Using Functions, Formulas and Calculations in Web Intelligence
# Contents

<table>
<thead>
<tr>
<th>Chapter 1</th>
<th>About this guide</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 2</td>
<td>Using standard and custom calculations</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Using standard and custom calculations in your reports</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Standard calculations</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Using formulas to build custom calculations</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>Working with functions</td>
<td>13</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>Understanding calculation contexts</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>What are calculation contexts?</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>The input context</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>The output context</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Default calculation contexts</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Default contexts in a vertical table</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>Default contexts in a horizontal table</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Default contexts in a crosstab</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Default contexts in a section</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Default contexts in a break</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>Modifying the default calculation context with extended syntax</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Extended syntax operators</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>Web Intelligence extended syntax keywords</td>
<td>38</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Web Intelligence functions, operators and keywords</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Web Intelligence functions</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Aggregate functions</td>
<td>50</td>
</tr>
</tbody>
</table>
Chapter 6  Calculating values with smart measures  215

Smart measures defined ................................................................. 216
Grouping sets and smart measures ............................................. 216
  How Web Intelligence manages grouping sets ......................... 217
Smart measures and the scope of analysis .................................. 218
Smart measures and SQL ............................................................. 218
  Grouping sets and the UNION operator .................................... 218
Smart measures and formulas ..................................................... 221
  Smart measures and dimensions containing formulas ............ 221
  Smart measures in formulas .................................................... 222
Smart measures and filters ......................................................... 222
  Smart measures and filters on dimensions ............................ 222
  Smart measures and drill filters .............................................. 223

Chapter 7  Comparing values using Web Intelligence functions  225

Comparing values using the Previous function ......................... 226
Comparing values using the RelativeValue function ................ 226
  Slicing dimensions and the RelativeValue function ............... 228
  Slicing dimensions and sections ........................................... 230
Order of slicing dimensions ...................................................... 232
Slicing dimensions and sorts ..................................................... 234
Using RelativeValue in crosstabs .............................................. 236
Contents

Appendix A  Get More Help  237

Index  241
About this guide
The Using Functions, Formulas and Calculations in Web Intelligence guide provides detailed information on the advanced calculation capabilities in Web Intelligence. It also provides a syntax reference to the Web Intelligence functions and operators.

The guide presents this information generically, without reference to the Web Intelligence interface. For information on how to work with calculation-related features in your Web Intelligence documents (for example, how to add a variable or a formula to a report), see Performing On-Report Analysis With Web Intelligence, Building Reports with the Java Report Panel and Web Intelligence Rich Client User's Guide.
Using standard and custom calculations
Using standard and custom calculations in your reports

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

Standard calculations

You can use standard calculation functions to make quick calculations on the data in Web Intelligence reports. 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>
</tbody>
</table>
### Calculation

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 a standard calculation to a table column, the calculation result appears in a footer in the column. Web Intelligence adds a footer for the result of each calculation if you apply multiple calculations to the same column.

### Using formulas to build custom calculations

Custom calculations allow you to add additional calculations to your report beyond its base objects and the standard calculations provided by Web Intelligence.

You add a custom calculation by writing a formula that Web Intelligence evaluates when you run the report. 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 \( \frac{\text{Sales Revenue}}{\text{Number Sold}} \) gives this value by dividing the revenue by the number of items sold in order to give the revenue per item.

Related Topics
- What are calculation contexts? on page 24

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.

Example: Create a formula to return a statistical variance

Variance is a statistical term. The variance of a set of values measures the spread of those values around their average. Web Intelligence has the function \( \text{Var()} \) that 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 formula is as follows:

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

This formula is clearly unwieldy. By using variables you can simplify it to:

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

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.

---

**Working with functions**

A custom calculation sometimes contains report objects only, for example [Sales Revenue]/[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([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**

- *Web Intelligence function and formula operators* on page 181
- *Web Intelligence functions* on page 50
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 \( \text{Average([Revenue])} \) appears in a cell as \( =\text{Average([Revenue])} \). 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([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.

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 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 Abs function:

```
num Abs(number)
```

This syntax tells you that the 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.

Examples of functions

**Example: Showing prompt input with the 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:

"Quarterly Revenues for " + UserResponse( "eFashion";"Choose a State")

The report is as follows:

<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>$241,453</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>$107,005</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>$133,306</td>
</tr>
<tr>
<td></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>$254,722</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>$230,573</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>$331,067</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>$1,150,658.80</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>$255,655</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>$354,724</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>$273,186</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>$250,617</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>$1,134,085.40</td>
</tr>
</tbody>
</table>

**Example: Calculating a percentage using the Percentage function**

Web Intelligence has the Percentage function for calculating 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 $\text{Percentage ([Sales Revenue])}$. 
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.

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 \( \frac{\text{Sales Revenue}}{\text{Sum(Sales Revenue)}} \) gives the sales revenue as a percentage of the total revenue.

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

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

and the Percentage of Year column has the formula:

\[ \frac{\text{Sales revenue}}{\text{Sum([Sales revenue] In Section)}} \]
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* on page 34

**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. Web Intelligence has the function Var() that 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 formula is as follows:

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

which is clearly unwieldy.

**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.

**Web Intelligence function and formula operators**

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

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.

Logical operators

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

Function-specific operators

Some Web Intelligence 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.
Understanding calculation contexts
What are calculation contexts?

