr>
<tr>
<td>2003</td>
<td>Q2</td>
<td>$4,006,716</td>
<td>$3,367,512.4</td>
</tr>
<tr>
<td>2003</td>
<td>Q3</td>
<td>$3,553,395</td>
<td>$3,367,512.4</td>
</tr>
<tr>
<td>2003</td>
<td>Q4</td>
<td>$3,366,041</td>
<td>$3,367,512.4</td>
</tr>
</tbody>
</table>
The report has a section based on Year. The Section Total column has the formula:

**`Sum ([[Sales Revenue]] In Section)`**

The figure in the Section Total column is the total revenue for 2001, because the section break occurs on the Year object. Without the Section keyword this column would duplicate the figures in the Sales revenue column, because it would use the default output context ([Year];[Quarter]).

### 6.4 Rounding and truncating numbers

Several functions contain a parameter that determines to what level the function rounds or truncates the value it returns. This parameter accepts an integer that is either greater than 0, 0, or less than 0. The following table explains how numbers are rounded and truncated in these cases:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| > 0       | The function rounds/truncates to `<parameter>` decimal places.  
            | Examples:  
            | `Round(3.13;1)` returns 3.1  
            | `Round(3.157;2)` returns 3.16 |
| 0         | The function rounds/truncates to the nearest integer.  
            | Examples:  
            | `Truncate(3.7;0)` returns 3  
            | `Truncate(4.164;0)` returns 4 |
| < 0       | The function rounds/truncates to the nearest 10 (parameter = -1), 100 (parameter = -2), 1000 (parameter = -3) and so on.  
            | Examples:  
            | `Round(123.76;-1)` returns 120  
            | `Round(459.9;-2)` returns 500  
            | `Truncate(1600;-3)` returns 1000 |

**Note:**  
Numbers are represented internally as doubles and are accurate up to sixteen digits.
6.5 Referring to members and member sets in hierarchies

You refer to members and member sets in functions using the syntax [hierarchy] & path.function. The path and function parts are optional. In path, you refer to each member in square brackets, with members separated by full stops. The names of members and levels are case-sensitive.

**Note:**
You use member sets to override the default calculation context for a hierarchy. In functions that accept member sets, you enclose the member set in {}.

You refer to ranges of members using a colon (:) between the start and end member, and with the full path specified for each member. A range includes all members at the same level as the specified members.

An example of range syntax is: [Sales Hierarchy] & [Customer_Type].[ENTERPRISE];[Large].[Nancy Davolio]: [Sales Hierarchy] & [Customer_Type].[ENTERPRISE];[Large].[Andrew Smith].

**Example:** Referring to members and member sets

You have the following hierarchy:

<table>
<thead>
<tr>
<th>Sales Hierarchy</th>
<th>Order Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer_Type</td>
<td>277,290,434</td>
</tr>
<tr>
<td>ENTERPRISE</td>
<td>180,063,361</td>
</tr>
<tr>
<td>Large</td>
<td>113,905,997</td>
</tr>
<tr>
<td>Nancy Davolio</td>
<td>44,855,689</td>
</tr>
<tr>
<td>Janet Leverling</td>
<td>44,050,308</td>
</tr>
<tr>
<td>Andrew Smith</td>
<td>30,000,000</td>
</tr>
<tr>
<td>GLOBAL</td>
<td>91,157,363</td>
</tr>
</tbody>
</table>
• \([\text{Sales Hierarchy}]\&[\text{Customer_Type}].[\text{ENTERPRISE}].[\text{Large}].\text{Children}\) refers to the [Nancy Davolio], [Janet Leverling] and [Andrew Smith] members.

• \(\text{Sum}([\text{Order Amount}]\;\{[\text{Sales Hierarchy}]\&[\text{Customer_Type}].[\text{ENTERPRISE}].[\text{Large}].\text{children}\})\) returns 113,905,997 (the sum of the measure for the three child members).

• \([\text{Sales Hierarchy}]\&[\text{Customer_Type}].[\text{ENTERPRISE}].[\text{Large}].[\text{Janet Leverling}]\) refers to the [Janet Leverling] member.

• \(\text{Sum}([\text{Order Amount}]\;\{[\text{Sales Hierarchy}]\&[\text{Customer_Type}].[\text{ENTERPRISE}].[\text{Large}].[\text{Janet Leverling}]\};\{[\text{Sales Hierarchy}]\&[\text{Customer_Type}].[\text{ENTERPRISE}].[\text{Large}].[\text{Nancy Davolio}]\})\) returns 88,905,997 (the sum of the measure for the two members).

• \([\text{Sales Hierarchy}]\&[\text{Customer_Type}].[\text{ENTERPRISE}].[\text{Large}].[\text{Nancy Davolio}]:[\text{Sales Hierarchy}]\&[\text{Customer_Type}].[\text{ENTERPRISE}].[\text{Large}].[\text{Andrew Smith}]\) refers to the [Nancy Davolio], [Janet Leverling] and [Andrew Smith] members.

• \(\text{Sum}([\text{Order Amount}]\;\{[\text{Sales Hierarchy}]\&[\text{Customer_Type}].[\text{ENTERPRISE}].[\text{Large}].[\text{Nancy Davolio}]\};\{[\text{Sales Hierarchy}]\&[\text{Customer_Type}].[\text{ENTERPRISE}].[\text{Large}].[\text{Andrew Smith}]\})\) returns 113,905,997 (the sum of the measure for the three members in the range).

• \([\text{Sales Hierarchy}].\text{children}\) refers to all members in the [Sales Hierarchy] hierarchy.

• \(\text{Sum}([\text{Order Amount}]\;\{[\text{Sales Hierarchy}]\&[\text{Customer_Type}].\text{children}\})\) returns 277,290.434.
**Troubleshooting formulas**

7.1 **Formula error and information messages**

In some cases a formula cannot return a value and returns an error or information message beginning with "#". The message appears in the cell in which the formula is placed.

You can format report data that returns error messages using conditional formatting.

7.1.1 **#COMPUTATION**

#COMPUTATION occurs when a slicing dimension specified in the `RelativeValue` function is no longer available in the calculation context of the block where the function is placed.

#COMPUTATION also occurs when a merged object containing a hierarchy is included in a report.

#COMPUTATION is also related to the misuse of context operators in a formula.

**Related Topics**

- `RelativeValue`

7.1.2 **#CONTEXT**

#CONTEXT appears in a measure when the measure has a non-existent calculation context.

#CONTEXT is related to the #INCOMPATIBLE and #DATASYNC error messages, which appear in dimensions when a block contains a non-existent calculation context.

In the case of #INCOMPATIBLE the context is non-existent because the dimensions are incompatible; in the case of #DATASYNC the context is non-existent because the dimensions are from multiple unsynchronized data providers.
Example: **Non-existent calculation context in a query**

If a block based on the Island Resorts Marketing universe contains the Reservation Year and Revenue objects, the #CONTEXT error message appears because it is not possible to aggregate revenue by reservation year. (Reservations have not yet generated any revenue.)

---

### 7.1.3 #DATASYNC

#DATASYNC occurs when you place a dimension from a different data provider in a block containing dimensions from another data provider, and the two data providers are not synchronized through a merged dimension. #DATASYNC appears in all dimensions in the block and #CONTEXT in the measures.

**Example:** **Dimensions from different data providers in a block**

If a report based on the Island Resorts Marketing universe contains data providers with the objects (Year, Revenue) and (Quarter), a block containing Year, Quarter and Revenue displays #DATASYNC in the Year and Quarter columns because the two data providers are not synchronized through a merged dimension.

---

### 7.1.4 #DIV/0

#DIV/0 occurs when a formula tries to divide a number by zero, which is mathematically impossible. Zero can never appear as a divisor.

**Example:** **Determining revenue per item**

You have a report showing sales revenues, numbers of items sold and the revenue per item (which is calculated by dividing the sales revenue by the number of items sold).

You had a very bad quarter in which you didn’t create any revenue; the Revenue per Item column returns #DIV/0 for this quarter, because the formula is attempting to divide by zero; that is, divide the revenue by zero number of items sold.

---

### 7.1.5 #ERROR

#ERROR is the default error message that covers all errors not covered by other error messages.
7.1.6 #EXTERNAL

#EXTERNAL occurs when a formula references an external function that is not available to use in Web Intelligence.

7.1.7 #INCOMPATIBLE

#INCOMPATIBLE occurs when a block contains incompatible objects.

Example: Incompatible objects in a query

If a block based on the Island Resorts Marketing universe contains the Year and Reservation Year dimensions, the columns containing these dimensions show #INCOMPATIBLE because these objects are incompatible.

7.1.8 #MIX

#MIX occurs when an aggregated measure has different units. For example, a cell shows #MIX if it aggregates currency values denominated in different currencies.

7.1.9 #MULTIVALUE

#MULTIVALUE occurs when you place a formula that returns more than one value in a cell that outputs one value only.

Example: Multivalue in a cell

You have a report showing Country, Resort and Revenue and you add a cell to the report containing the formula [Revenue] ForEach ([Country]). This cell returns #MULTIVALUE because Country has two values in the report: ‘US’ and ‘France’.
One cell cannot display the revenues for both the US and France. Placed outside the table, a cell containing revenue can only aggregate the revenues in the table in some way (for example by summing or averaging them).

If the report is broken into sections on Country, the formula is correct when placed in a section because there is only one value of Country per section. Outside a section, however, the formula still returns #MULTIVALUE

---

**7.1.10 #OVERFLOW**

#OVERFLOW occurs when a calculation returns a value that is too large for the software to handle. This value, in exponential form, is 1.7E308 (1.7 followed by 307 zeros).

---

**7.1.11 #PARTIALRESULT**

#PARTIALRESULT occurs when all rows associated with a report object were not retrieved.

If #PARTIALRESULT occurs often in your reports and you have the appropriate security rights, modify the Max Rows Retrieved query property to allow the retrieval of more data. If you do not have the right to modify the query, see your administrator.

If your report contains smart measures it is more likely to display #PARTIALRESULT because smart measures require the retrieval of larger amounts of data than classic measures.

---

**7.1.12 #RANK**

#RANK occurs when you try to rank data based on an object that depends on the order of values. (Objects that use the **Previous** function or any running aggregate function depend on the order of values.) Ranking causes these objects to recalculate their values, which then changes the ranking, resulting in a circular dependency. Such a dependency can occur either when you use the Rank dialog box to create a ranking, or when you use the Rank function.

**Example: Ranking on running average or previous values**

If you attempt to rank a block on a column that contains the **Previous** function or any running aggregate function, the entire block returns #RANK.
7.1.13 #RECURSIVE

#RECURSIVE occurs when it is not possible to perform a calculation due to a circular dependency.

Example: Using the NumberOfPages() function

If you place the NumberOfPages function in a cell whose Autofit Height or Autofit Width properties are set, the cell returns #RECURSIVE because the placing of this formula in an Autofit cell creates a circular dependency. The function needs the exact size of the report to return a value, but the size of the cell, which affects the size of the report, is determined by the cell content.