The calculation context is the data that a calculation takes into account to generate a result. Web Intelligence, 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 on page 24
- The output context on page 25

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:

```
Sum ([Revenue] In ([Year]))
```

```
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.

---

**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:

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

That is, the output context appears after the parentheses of the function whose output context you are specifying. In this case, the output context tells Web Intelligence to calculate minimum revenue by year.

If you add an additional column containing this formula to the block, the result is as follows:
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 tells Web Intelligence which revenue by year and quarter to output. In full, with both input and output formulas explicitly specified, the formula looks like this:

```
Min ( [Sales Revenue] In ([Year];[Quarter])) In ([Year])
```

Explained in words, this formula tells Web Intelligence to “calculate revenues by year by quarter, then output 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.

### Default calculation contexts

Depending on where you place a measure or formula, Web Intelligence assigns a default calculation context to the measure.
Measures are semantically dynamic. This means that the figures returned by a measure depend on the dimensions with which it is associated. This combination of dimensions represents the calculation context.

Web Intelligence associates a default context with a measure depending on where the measure is placed. You can change this 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>Total:</td>
<td>8000</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 revenue in the report</td>
</tr>
<tr>
<td>Section header total</td>
<td>8000</td>
<td>Year</td>
</tr>
</tbody>
</table>
### 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>
Example: Default contexts in a vertical table

The following table shows the default contexts in a vertical table:

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
<th>$36387203</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>$2660700</td>
<td>$2660699.50</td>
</tr>
<tr>
<td>2001</td>
<td>Q2</td>
<td>$2278693</td>
<td>$2278693.40</td>
</tr>
<tr>
<td>2001</td>
<td>Q3</td>
<td>$1367841</td>
<td>$1367840.70</td>
</tr>
<tr>
<td>2001</td>
<td>Q4</td>
<td>$1788580</td>
<td>$1788580.40</td>
</tr>
<tr>
<td>2002</td>
<td>Q1</td>
<td>$3326172</td>
<td>$3326172.20</td>
</tr>
<tr>
<td>2002</td>
<td>Q2</td>
<td>$2640651</td>
<td>$2640650.80</td>
</tr>
<tr>
<td>2002</td>
<td>Q3</td>
<td>$2679303</td>
<td>$2679303.00</td>
</tr>
<tr>
<td>2002</td>
<td>Q4</td>
<td>$4196120</td>
<td>$4196120.00</td>
</tr>
<tr>
<td>2003</td>
<td>Q1</td>
<td>$3742969</td>
<td>$3742968.90</td>
</tr>
<tr>
<td>2003</td>
<td>Q2</td>
<td>$4006718</td>
<td>$4006717.50</td>
</tr>
<tr>
<td>2003</td>
<td>Q3</td>
<td>$3953395</td>
<td>$3953395.30</td>
</tr>
<tr>
<td>2003</td>
<td>Q4</td>
<td>$3356041</td>
<td>$3356041.10</td>
</tr>
</tbody>
</table>

**Sum:** $36387203

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.

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>
</tbody>
</table>
### Default Calculation Contexts

<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 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:

<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>76,596</td>
<td>70,000</td>
<td>50,540</td>
</tr>
<tr>
<td>US</td>
<td>856,560</td>
<td>136,831</td>
<td>189,886</td>
<td>234,574</td>
<td>235,269</td>
</tr>
<tr>
<td>Sum</td>
<td>1,115,730</td>
<td>258,726</td>
<td>256,441</td>
<td>304,654</td>
<td>285,803</td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>1,115,730</td>
<td></td>
<td></td>
<td></td>
<td>259,170</td>
</tr>
<tr>
<td>US</td>
<td>856,560</td>
<td></td>
<td></td>
<td></td>
<td>235,269</td>
</tr>
<tr>
<td>Sum</td>
<td>1,115,730</td>
<td>258,726</td>
<td>256,441</td>
<td>304,654</td>
<td>1,115,730</td>
</tr>
</tbody>
</table>
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:
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>Quarter</th>
<th>Sales revenue</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>$2,660,700</td>
<td>8,096,123.6</td>
</tr>
<tr>
<td>Q2</td>
<td>$2,279,003</td>
<td>8,096,123.6</td>
</tr>
<tr>
<td>Q3</td>
<td>$1,367,841</td>
<td>8,096,123.6</td>
</tr>
<tr>
<td>Q4</td>
<td>$1,788,580</td>
<td>8,096,123.6</td>
</tr>
<tr>
<td>Sum:</td>
<td></td>
<td>8,096,123.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Sales revenue</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>$3,326,172</td>
<td>13,232,246</td>
</tr>
<tr>
<td>Q2</td>
<td>$2,840,651</td>
<td>13,232,246</td>
</tr>
<tr>
<td>Q3</td>
<td>$2,879,303</td>
<td>13,232,246</td>
</tr>
<tr>
<td>Q4</td>
<td>$4,186,120</td>
<td>13,232,246</td>
</tr>
<tr>
<td>Sum:</td>
<td></td>
<td>13,232,246</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Sales revenue</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>$3,742,989</td>
<td>15,059,142.8</td>
</tr>
<tr>
<td>Q2</td>
<td>$4,006,718</td>
<td>15,059,142.8</td>
</tr>
<tr>
<td>Q3</td>
<td>$3,953,395</td>
<td>15,059,142.8</td>
</tr>
<tr>
<td>Q4</td>
<td>$3,356,041</td>
<td>15,059,142.8</td>
</tr>
<tr>
<td>Sum:</td>
<td></td>
<td>15,059,142.8</td>
</tr>
</tbody>
</table>
The calculation is in the... | The input context is... | The output context is...
---|---|---
Header | Current instance of the break. | All the data is aggregated, then the calculation function returns a single value.
Footer | Current instance of the break. | All the data is aggregated, then the calculation function returns a single value.

**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>$ (000)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
<td>$2660700</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>$2279003</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>$1367841</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>$1788580</td>
</tr>
<tr>
<td>2001</td>
<td>Sum</td>
<td>$8096123</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>$ (000)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
<td>$3326172</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>$2840661</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>$2679303</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>$4186120</td>
</tr>
<tr>
<td>2002</td>
<td>Sum</td>
<td>$1323246</td>
</tr>
</tbody>
</table>

**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.
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.

**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>$8096123.60</td>
<td>$2660699.50</td>
</tr>
<tr>
<td>2002</td>
<td>$13232246.00</td>
<td>$4156120.00</td>
</tr>
<tr>
<td>2003</td>
<td>$15053142.60</td>
<td>$4006717.50</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></td>
<td>Q1</td>
<td>$2660700</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>$2279003</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>$1367841</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>$1788580</td>
</tr>
<tr>
<td>2001</td>
<td></td>
<td>Max: 2660699.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
<td>$3326172</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>$2879303</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>$4186120</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td></td>
</tr>
<tr>
<td>2002</td>
<td></td>
<td>Max: 4186120</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
<td>$3742989</td>
</tr>
<tr>
<td></td>
<td>Q2</td>
<td>$4006718</td>
</tr>
<tr>
<td></td>
<td>Q3</td>
<td>$3953395</td>
</tr>
<tr>
<td></td>
<td>Q4</td>
<td>$3356041</td>
</tr>
<tr>
<td>2003</td>
<td></td>
<td>Max: 4006717.5</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

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

This formula tells Web Intelligence to calculate the maximum sales revenue for each (Year,Quarter) combination, then output 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.
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]).

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.

---

### Web Intelligence 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.
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.

<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>$2,660,700</td>
<td>36,387,512.4</td>
</tr>
<tr>
<td>2001</td>
<td>Q2</td>
<td>$2,279,003</td>
<td>36,387,512.4</td>
</tr>
<tr>
<td>2001</td>
<td>Q3</td>
<td>$1,367,841</td>
<td>36,387,512.4</td>
</tr>
<tr>
<td>2001</td>
<td>Q4</td>
<td>$1,788,580</td>
<td>36,387,512.4</td>
</tr>
<tr>
<td>2002</td>
<td>Q1</td>
<td>$3,326,172</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>$2,979,303</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>$3,742,989</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>$3,356,041</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]).
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>Quarter</th>
<th>Sales revenue</th>
<th>Section Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>$2,660,700</td>
<td>8,095,814</td>
</tr>
<tr>
<td>Q2</td>
<td>$2,778,693</td>
<td>8,095,814</td>
</tr>
<tr>
<td>Q3</td>
<td>$1,567,841</td>
<td>8,095,814</td>
</tr>
<tr>
<td>Q4</td>
<td>$1,788,580</td>
<td>8,095,814</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]).
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.

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

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]).
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.

### 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>When placed in...</td>
<td>References this data...</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-----------------------------</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>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 block.

**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:

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales revenue</th>
<th>Percentage of Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>$2660700</td>
<td>7.31</td>
</tr>
<tr>
<td>2001</td>
<td>Q2</td>
<td>$2279003</td>
<td>6.26</td>
</tr>
<tr>
<td>2001</td>
<td>Q3</td>
<td>$1367341</td>
<td>3.76</td>
</tr>
<tr>
<td>2001</td>
<td>Q4</td>
<td>$1788580</td>
<td>4.92</td>
</tr>
<tr>
<td>2002</td>
<td>Q1</td>
<td>$3326172</td>
<td>9.14</td>
</tr>
<tr>
<td>2002</td>
<td>Q2</td>
<td>$2840551</td>
<td>7.81</td>
</tr>
<tr>
<td>2002</td>
<td>Q3</td>
<td>$2679303</td>
<td>7.81</td>
</tr>
<tr>
<td>2002</td>
<td>Q4</td>
<td>$4186120</td>
<td>11.5</td>
</tr>
<tr>
<td>2003</td>
<td>Q1</td>
<td>$3742989</td>
<td>10.29</td>
</tr>
<tr>
<td>2003</td>
<td>Q2</td>
<td>$4006718</td>
<td>11.01</td>
</tr>
<tr>
<td>2003</td>
<td>Q3</td>
<td>$3953395</td>
<td>10.86</td>
</tr>
<tr>
<td>2003</td>
<td>Q4</td>
<td>$3356041</td>
<td>9.22</td>
</tr>
<tr>
<td></td>
<td><strong>Sum:</strong></td>
<td></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>

The formula for the Percentage of Total column is:

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

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

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

This formula tells Web Intelligence to remove Year and Quarter from the output context; in other words, to calculate 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:

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Month</th>
<th>Sales revenue</th>
<th>Percentage of Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Q1</td>
<td>1</td>
<td>$10,9541.20</td>
<td>26.13</td>
</tr>
<tr>
<td>2001</td>
<td>Q1</td>
<td>2</td>
<td>$6,30073.20</td>
<td>29.97</td>
</tr>
<tr>
<td>2001</td>
<td>Q1</td>
<td>3</td>
<td>$10,27085.10</td>
<td>27.12</td>
</tr>
<tr>
<td>2001</td>
<td>Q2</td>
<td>4</td>
<td>$8,95259.80</td>
<td>28.1</td>
</tr>
<tr>
<td>2001</td>
<td>Q2</td>
<td>5</td>
<td>$8,66615.10</td>
<td>24.3</td>
</tr>
<tr>
<td>2001</td>
<td>Q2</td>
<td>6</td>
<td>$5,17818.50</td>
<td>21.77</td>
</tr>
<tr>
<td>2001</td>
<td>Q3</td>
<td>7</td>
<td>$5,25903.50</td>
<td>20.42</td>
</tr>
<tr>
<td>2001</td>
<td>Q3</td>
<td>8</td>
<td>$1,73756.40</td>
<td>11.1</td>
</tr>
<tr>
<td>2001</td>
<td>Q3</td>
<td>9</td>
<td>$6,88818.60</td>
<td>16.46</td>
</tr>
<tr>
<td>2001</td>
<td>Q4</td>
<td>10</td>
<td>$6,65206.40</td>
<td>18.04</td>
</tr>
<tr>
<td>2001</td>
<td>Q4</td>