7.1.14 #REFRESH

#REFRESH appears in report cells whose values are derived from objects that were stripped from a query and then re-added to the query. Objects are stripped from a query when the Enable query stripping query property is selected and the objects do not contribute to any reports based on the query.

The cells are re-populated with values from the objects when the query is refreshed.

7.1.15 #REPFORMULA

Unable to find a Web Intelligence equivalence.

7.1.16 #SECURITY

#SECURITY occurs when you attempt to use a function for which you do not have security rights.

Example: Using the DataProviderSQL() function

If a user who does not have the right to view data provider SQL places the DataProviderSQL() function in a cell, the #SECURITY message appears in the cell.
7.1.17 #SYNTAX

#SYNTAX occurs when a formula references an object that no longer exists in the report.

Example: Referencing a non-existent object

You have a report that originally showed Year, Quarter and Sales revenue, with an additional column showing difference between the revenue and the average yearly revenue. This figure is given by the variable Difference from Yearly Average.

If the Difference from Yearly Average variable is deleted from the report, the column containing it returns #SYNTAX.

7.1.18 #TOREFRESH

#TOREFRESH appears in cells based on smart measures when the value returned by the smart measure is not available. This situation occurs when the “grouping set” containing the value is not available in the data provider.

You remove the #TOREFRESH error by refreshing the data.

7.1.19 #UNAVAILABLE

#UNAVAILABLE appears when it is not possible to calculate the value of a smart measure.

This occurs when it is not possible to display the values in a filtered smart measure without applying a filter to the query. Because this carries a risk of impacting other reports based on the same query, no filter is applied.
Comparing values using functions

8.1 Comparing values using the Previous function

The Previous function returns a comparative previous value of an expression. The value returned depends on the layout of the report.

For more powerful comparison capabilities, use the RelativeValue function. RelativeValue returns a previous or subsequent comparative value of an expression. The value returned does not depend on the layout of the report.

Related Topics
• Previous
• RelativeValue
• Comparing values using the RelativeValue function

8.2 Comparing values using the RelativeValue function

The RelativeValue function returns comparative values of an expression. The function returns these values independently of the layout of a report.

When using RelativeValue, you specify the following:
• The expression whose comparative value you want to find (the expression must be a measure or a detail of a dimension available in the block)
• The list of “slicing dimensions”
• The offset.

The function uses the slicing dimensions, the offset, and the “sub-axis dimensions” (which are implied by the slicing dimensions) to return a comparative value. The sub-axis dimensions are all the other dimensions in the calculation context apart from the slicing dimensions.

Expressed in general terms, RelativeValue returns the value of the expression in the row which, in the list of values of the slicing dimensions, is offset rows removed from the current row, and where the values of the sub-axis dimensions are the same as in the current row.
Note:
All slicing dimensions must always be in the calculation context of the block in which the function is placed. If a slicing dimension is subsequently removed, the function returns #COMPUTATION.

Example:
In this example, the RelativeValue column contains the following formula:

\[
\text{RelativeValue}([\text{Revenue}] ; ([\text{Year}]) ; -1)
\]

• The expression is [Revenue];
• The slicing dimension is [Year];
• The offset is -1 (the function returns the immediately previous value in the list).

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales Person</th>
<th>Revenue</th>
<th>RelativeValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>Q1</td>
<td>Smith</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q2</td>
<td>Jones</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q3</td>
<td>Wilson</td>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q4</td>
<td>Harris</td>
<td>3000</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Q1</td>
<td>Smith</td>
<td>4000</td>
<td>1000</td>
</tr>
<tr>
<td>2008</td>
<td>Q2</td>
<td>Jones</td>
<td>3400</td>
<td>2000</td>
</tr>
<tr>
<td>2008</td>
<td>Q3</td>
<td>Wilson</td>
<td>2000</td>
<td>1500</td>
</tr>
<tr>
<td>2008</td>
<td>Q4</td>
<td>Harris</td>
<td>1700</td>
<td>3000</td>
</tr>
</tbody>
</table>

Expressed as a business question, the formula returns the revenue generated by the same sales person in the same quarter in the previous year.

Expressed as a calculation in words, the formula returns the value of [Revenue] (the expression) in the row where the value of [Year] (the slicing dimension) is the previous value from the list of values of the [Year] object, and where the values of [Quarter] and [Sales Person] (the sub-axis dimensions) are the same as in the current row.

Related Topics
• RelativeValue
The `RelativeValue` function uses the list of values of the slicing dimensions to find the comparative row. The function returns the comparative value of the expression specified in the function that is offset number of rows away in the list of slicing dimensions.

As a result, the sort order of the slicing dimensions is crucial in determining the function output.

**Example: Multiple slicing dimensions**

In the table below, the `RelativeValue` column has the following formula:

\[
\text{RelativeValue([Revenue];([Year];[Quarter]);-1)}
\]

- The expression is `[Revenue]`;
- The slicing dimensions are `([Year];[Quarter])`;
- The offset is `-1` (the function returns the immediately previous value in the list).