<td>11</td>
<td>$4,64024.20</td>
<td>18.55</td>
</tr>
<tr>
<td>2001</td>
<td>Q4</td>
<td>12</td>
<td>$6,49349.80</td>
<td>21.01</td>
</tr>
<tr>
<td>2002</td>
<td>Q1</td>
<td>1</td>
<td>$13,35401.90</td>
<td>34.77</td>
</tr>
<tr>
<td>2002</td>
<td>Q1</td>
<td>2</td>
<td>$6,09012.80</td>
<td>28.97</td>
</tr>
<tr>
<td>2002</td>
<td>Q1</td>
<td>3</td>
<td>$1,361757.50</td>
<td>36.49</td>
</tr>
<tr>
<td>2002</td>
<td>Q2</td>
<td>4</td>
<td>$10,88308.90</td>
<td>33.53</td>
</tr>
<tr>
<td>2002</td>
<td>Q2</td>
<td>5</td>
<td>$10,81884.80</td>
<td>30.38</td>
</tr>
<tr>
<td>2002</td>
<td>Q2</td>
<td>6</td>
<td>$6,30457.10</td>
<td>29.03</td>
</tr>
<tr>
<td>2002</td>
<td>Q3</td>
<td>7</td>
<td>$8,01954.70</td>
<td>31.14</td>
</tr>
<tr>
<td>2002</td>
<td>Q3</td>
<td>8</td>
<td>$5,81093.60</td>
<td>37.16</td>
</tr>
<tr>
<td>2002</td>
<td>Q3</td>
<td>9</td>
<td>$1,496254.80</td>
<td>36.84</td>
</tr>
<tr>
<td>2002</td>
<td>Q4</td>
<td>10</td>
<td>$15,45871.80</td>
<td>42.57</td>
</tr>
<tr>
<td>2002</td>
<td>Q4</td>
<td>11</td>
<td>$10,81815.30</td>
<td>41.47</td>
</tr>
<tr>
<td>2003</td>
<td>Q4</td>
<td>12</td>
<td>$3,658332.90</td>
<td>50.43</td>
</tr>
<tr>
<td>2003</td>
<td>Q1</td>
<td>1</td>
<td>$1,501366.70</td>
<td>39.09</td>
</tr>
<tr>
<td>2003</td>
<td>Q1</td>
<td>2</td>
<td>$8,63451.90</td>
<td>41.07</td>
</tr>
<tr>
<td>2003</td>
<td>Q1</td>
<td>3</td>
<td>$1,378170.30</td>
<td>36.39</td>
</tr>
<tr>
<td>2003</td>
<td>Q2</td>
<td>4</td>
<td>$12,22329.40</td>
<td>36.37</td>
</tr>
<tr>
<td>2003</td>
<td>Q2</td>
<td>5</td>
<td>$1,814147.30</td>
<td>45.32</td>
</tr>
<tr>
<td>2003</td>
<td>Q2</td>
<td>6</td>
<td>$1,170240.60</td>
<td>49.2</td>
</tr>
<tr>
<td>2003</td>
<td>Q3</td>
<td>7</td>
<td>$1,247313.50</td>
<td>48.44</td>
</tr>
<tr>
<td>2003</td>
<td>Q3</td>
<td>8</td>
<td>$8,09366.40</td>
<td>51.74</td>
</tr>
<tr>
<td>2003</td>
<td>Q3</td>
<td>9</td>
<td>$1,896716.40</td>
<td>46.7</td>
</tr>
<tr>
<td>2003</td>
<td>Q4</td>
<td>10</td>
<td>$1,430300.10</td>
<td>39.39</td>
</tr>
<tr>
<td>2003</td>
<td>Q4</td>
<td>11</td>
<td>$1,045098.60</td>
<td>39.98</td>
</tr>
<tr>
<td>2003</td>
<td>Q4</td>
<td>12</td>
<td>$8,62642.20</td>
<td>28.56</td>
</tr>
</tbody>
</table>

Sum: 1200
Why is this? The problem lies in:

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

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.
Understanding calculation contexts

*Modifying the default calculation context with extended syntax*
Web Intelligence functions, operators and keywords
Web Intelligence functions

Web Intelligence divides functions 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>
</tbody>
</table>

Aggregate functions

Average

Description
Returns the average value of a measure

Function Group
Aggregate
Syntax
num Average(measure;[IncludeEmpty])

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>IncludeEmpty</td>
<td>Includes empty rows in the calculation</td>
<td>Keyword</td>
<td>No (Empty rows excluded by default)</td>
</tr>
</tbody>
</table>

Notes

- You can use extended syntax context operators with `Average`.
- You can specify `IncludeEmpty` as the second argument to the function. When you specify this argument, the function takes empty (null) rows into consideration in the calculation.

Examples

If the [Sales Revenue] measure has the values 41569, 30500, 40000 and 50138, `Average([Sales Revenue])` returns 40552.

Related Topics
- `IncludeEmpty operator` on page 189

Count

Description

Returns the number of values in a dimension or measure

Function Group

Aggregate
Syntax

integer Count(dimension|measure;[IncludeEmpty];[Distinct|All])

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>IncludeEmpty</td>
<td>Includes empty values in the calculation</td>
<td>Keyword</td>
<td>No</td>
</tr>
<tr>
<td>Distinct</td>
<td>All</td>
<td>Includes distinct values only (default for dimensions) or all values (default for measures) in the calculation</td>
<td>Keyword</td>
</tr>
</tbody>
</table>

Notes

- You can use extended syntax context operators with Count.
- You can specify IncludeEmpty as the second argument to the function. When you specify this argument, the function takes empty (null) rows into consideration in the calculation.
- The Distinct/All parameter is optional. If you do not specify this parameter, the default values are Distinct for dimensions and All for measures.

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.
Related Topics

- IncludeEmpty operator on page 189
- Distinct/All operators on page 188

First

Description
Returns the first value in a data set

Function Group
Aggregate

Syntax
`input_type 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 in 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.
Interpolation

Description
Calculates empty measure values by interpolation

Function Group
Numeric

Syntax
num Interpolation(measure;[PointToPoint|Linear];[NotOn 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>PointToPoint|Linear</td>
<td>The interpolation method:</td>
<td>Keyword</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>• PointToPoint - point-to-point interpolation</td>
<td></td>
<td>(PointToPoint is default)</td>
</tr>
<tr>
<td></td>
<td>• Linear - linear regression with least squares interpolation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NotOnBreak</td>
<td>Prevents the function from resetting the calculation on block and section 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>

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

\( \text{Interpolation}([\text{Value}]) \) supplies the following missing values using the default point-to-point interpolation method:

<table>
<thead>
<tr>
<th>Day</th>
<th>Value</th>
<th>\text{Interpolation}([\text{Value}])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday</td>
<td>12</td>
<td>12</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

• \textit{Linear operator} on page 189
• \textit{PointToPoint operator} on page 190
**Last**

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

**Function Group**
Aggregate

**Syntax**
```
input_type 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>measure</td>
<td>Any dimension or measure</td>
<td>Dimension or measure</td>
</tr>
</tbody>
</table>

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

**Max**

**Description**
Returns the largest value in a dimension or measure
Function Group
Aggregate

Syntax
input_type Max(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
You can use extended syntax context operators with Max.

Examples
If the Sales revenue measure has the values 3000, 60034 and 901234, Max([Sales Revenue]) returns 901234.

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

Median

Description
Returns the median (middle value) of a measure

Function Group
Aggregate

Syntax
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, Median takes the average of the middle two values.

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

Min

Description
Returns the smallest value in a dimension or measure

Function Group
Aggregate

Syntax
any_type Min(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
You can use extended syntax context operators with Min.
Examples

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

If the City dimension has the values Aberdeen and London, Min([City]) returns "Aberdeen".

Mode

Description

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

Function Group

Aggregate

Syntax

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</td>
<td>measure</td>
<td>Any dimension or measure</td>
<td>Measure</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].
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>
<tr>
<td>Sum:</td>
<td>10000</td>
<td>100</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 Percentage column has the formula `Percentage([Sales Revenue];Row)`
### 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 \([\text{measure}]\) has the set of numbers \((10;20;30;40;50)\), \(\text{Percentile}(\text{measure};0.3)\) returns 22, which is greater than or equal to 30% of the numbers in the set.

Product

Description

Multiplies the values of a measure

Function Group

Aggregate

Syntax

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

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.

RunningAverage

Description

Returns the running average of a measure

Function Group

Aggregate
Syntax

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

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 `RunningAverage`.
- You can set the calculation direction with the `Row` and `Col` operators.
- If you apply a sort on the measure referenced by `RunningAverage`, Web Intelligence applies the sort to the measure first, then calculates the running average.
- 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.
- `RunningAverage` does not automatically reset the average after a block break or new section.

Examples

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

<table>
<thead>
<tr>
<th>Country</th>
<th>Resort</th>
<th>Revenue</th>
<th>Running Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>Hawaiian Club</td>
<td>1,479,660</td>
<td>835,420</td>
</tr>
<tr>
<td>US</td>
<td>Bahamas Beach</td>
<td>971,444</td>
<td>1,225,552</td>
</tr>
</tbody>
</table>
RunningAverage([Revenue];([Country])) returns these results in the following table:

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

Related Topics
• IncludeEmpty operator on page 189
• Row/Col operators on page 191

**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 calc-</td>
<td>Keyword</td>
<td>No</td>
</tr>
<tr>
<td>reset_dims</td>
<td>Resets the calculation on the</td>
<td>Dimension list</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>specified dimensions</td>
<td></td>
<td></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`, Web Intelligence applies the sort to the measure first, then calculates the running count.
- 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* on page 189
- *Row/Col operators* on page 191
- *IncludeEmpty operator* on page 189
- *IncludeEmpty operator* on page 189

**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>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>
</tbody>
</table>
### RequiredTypeDescriptionParameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<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`, Web Intelligence applies the sort to the measure first, then calculates the running maximum.
- 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* on page 189
- *Row/Col operators* on page 191
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>detail</td>
<td>measure</td>
<td>Any 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 RunningMin.
- You can set the calculation direction with the Row and Col operators.
- If you apply a sort on the measure referenced by RunningMin, Web Intelligence applies the sort to the measure first, then calculates the running minimum.
- 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>
<tr>
<td>US</td>
<td>Bahamas Beach</td>
<td>971,444</td>
<td>835,420</td>
</tr>
<tr>
<td>US</td>
<td>Hawaiian Club</td>
<td>1,479,660</td>
<td>835,420</td>
</tr>
</tbody>
</table>

Related Topics

• IncludeEmpty operator on page 189
• Row/Col operators on page 191

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`, Web Intelligence applies the sort to the measure first, then calculates the running product.
- 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>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>

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

<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>5784</td>
</tr>
</tbody>
</table>

**Related Topics**
- *IncludeEmpty operator* on page 189
- *Row/Col operators* on page 191

### RunningSum

**Description**

Returns the running sum of a measure

**Function Group**

Aggregate

**Syntax**

```plaintext
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>
<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 the *RunningSum*.  

---

Web Intelligence functions, operators and keywords

Using Functions, Formulas and Calculations in Web Intelligence
• 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, Web Intelligence applies the sort to the measure first, then calculates the running sum.
• 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** on page 189
• **Row/Col operators** on page 191
StdDev

Description
Returns the standard deviation of a measure

Function Group
Aggregate

Syntax
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 \((\text{number of numbers in the set - 1})\)
- finding the square root of the result

Examples
If \(\text{measure}\) has the set of values \((2, 4, 6, 8)\) \(\text{StdDev([measure])}\) returns 2.58.

Related Topics
- \(\text{Var}\) on page 76
**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) StdDevP([measure]) returns 2.24.
**Sum**

**Description**
Returns the sum of a measure

**Function Group**
Aggregate

**Syntax**
num Sum(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**
You can use extended syntax context operators with Sum.

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

**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 \( \frac{\text{number of numbers in the set} - 1}{\} \)

The variance is the square of the standard deviation.

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

Examples

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

Related Topics

- \( \text{StdDev} \) on page 74

VarP

Description

Returns the population variance of a measure

Function Group

Aggregate
Syntax

```plaintext
num VarP(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 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 \( \frac{\text{number of numbers in the set}}{\text{number of numbers}} \)

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

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

Examples

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

Related Topics

- `StdDevP` on page 75

Character functions

Asc

Description

Returns the ASCII value of a character
Function Group
Character

Syntax
int Asc(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)
### 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".