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales Person</th>
<th>Revenue</th>
<th>RelativeValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>Q1</td>
<td>Smith</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q2</td>
<td>Smith</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q3</td>
<td>Smith</td>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q4</td>
<td>Smith</td>
<td>3000*</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q1</td>
<td>Jones</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q2</td>
<td>Jones</td>
<td>3400</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q3</td>
<td>Jones</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q4</td>
<td>Jones</td>
<td>1700</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Q1</td>
<td>Smith</td>
<td>5000**</td>
<td>3000*</td>
</tr>
<tr>
<td>2008</td>
<td>Q2</td>
<td>Smith</td>
<td>3000***</td>
<td>5000**</td>
</tr>
<tr>
<td>2008</td>
<td>Q3</td>
<td>Smith</td>
<td>2700****</td>
<td>3000***</td>
</tr>
<tr>
<td>2008</td>
<td>Q4</td>
<td>Smith</td>
<td>6800</td>
<td>2700****</td>
</tr>
</tbody>
</table>

Expressed as a business question, the formula returns the revenue generated by the same sales person in the previous quarter.

Expressed as a calculation in words, the formula returns the value of `[Revenue]` in the row where the values of `[Year]` and `[Quarter]` represent the previous value in the `([Year];[Quarter])` list of values, and where the value of `[Sales Person]` is the same as in the current row.

The function uses the list of values of the slicing dimensions to find the comparative revenue:
The sort order of the slicing dimensions determines the output of the function. The * in the tables show the sort order.

Related Topics
• RelativeValue

8.2.2 Slicing dimensions and sections

A slicing dimension can be in the section master cell of a report.

Example:

In the table below, the RelativeValue column has the following formula:

```
RelativeValue([Revenue];([Year];[Quarter]);-1)
```

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>Q1</td>
</tr>
<tr>
<td>2007</td>
<td>Q2</td>
</tr>
<tr>
<td>2007</td>
<td>Q3</td>
</tr>
<tr>
<td>2007</td>
<td>Q4</td>
</tr>
<tr>
<td>2008</td>
<td>Q1</td>
</tr>
<tr>
<td>2008</td>
<td>Q2</td>
</tr>
<tr>
<td>2008</td>
<td>Q3</td>
</tr>
<tr>
<td>2008</td>
<td>Q4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Sales Person</th>
<th>Revenue</th>
<th>RelativeValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Smith</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td>Smith</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>Smith</td>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>Smith</td>
<td>3000*</td>
<td></td>
</tr>
</tbody>
</table>
The function uses the list of values of the slicing dimensions to find the comparative revenue:

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>Q1</td>
</tr>
<tr>
<td>2007</td>
<td>Q2</td>
</tr>
<tr>
<td>2007</td>
<td>Q3</td>
</tr>
<tr>
<td>2007</td>
<td>Q4</td>
</tr>
<tr>
<td>2008</td>
<td>Q1</td>
</tr>
<tr>
<td>2008</td>
<td>Q2</td>
</tr>
<tr>
<td>2008</td>
<td>Q3</td>
</tr>
<tr>
<td>2008</td>
<td>Q4</td>
</tr>
</tbody>
</table>

The sort order of the slicing dimensions determines the output of the function. The * in the tables show the sort order.

**Related Topics**

- RelativeValue
8.2.3 Order of slicing dimensions

Because the sort order of the list of values of the slicing dimensions determines the output of \texttt{RelativeValue}, the order in which the slicing dimensions are specified impacts the output of the function.

Example: \textbf{Order of slicing dimensions}

In the table below, the \texttt{RelativeValue} column has the following formula:

\texttt{RelativeValue([Revenue];([Year];[Quarter]);-1)}

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales Person</th>
<th>Revenue</th>
<th>RelativeValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>Q1</td>
<td>Smith</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q2</td>
<td>Smith</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q3</td>
<td>Smith</td>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q4</td>
<td>Smith</td>
<td>3000*</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q1</td>
<td>Jones</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q2</td>
<td>Jones</td>
<td>3400</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q3</td>
<td>Jones</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q4</td>
<td>Jones</td>
<td>1700</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Q1</td>
<td>Smith</td>
<td>5000**</td>
<td>3000*</td>
</tr>
<tr>
<td>2008</td>
<td>Q2</td>
<td>Smith</td>
<td>3000***</td>
<td>5000**</td>
</tr>
<tr>
<td>2008</td>
<td>Q3</td>
<td>Smith</td>
<td>2700****</td>
<td>3000***</td>
</tr>
<tr>
<td>2008</td>
<td>Q4</td>
<td>Smith</td>
<td>6800</td>
<td>2700****</td>
</tr>
</tbody>
</table>

Expressed as a business question, the formula returns the revenue generated by the same sales person in the previous quarter.

The sort order of the slicing dimensions is as follows:

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>Q1</td>
</tr>
<tr>
<td>2007</td>
<td>Q2</td>
</tr>
<tr>
<td>2007</td>
<td>Q3</td>
</tr>
<tr>
<td>2007</td>
<td>Q4</td>
</tr>
</tbody>
</table>

*
The function is changed to:

\[ \text{RelativeValue}([\text{Revenue}];([\text{Quarter}];[\text{Year}]);-1) \]

The sort order of the slicing dimensions becomes:

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Year</th>
<th>Year Quarter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2007</td>
<td>+</td>
</tr>
<tr>
<td>Q1</td>
<td>2008</td>
<td>**</td>
</tr>
<tr>
<td>Q2</td>
<td>2007</td>
<td>***</td>
</tr>
<tr>
<td>Q2</td>
<td>2008</td>
<td>****</td>
</tr>
<tr>
<td>Q3</td>
<td>2007</td>
<td>*****</td>
</tr>
<tr>
<td>Q3</td>
<td>2008</td>
<td>*********</td>
</tr>
<tr>
<td>Q4</td>
<td>2007</td>
<td>**********</td>
</tr>
<tr>
<td>Q4</td>
<td>2008</td>
<td>***********</td>
</tr>
</tbody>
</table>

The sort order has the following impact on the function result:

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales Person</th>
<th>Revenue</th>
<th>RelativeValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>Q1</td>
<td>Smith</td>
<td>1000*</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q2</td>
<td>Smith</td>
<td>2000***</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q3</td>
<td>Smith</td>
<td>1500****</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q4</td>
<td>Smith</td>
<td>3000******</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q1</td>
<td>Jones</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q2</td>
<td>Jones</td>
<td>3400</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q3</td>
<td>Jones</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q4</td>
<td>Jones</td>
<td>1700</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Q1</td>
<td>Smith</td>
<td>5000**</td>
<td>1000*</td>
</tr>
</tbody>
</table>
Expressed as a business question, the formula now returns the revenue generated by the same sales person in the same quarter of the previous year.