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

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

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".
**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.
```

**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.

HTMLEncode

Description
Applies HTML encoding rules to a string

Function Group
Character

Syntax
string HTMLEncode(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>An HTML string</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Examples**


**InitCap**

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

**Function Group**
Character

**Syntax**

```
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".
Left

Description
Returns the leftmost characters of a string

Function Group
Character

Syntax
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".

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").

LeftTrim

Description

Trims the leading spaces from a string

Function Group

Character
Syntax

```
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".
```

Length

Description

Returns the number of characters in a string

Function Group

Character

Syntax

```
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".
```
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".

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".

Match("New York";"P*") returns False.

### Pos

**Description**

Returns the starting position of a text pattern in a string

**Function Group**

Character

**Syntax**

```plaintext
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.
- `Pos("New York"; "York")` returns 5.

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".

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".

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

RightTrim

Description
Trims the trailing spaces from a string

Function Group
Character

Syntax
string RightTrim(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
RightTrim([Country]) returns "France" if [Country] is "France ".

Using Functions, Formulas and Calculations in Web Intelligence 93
## Substr

### Description

Returns part of a string

### Function Group

Character

### Syntax

\[
\text{string SubStr(string;start;length)}
\]

### 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".

## Trim

### Description

Trims the leading and trailing spaces from a string
Function Group
Character

Syntax
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".

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".

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

WordCap

Description
Capitalizes the first letter of all the words in a string

Function Group
Character

Syntax
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".

Date and Time functions

CurrentDate

Description
Returns the current date formatted according to the regional settings

Function Group
Date and Time
Syntax

date CurrentDate()

Examples

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

**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.

**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").

**DayNumberOfMonth**

Description

Returns the day number in a month

Function Group

Date and Time

Syntax

```plaintext
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.
DayNumberOfWeek

Description
Returns 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
Web Intelligence 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).
```

DayNumberOfYear

Description
Returns the day number in a year

Function Group
Date and Time
Syntax

\[ \text{int DayNumberOfYear(date)} \]

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.

DaysBetween

Description

Returns the number of days between two dates

Function Group

Date and Time

Syntax

\[ \text{int DaysBetween(first_date;last_date)} \]

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.
**LastDayOfMonth**

**Description**

Returns the date of the last day in a month

**Function Group**

Date and Time

**Syntax**

`date LastDayOfMonth(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**

`LastDayOfMonth([Sale Date])` returns 31 December 2005 if [Sale Date] is 11 December 2005.

**LastDayOfWeek**

**Description**

Returns the date of the last day in a week

**Function Group**

Date and Time

**Syntax**

`date LastDayOfWeek(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
Web Intelligence treats Monday as the first day of the week.

Examples
LastDayOfWeek([Sale Date]) returns 15 May 2005 (a Sunday) if [Sale Date] is 11 May 2005.

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.
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.

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.

Quarter

Description

Returns the quarter number in a date

Function Group

Date and Time

Syntax

```sql
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 quar-ter</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.
RelativeDate

**Description**
Returns a date relative to another date

**Function Group**
Date and Time

**Syntax**
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.
ToDateFormat

Description
Returns a character string formatted according to a date format

Function Group
Date and Time

Syntax
\texttt{date \textbf{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
\texttt{ToDate("15/12/2002";"dd/MM/yyyy")} returns 15/12/2002.

Week

Description
Returns the week number in the year

Function Group
Date and Time

Syntax
\texttt{int \textbf{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).

Year

Description

Returns the year in a date

Function Group

Date and Time

Syntax

```plaintext
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.
Data Provider functions

Connection

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

Function Group
Data Provider

Syntax
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.

Examples
Connection([SalesQuery]) might return "BO_DRV_CONNECT_MODE=0;BO_DSN=eFashion;ODBC_USER=;ODBC_PASSWORD=;" (the return value differs depending on the database connection).

DataProvider

Description
Returns the name of the data provider containing a report object
Function Group
Data Provider

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

DataProviderKeyDate

Description
Returns the keydate of a data provider

Function Group
Data Provider

Syntax
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.

**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".
DataProviderSQL

Description
Returns the SQL generated by a data provider

Function Group
Data Provider

Syntax
stringDataProviderSQL(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
DataProviderSQL([Query 1]) returns "SELECT country.country_name FROM country" if the data provider SQL is "SELECT country.country_name FROM country".

DataProviderType

Description
Returns the type of a data provider

Function Group
Data Provider
**Syntax**

```c
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.

---

**IsPromptAnswered**

**Description**

Determines whether a prompt has been answered

**Function Group**

Data Provider

**Syntax**

```c
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.

### LastExecutionDate

#### Description

Returns the date on which a data provider was last refreshed

#### Function Group

Data Provider

#### Syntax

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 theDataProvider 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 on page 109

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.

**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 on page 109

**NumberOfDataProviders**

Description

Returns the number of data providers in a report

Function Group

Data Provider

Syntax

```csharp
int NumberOfDataProviders()
```

Examples

NumberOfDataProviders() returns 2 if the report has two data providers.

**NumberOfRows**

Description

Returns the number of rows in a data provider

Function Group

Data Provider

Syntax

```csharp
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` on page 109

RefValueDate

Description

Returns the date of the reference data used for data tracking

Function Group

Data Provider

Syntax

```plaintext
date RefValueDate()
```

Examples

`RefValueDate()` returns 15 December 2008 if the reference date is 15 December 2008.
**RefValueUserResponse**

**Description**

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

**Function Group**

Data Provider

**Syntax**

`string RefValueUserResponse([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**

- 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 `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 `Index` operator is specified) separated by semi-colons.
Examples

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.

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.

UniverseName

Description

Returns a 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

- Web Intelligence automatically updates the name of the data provider in the formula. If in the above example 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 on page 109

UserResponse

Description

Returns the response to a prompt

Function Group

Data Provider

Syntax

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.

Document functions

**DocumentAuthor**

**Description**

Returns the InfoView logon of the document creator

**Function Group**

Document

**Syntax**

string DocumentAuthor()

**Examples**

DocumentAuthor() returns "gkn" if the document author's login is "gkn".