The change in the sort order of the slicing dimension changes the meaning of the formula. The * in the tables indicate the sort order.

8.2.4 Slicing dimensions and sorts

Because the sort order of the list of values of the slicing dimensions determines the function output, a sort applied to any dimension in the slicing dimensions impacts the function output.

Example: A custom sort applied to a slicing dimension

In the table below, the RelativeValue column has the following formula:

RelativeValue(Revenue; Year; Quarter; -1)

A custom sort (Q1, Q2, Q4, Q3) is applied to [Quarter], giving the following result for the function:
### The sorted list of slicing dimensions is as follows:

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales Person</th>
<th>Revenue</th>
<th>RelativeValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>Q1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q3</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Q1</td>
<td>**</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Q2</td>
<td>***</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Q4</td>
<td>****</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Q3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The * in the tables show the sort order.

**Related Topics**
- RelativeValue

---

### 8.2.5 Using RelativeValue in crosstabs

The RelativeValue function works in crosstabs in exactly the same way as in vertical tables. The layout of the data in a crosstab has no impact on the function output.

**Related Topics**
- RelativeValue
# More Information

<table>
<thead>
<tr>
<th>Information Resource</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAP product information</td>
<td><a href="http://www.sap.com">http://www.sap.com</a></td>
</tr>
<tr>
<td>Access the most up-to-date English documentation covering all SAP BusinessObjects products at the SAP Help Portal:</td>
<td></td>
</tr>
<tr>
<td>• <a href="http://help.sap.com/bobi">http://help.sap.com/bobi</a> (Business Intelligence)</td>
<td></td>
</tr>
<tr>
<td>Certain guides linked to from the SAP Help Portal are stored on the SAP Service Marketplace. Customers with a maintenance agreement have an authorized user ID to access this site. To obtain an ID, contact your customer support representative.</td>
<td></td>
</tr>
<tr>
<td>To find a comprehensive list of product documentation in all supported languages, visit: <a href="http://help.sap.com/boall">http://help.sap.com/boall</a>.</td>
<td></td>
</tr>
<tr>
<td>The SAP Support Portal contains information about Customer Support programs and services. It also has links to a wide range of technical information and downloads. Customers with a maintenance agreement have an authorized user ID to access this site. To obtain an ID, contact your customer support representative.</td>
<td></td>
</tr>
<tr>
<td>These articles were formerly known as technical papers.</td>
<td></td>
</tr>
<tr>
<td>Information Resource</td>
<td>Location</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>----------</td>
</tr>
</tbody>
</table>
| Notes                                | https://service.sap.com/notes
These notes were formerly known as Knowledge Base articles. |
| Forums on the SAP Community Network  | https://www.sdn.sap.com/irj/scn/forums |
| Training                             | http://www.sap.com/services/education
From traditional classroom learning to targeted e-learning seminars, we can offer a training package to suit your learning needs and preferred learning style. |
| Consulting                           | http://www.sap.com/services/bysubject/businessobjectsconsulting
Consultants can accompany you from the initial analysis stage to the delivery of your deployment project. Expertise is available in topics such as relational and multidimensional databases, connectivity, database design tools, and customized embedding technology. |
Index

A
Abs function 122
Aggregate function 49
aggregate functions
description of 49
aggregation functions
including multiple dimensions in 74
All operator 160, 171, 172, 173
Ancestor function 150
And operator 19, 168, 169
Asc function 72
ASCII values 72
    returning characters associated
    with 73
average
    calculating a running average 60
Average function 50, 174
    using with extended syntax
    keywords 36, 183

B
base 10 logarithms 134
base n logarithms 134
Between operator 19, 168, 170
Block keyword 36, 183
BlockName function 151
blocks
    displaying the names of 151
Body keyword 37, 184
boolean expressions
    linking with And operator 169
    returning the opposite of 169
boolean values
    identifying 118
    testing 156, 158
Bottom operator 136, 172
Break keyword 35, 185
Break operator 172
breaks
    default calculation contexts in 28
building custom calculations using
    formulas 12
calculating a base 10 logarithm 134
calculating a base n logarithm 134
calculating a cosine 123
calculating a factorial 130
calculating a maximum value 54
calculating a median 55
calculating a minimum value 55
calculating a mode 56
calculating a natural logarithm 133
calculating a percentage 14, 57
calculating a percentile 59
calculating a population standard
deviation 69
calculating a population variance 71
calculating a product 59
calculating a running average 60
calculating a running count 61
calculating a running maximum 63
calculating a running minimum 64
calculating a running product 65
calculating a running sum 66
calculating a square root 139
calculating a standard deviation 68
calculating a sum 70
calculating a tangent 140
calculating a variance 70
calculating an average 50
calculating an exponential function 130
calculating rounding errors 127, 128
calculating the absolute value of a
    number 122
calculating the sine of an angle 139
calculation contexts
    and smart measures 45
    changing with extended syntax 24
default 24
defined 21
    input context 21
    output context 21, 22
calculations
    custom 11
    standard 11
Ceil function 123
cells
    including functions in 13
    including text in 13
Char function 73
character functions
description of 49
character strings
    applying URL encoding rules to 86
    calculating length of 79
    capitalizing first letter of 77
    capitalizing all first letters in 86
    capitalizing first letters 86
    converting to lowercase 80
    extracting sections from 84
    joining/concatenating 18, 74, 168
    matching to a pattern 80
    padding with other strings 78, 83
    removing leading spaces from 79, 85
    removing trailing spaces from 84, 85
    repeating 74
    replacing parts of 82
    returning leftmost characters of 77
    returning rightmost characters of
    82
    transforming to uppercase 85
    turning to numbers 140
characters
    displaying from ASCII values 73
    returning ASCII values of 72
charts
displaying the names of 151
Children function 142
Col operator 60, 63, 64, 65, 66, 176
ColumnNumber function 152
columns
displaying the numbers of 152
comparing values using Previous 197
comparing values using RelativeValue 197, 198, 200, 202, 204, 205
concatenating character strings 18, 74, 168
Concatenation function 74
conditional operators 18, 168
Connection function 98
currency
converting from euros 124
converting to euros 125
Cos function 123
cosine 123
Count function 51, 171, 173, 174
Count standard calculation 11
counting rows in tables 159
counting values 61
crosstabs
and the RelativeValue function 205
default calculation contexts in 26
currencies
converting between European currencies 124, 125
CurrentDate function 87
CurrentTime function 87
CurrentUser function 152
custom calculations 11
  using formulas to build 12
data
  refreshing 42
data provider functions
description of 49
data providers 104
  displaying number of rows in 105
  displaying the universe name 107
  viewing SQL generated by 100
DataProvider function 99
DataProviderKeyDate function 99
DataProviderKeyDateCaption function 100
DataProviderSQL function 100
DataProviderType function 101
date and time functions
description of 49
dates
calculating relative dates 94
dates (continued)
  formatting 75, 96
  identifying 116
DayName function 88
DayNumberOfMonth function 88
DayNumberOfWeek function 89
DayNumberOfYear function 89
DaysBetween function 90
default calculation contexts
  in breaks 28
  in crosstabs 26
  in horizontal tables 26
  in sections 27
  in vertical tables 25
modifying with extended syntax 29
Default standard calculation 11
Depth function 143
Descendants function 143
dimensions
adding to the calculation context 32, 181
  and #DATASYNC error message 192
  and #INCOMPATIBLE error message 193
  and grouping sets 45
  including multiple dimensions in aggregation functions 74
  removing from the calculation context 32, 181
  slicing dimensions 197, 198
  specifying in calculation context 30, 179
displaying column numbers 152
displaying function syntax 13
displaying page numbers 161
displaying prompt responses 107
displaying the author of documents 108
displaying the creation date of documents 109
displaying the DocumentLocale 153
displaying the dominant Preferred ViewingLocale 154
displaying the last date a document was saved 110
displaying the last time a document was saved 111
displaying the login of a document author 108
displaying the login of a document owner 110
displaying the names of charts 151
displaying the names of report objects 159
displaying the names of tables 151
displaying the number of pages in a report 160
displaying the owner of documents 110
displaying the Preferred Viewing Locale 156
displaying the ProductLocale 154
displaying the row number 167
displaying universe names 107
Distinct operator 173
document functions
description of 49
DocumentLocale
displaying 153
DocumentAuthor function 108
DocumentDate function 110
DocumentName function 110
DocumentOwner function 110
DocumentPartiallyRefreshed function 111
documents
displaying the author of 108
displaying the creation date of 109
displaying the last save date of 110
displaying the last save time of 111
displaying the name of 110
displaying the owner of 110
  viewing queries in 113
DocumentTime function 111
dominant Preferred ViewingLocale
displaying 154
drill filters 172
  and smart measures 48
  displaying 171
  ignoring 172
  Drill operator 160, 172
  DrillFilter function 112
error messages
  #COMPUTATION 165, 191
  #CONTEXT 191, 192
  #DATASYNC 191, 192
  #DIV/0 192
  #ERROR 124, 125, 127, 128, 140, 192
  #EXTERNAL 193
  #INCOMPATIBLE 191, 192, 193
  #MIX 193
  #MULTIVALUE 167, 193
  #OVERFLOW 194
  #PARTIALRESULT 194
  #RANK 194
  #RECURSIVE 195
  #REFRESH 195
  #REPFORMULA 195

210
2012-03-16
error messages (continued)
    #SECURITY 195
    #SYNTAX 196
    #TOREFRESH 42, 45, 196
    #UNAVAILABLE 46, 196

errors
    identifying 117
    EuroConvertFrom function 124, 188
    EuroConvertTo function 125, 188
    EuroFromRoundError function 127, 188
European currencies
    converting between 124, 125
    euros
        converting from 124
        converting to 125
    EuroToRoundError function 128, 188
    Even function 115
    even numbers
        identifying 115, 121
    Exp function 130
    exponential function 130
    extended syntax 19
        Block keyword 36, 183
        Body keyword 37, 184
        Break keyword 35, 185
        ForAll operator 29, 32, 181
        ForEach operator 29, 32, 178, 181
        In operator 29, 30, 178, 179
        modifying default calculation context with 29
        Report keyword 33, 38, 186
        Section keyword 34, 187
    extended syntax keywords 33, 183
        making reports generic with 38
        using with Average function 36, 183
        using with Sum function 33, 34, 35, 37, 184, 185, 186, 187