**DocumentCreationDate**

**Description**

Returns the date on which a document was created
Function Group
Document

Syntax
\[ \text{date DocumentCreationDate()} \]

Examples
\[ \text{DocumentCreationDate()} \text{ returns 15 December 2008 if the document was created on 15 December 2008.} \]

\section*{DocumentCreationTime}

Description
Returns the time when a document was created

Function Group
Document

Syntax
\[ \text{time DocumentCreationTime()} \]

Examples
\[ \text{DocumentCreationTime()} \text{ returns 11:15 if the document was created at 11:15.} \]

\section*{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.

DocumentName

Description

Returns the document name

Function Group

Document

Syntax

string DocumentName()

Examples

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

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.

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.

DrillFilters

Description

Returns the drill filters applied to a document or object in drill mode
Function Group

Document

Syntax

\texttt{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 \texttt{obj} or \texttt{separator} required</td>
</tr>
<tr>
<td>separator</td>
<td>The drill filter sepa-</td>
<td>String</td>
<td>Either \texttt{obj} or \texttt{separator} required</td>
</tr>
</tbody>
</table>

Notes

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

Examples

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

\texttt{DrillFilters()} returns "US - 1999" if the document has a filter restricting \texttt{[Country]} to "US" and \texttt{[Year]} to 1999.

\texttt{DrillFilters("/")} returns "US / 1999" if the document has filters restricting \texttt{[Country]} to "US" and \texttt{[Year]} to 1999.

\texttt{DrillFilters ([Quarter])} returns "Q3" if the document has a drill filter restricting \texttt{[Quarter]} to "Q3".
**LastPrintDate**

**Description**
Returns the date on which a document was last printed

**Function Group**
Document

**Syntax**
date LastPrintDate()

**Examples**
LastPrintDate() returns 12 December 2005 if the document was last printed on 12 December 2005.

**PromptSummary**

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

**Function Group**
Document

**Syntax**
string PromptSummary()

**Examples**
QuerySummary() returns information about all the prompts in a document.

Example output:

Enter Quantity Sold: 5000
Enter value(s) for State (optional): California, Texas, Utah
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
Or Sales Revenue Less Than 10000)

Query 2:
Source file: D:\Data\datacar.xls
Result objects: State, Year, Sales Revenue

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".

ReportFilterSummary

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

Document

**Syntax**

```plaintext
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.

**Output example:**

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))
Logical functions

Even

Description
Determines whether a number is even

Function Group
Logical

Syntax
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, Web Intelligence converts the return value to an integer (1=true; 0=false). You can format this number 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.
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, Web Intelligence converts the return value to an integer (1=true; 0=false). You can format this number 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 on page 167
IsError

Description
Determines whether an object returns an error

Function Group
Logical

Syntax
bool IsError(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
• IsError returns a boolean value that you can use in the If function.
• If you place IsError directly into a column, Web Intelligence converts the return value to an integer. You can format this number 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* on page 167
**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, Web Intelligence converts the return value to an integer. You can format this number 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** on page 167
IsNull

Description
Determines whether a value is null

Function Group
Logical

Syntax
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, Web Intelligence converts the return value to an integer. You can format this number 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 on page 167
**IsNumber**

**Description**
Determines whether a value is a number

**Function Group**
Logical

**Syntax**
```java
bool IsNumber(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**
- IsNumber returns a boolean value that you can use in the If function.
- If you place IsNumber directly into a column, Web Intelligence converts the return value to an integer. You can format this number 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* on page 167
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, Web Intelligence converts the return value to an integer. You can format this number 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 on page 167
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, Web Intelligence converts the return value to an integer. You can format this number 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 on page 167
**Odd**

**Description**
Determined 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, Web Intelligence converts the return value to an integer. You can format this number 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* on page 167

# Numeric functions

## 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.

## 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.

Cos

Description
Returns the cosine of an angle

Function Group
Numeric

Syntax
num Cos(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>

Exampleless

\[ \cos(180) \] returns -0.6.

**EuroConvertFrom**

**Description**

Converts a euro amount to another currency

**Function Group**

Numeric

**Syntax**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>euro_amount</code></td>
<td>The amount in euros</td>
<td>Number</td>
<td>Yes</td>
</tr>
<tr>
<td><code>curr_code</code></td>
<td>The ISO code of the target currency</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td><code>round_level</code></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>Currency</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>Portugese 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
• How Web Intelligence rounds and truncates numbers on page 204

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>Currency Code</th>
<th>Currency Name</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>Portugese escudo</td>
</tr>
<tr>
<td>FIM</td>
<td>Finnish mark</td>
</tr>
</tbody>
</table>

**Related Topics**

- *How Web Intelligence rounds and truncates numbers* on page 204
**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 1995.80. 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>Code</th>
<th>Currency</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>Portugese escudo</td>
</tr>
<tr>
<td>FIM</td>
<td>Finnish mark</td>
</tr>
</tbody>
</table>

**Related Topics**

- *How Web Intelligence rounds and truncates numbers* on page 204
EuroToRoundError

Description
Returns the rounding error in a conversion to euros

Function Group
Numeric

Syntax
num 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>noneuro_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>Code</th>
<th>Currency</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>Portugese escudo</td>
</tr>
<tr>
<td>FIM</td>
<td>Finnish mark</td>
</tr>
</tbody>
</table>

Related Topics

- *How Web Intelligence rounds and truncates numbers* on page 204

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.

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.

Floor

Description
Returns a number rounded down to the nearest integer

Function Group
Numeric

Syntax
\[
\text{int Floor(number)}
\]

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.
**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

Ln(10) returns 2.

**Log**

Description
Returns the logarithm of a number in a specified base

Function Group
Numeric

Syntax
num Log(number;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

Log(125;5) returns 3.

---

### Log10

**Description**

Returns the base 10 logarithm of a number

**Function Group**

Numeric

**Syntax**

num Log10(number)

**Input**

| input_number | A number |

**Examples**

Log10(100) returns 2.
**Mod**

**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**

Mod(10;4) returns 2.