F

Fact function 130
    factorial 130
    Fill function 74
    filtering smart measures 47
    filters
        and smart measures 48
        block 114
        displaying all 171
        displaying drill filters 171
drill 172
drill filters 48
    filtering smart measures 47
    on dimensions 46
    report 114, 172

filters (continued)
    section 114
    filters on dimensions
        affect on smart measures of 46
    First function 53
    Floor function 131
    ForAll operator 32, 38, 181
    ForceMerge function 152
    ForEach operator 32, 181
    FormatDate function 75
    FormatNumber function 76
    formatting numbers 76
    Formula Editor
        displaying function syntax in 13
    formulas
        and smart measures 45
        building custom calculations using 12
        error messages generated by 191
        simplifying with variables 12, 17
        smart measures in 45
        use of operators in 18, 167
    free-standing cells
        and #MULTIVALUE error message 193
    function syntax
        example of 13
    functions
        Abs 122
        Aggregate 49
        Ancestor 150
        Asc 72
        Average 36, 50, 174, 183
        BlockName 151
categories of 49
        Ceiling 123
        Char 73
        Children 142
        ColumnNumber 152
        Concatenation 74
        Connection 98
        Cos 123
        Count 51, 171, 173, 174
        CurrentDate 87
        CurrentTime 87
        CurrentUser 152
        DataProvider 99
        DataProviderKeyDate 99
        DataProviderKeyDateCaption 100
        DataProviderSQL 100
        DataProviderType 101
        DayName 88
        DayNumberOfMonth 88
        DayNumberOfWeek 89
        DayNumberOfYear 89
functions (continued)
    DaysBetween 90
defined 13
    Depth 143
    Descendants 143
    DocumentAuthor 108
    DocumentCreationDate 109
    DocumentCreationDate function 109
    DocumentCreationTime 109
    DocumentCreationTime function 109
    DocumentDate 110
    DocumentName 110
    DocumentOwner 110
    DocumentPartiallyRefreshed 111
    DocumentTime 111
    DrillFilters 112
    EuroConvertFrom 124, 188
    EuroConvertTo 125, 188
    EuroFromRoundError 127, 188
    EuroToRoundError 128, 188
    Even 115
    examples of 14
    Exp 130
    Fact 130
    Fill 74
    First 53
    Floor 131
    ForceMerge 152
    FormatDate 75
    FormatNumber 76
    function syntax 13
    GetContentLocale 153
    GetDominantPreferredViewingLocale 154
    GetLocale 154
    GetLocalized 155
    GetPreferredViewingLocale 156
    HTMLEncode 76
    If 18, 158, 168
    including in cells 13
    InitCap 77
    Interpolation 131, 174, 175
    Interpolation function 175
    IsDate 116
    IsError 117
    IsLeaf 146
    IsLogical 118
    IsNull 118
    IsNumber 119
    IsPromptAnswered 101
    IsString 120
    IsTime 121
    Key 146
    Lag 147
functions (continued)
  Last 53
  LastDayOfMonth 90
  LastDayOfWeek 91
  LastExecutionDate 102
  LastExecutionDuration 103
  LastExecutionTime function 104
  Left 77
  LeftPad 78
  LeftTrim 79
  LineNumber 159
  Ln 133
  Log 134
  Log10 134
  Lower 80
  Match 80
  Max 32, 54, 181
  Median 55
  Min 55
  mixing with text in cells 13
  Mod 135
  Mode 56
  Month 92
  MonthNumberOfYear 92
  MonthsBetween 93
  NameOf 159
  NoFilter 160, 171, 172
  NumberOfDataProviders 104
  NumberOfPages 160, 195
  NumberOfRows 105
  Odd 121
  Page 161
  Parent 148
  Percentage 14, 57, 72, 176
  Percentage function 176
  Percentile 59
  Pos 81
  Power 135
  Previous 161, 175, 177, 194, 197
  Product 59
  PromptSummary 112
  Quarter 93
  QuerySummary 113
  Rank 136, 172
  referring to member sets in 189
  referring to members in 189
  RefValue 164
  RefValueDate 105
  RefValueUserResponse 106, 174
  RelativeDate 94
  RelativeValue 165, 191, 197, 200,
  202, 204, 205
  Replace 82
  ReportFilter 114
  ReportFilterSummary 114

functions (continued)
  ReportName 166
  Right 82
  RightPad 83
  RightTrim 84
  Round 137, 188
  RowIndex 167
  RunningAverage 60, 174, 176
  RunningAverage function 176
  RunningCount 61, 174, 176
  RunningCount function 176
  RunningMax 63, 176
  RunningMax function 176
  RunningMin 64, 176
  RunningMin function 176
  RunningProduct 65, 176
  RunningProduct function 176
  RunningSum 66, 176
  RunningSum function 176
  ServerValue 68
  Siblings 149
  Sign 138
  Sin 139
  Sqrt 139
  StdDev 68
  StdDevP 69
  Substr 84
  Sum 14, 32, 70, 181, 184, 185, 186, 187
  Sum function 70
  Tan 140
  ToDate 96
  ToNumber 140
  Trim 85
  Truncate 141, 188
  UniqueNameOf 167
  UniverseName 107
  Upper 85
  URLEncode 86
  UserResponse 14, 107, 174
  Var 70
  VarP 71
  Week 97
  WordCap 86
  Year 97

G
  GetContentLocale function 153
  GetDominantPreferredViewingLocale
  function 154
  GetLocale function 154
  GetLocalized function 155
  GetPreferredViewingLocale function
  156

H
  horizontal tables
    default calculation contexts in 26
    HTMLEncode function 76

I
  identifying boolean values 118
  identifying dates 116
  identifying errors 117
  identifying even numbers 115, 121
  identifying null values 118
  identifying numbers 119
  identifying odd numbers 115, 121
  identifying strings 120
  identifying time values 121
  If function 18, 158, 168
  If...Then...Else condition 156
  In operator 30, 179
  IncludeEmpty operator 174
  Index operator 107
  InitCap function 77
  InList operator 19, 168
  InList operator 170
  input context
    defined 21
  interpolating measure values 131, 175
  Interpolation function 131, 174, 175
  IsDate Function 116
  IsError Function 117
  IsLeaf function 146
  IsLogical function 118
  IsNull Function 118
  IsNumber Function 119
  IsPromptAnswered function 101
  IsString Function 120
  IsTime Function 121

J
  joining character strings 18, 74, 168

K
  Key function 146
  keywords
    Block 36, 183
keywords (continued)

Body 37, 184
Break 35, 185
extended syntax keywords 33, 183
making reports generic with 38
Report 33, 38, 186
Section 34, 187

L

Lag function 147
Last function 53
LastDayOfMonth function 90
LastDayOfWeek function 91
LastExecutionDate function 102
LastExecutionDuration function 103
LastExecutionTime function 104
leading spaces
removing from character strings 79, 85
Left function 77
LeftPad function 78
LeftTrim function 79
Length function 79
Linear operator 131, 174
LineNumber function 159
Ln function 133
locales
displaying the Document Locale 153
displaying the dominant Preferred Viewing Locale 154
displaying the Product Locale 154
Log function 134
Log10 function 134
logarithms 133, 134
logical functions
description of 49
logical operators 19, 168
Lower function 80

M

Match function 80
mathematical operators 18, 168
Max function 54
using with context operators 32, 181
Maximum standard calculation 11
maximum values
calculating a running maximum 63
measure values
interpolating 131, 175
measures
and default calculation contexts 24
returning previous values of 165
measures (continued)
returning subsequent values of 165
smart measures defined 41
median 55
Median function 55
member sets
referring to in functions 189
members
referring to in functions 189
Min function 55
Minimum standard calculation 11
minimum values
calculating a running minimum 64
misc functions
description of 49
Mod function 135
mode 56
Mode function 56
Month function 92
MonthNumberOfYear function 92
MonthsBetween function 93
multiplying numbers 59, 65

N

NameOf function 159
natural logarithms 133
NoFilter function 160, 171, 172
NotNull operator 161, 175
Not operator 19, 168, 169
NotOnBreak operator 131, 175
null values
identifying 118
NumberOfDataProviders function 104
NumberOfPages function 160
and #RECURSIVE error message 195
NumberOfRows function 105
numbers
calculating a modulus 135
determining if odd or even 115, 121
dividing 135
formatting 76
identifying 119
modulus 135
multiplying 59, 65
raising to a power 135
rounding 123, 137
rounding down 131
rounding down numbers 131
rounding numbers 137
truncating 141
truncating numbers 141
turning character strings to 140
numeric functions
description of 49

O

objects
displaying the name of 167
viewing filters on 114
Odd function 121
odd numbers
identifying 115, 121
operators
All 160, 171, 172, 173
And 19, 168, 169
Between 19, 168, 170
Bottom 136, 172
Break 172
Col 60, 61, 63, 64, 65, 66, 176
conditional 18, 168
defined 18, 167
Distinct 173
Drill 160, 172
ForAll 32, 38, 181
ForEach 32, 181
function-specific 19, 171
function-specific operators 19, 171
In 30, 179
IncludeEmpty 61, 174
Index 107, 174
Index operator 174
Inlist 19, 168
InList 170
Linear 131, 174
logical 19, 168
mathematical 18, 188
mathematical operators 18, 168
NoNull 161, 175
Not 19, 168, 169
NotOnBreak 131, 175
Or 19, 168, 169
PointToPoint 131, 175
range 182
Row 60, 61, 63, 64, 65, 66, 176
Self 161, 177
set 182
Top 136, 172
Where 178
Or operator 19, 168, 169
output context
defined 22

P

Page function 161
pages
displaying number of in reports 160
pages numbers
displaying in reports 161
Parent function 148
Sum function (continued)
using with extended syntax
keywords 33, 34, 35, 37, 38,
184, 185, 186, 187
Sum standard calculation 11

T

tables
    counting rows in 159
displaying the names of 151
horizontal 26
vertical 25
Tan function 140
tangents 140
testing boolean values 156, 158
time values
    identifying 121
ToDate function 96
ToNumber function 140
Top operator 136, 172
trailing spaces
    removing from character strings 84, 85
Trim function 85

Truncate function 141, 188
truncating values 188

U

UNION operator 43
UniqueNameOf function 167
UniverseName function 107
universes
    displaying the names of 107
Upper function 85
URLEncode function 86
URLs
    applying encoding rules to 86
UserResponse function 14, 107, 174
Using Functions, Formulas and Calculations guide
    about 9

V

values
    comparing using Previous 197
    comparing using RelativeValue 197, 198, 200, 202, 204, 205
values (continued)
    counting 51, 61
    precision of 188
    rounding 188
    truncating 188
Var function 70
variables
    and smart measures 45
    simplifying formulas with 12, 17
variance 70
VarP function 71
vertical tables
    default calculation contexts in 25

W

Week function 97
Where operator 178
WordCap function 86

Y

Year function 97