Mod (10.2;4.2) returns 1.8.

**Power**

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

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

\[ \text{num Power(number;power)} \]

**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**

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

**Rank**

**Description**

Ranks a measure by dimensions

**Function Group**

Numeric

**Syntax**

\[ \text{int Rank(measure;[ranking dims];[Top|Bottom];[reset dims])} \]
### 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>
</tbody>
</table>
| Top|Bottom | Sets the ranking order:  
  - Top - descending  
  - Bottom - ascending | Keyword | No (Top is default) |
| reset_dims  | The dimensions that reset the ranking            | Dimension list | No      |

#### Notes
- If you do not specify ranking dimensions, Web Intelligence uses the default calculation context to calculate the ranking.
- 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** \( \text{Rank}([\text{Revenue}];([\text{Country}])) \):

<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** \( \text{Rank}([\text{Revenue}];([\text{Country}]);\text{Bottom}) \). The **Bottom** argument means that the measures are ranked in descending order.
<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 \( \text{Rank}([\text{Revenue}]; ([\text{Country}]; [\text{Resort}])) \):

<table>
<thead>
<tr>
<th>Country</th>
<th>Resort</th>
<th>Revenue</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>French Riviera</td>
<td>835,420</td>
<td>3</td>
</tr>
<tr>
<td>US</td>
<td>Bahamas Beach</td>
<td>971,444</td>
<td>2</td>
</tr>
<tr>
<td>US</td>
<td>Hawaiian Club</td>
<td>1,479,660</td>
<td>1</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* on page 187
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
• How Web Intelligence rounds and truncates numbers on page 204
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.

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.

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**

\[ \text{sqrt}(25) \text{ returns 5.} \]

**Tan**

**Description**
Returns the tangent of an angle

**Function Group**
Numeric

**Syntax**
\[ \text{num} \text{ Tan}(\text{angle}) \]

**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>

**Examples**

\[ \text{Tan}(90) \text{ returns -2.} \]
**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.

**Truncate**

**Description**
Truncates a number

**Function Group**
Numeric
Syntax

\text{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

\text{Truncate(3.423;2) returns 3.42.}

Related Topics
- \text{How Web Intelligence rounds and truncates numbers} on page 204

Misc functions

BlockName

Description

Returns the block name

Function Group

Misc

Syntax

\text{string BlockName()}
Examples
BlockName() returns "Block1" if it is placed in a block called "Block1".

ColumnNumber

Description
Returns the column number

Function Group
Misc

Syntax
int ColumnNumber()

Examples
ColumnNumber() returns 2 if the formula is placed in the second column of a table.

CurrentUser

Description
Returns the InfoView login of the current user

Function Group
Misc

Syntax
string CurrentUser()

Examples
CurrentUser() returns "gkn" if the current user's InfoView login is "gkn".
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)

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>

Output
The result of the calculation with the synchronized dimensions taken into account.

Notes
• ForceMerge returns #MULTIVALUE if applied to a smart measure because the grouping set necessary to calculate the smart measure does not exist.
• ForceMerge is the Web Intelligence 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.
**GetContentLocale**

**Description**
Returns the document locale

**Function Group**
Misc

**Syntax**
string GetContentLocale()

**Examples**
GetContentLocale() returns "fr_FR" if the content locale is "French (France)".

**GetLocale**

**Description**
Returns the current locale

**Function Group**
Misc

**Syntax**
string GetLocale()

**Examples**
GetLocale() returns "en_US" if the current locale is "en_US".
If...Then...Else

Description
Returns a value based on whether an expression is true or false

Function Group
Misc

Syntax
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 using ElseIf. The syntax is:
  If test_value Then true_value [Else false_value]ElseIf test_value Then true_value [Else false_value...]
- Web Intelligence also supports the syntax If(bool_value;true_value;false_value).
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 on page 168
• And operator on page 183
• Between operator on page 184
• Inlist operator on page 185
• Or operator on page 183
• Not operator on page 184

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:
  
  If(bool_value;true_value;If(bool_value;true_value;false_value|If...))

• Web Intelligence also supports the If...Then...Else syntax.

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 on page 167
**LineNumber**

**Description**
Returns the line number in a table

**Function Group**
Misc

**Syntax**
```
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.
```

**NameOf**

**Description**
Returns the name of an object

**Function Group**
Misc

**Syntax**
```
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

Web Intelligence uses `NameOf` in column and row headers in reports.

### Examples

`NameOf([Reservation Date])` returns "Reservation Date".

### NoFilter

**Description**

Ignores filters when calculating a value

**Function Group**

Misc

**Syntax**

`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.

**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.

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.

Previous

Description
Returns a previous value of an object

Function Group
Misc
Syntax

\texttt{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>measure</td>
<td>Self</td>
<td>The dimension or measure whose previous value the function returns, or the Self keyword</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 offset rows previous to the current row</td>
<td>Integer</td>
<td>No (default is 1)</td>
</tr>
<tr>
<td>NoNull</td>
<td>Tells the function to return the first non-null value starting from the offset</td>
<td>Keyword</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes

- The default value of \texttt{offset} is 1. \texttt{Previous([Revenue];1)} and \texttt{Previous([Revenue])} are functionally the same.
- When you include the \texttt{NoNull} argument, Web Intelligence returns the first non-null value of the object beginning from the cell \texttt{offset} rows before the current row and counting backwards.
- You can use extended syntax context operators with \texttt{Previous}.
- The \texttt{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.
- Web Intelligence applies \texttt{Previous} after applying all report, section and block filters.
- You cannot apply a filter on a formula that uses \texttt{Previous}.
- Web Intelligence applies \texttt{Previous} after applying all sorts.
• You cannot apply a sort on a formula that uses `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.
• Web Intelligence resets `Previous` 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></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Previous([Revenue]) returns the following values in the following crosstab:

<table>
<thead>
<tr>
<th>Region</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>6,000,000</td>
<td>5,000,000</td>
<td>5,000,000</td>
</tr>
<tr>
<td>UK</td>
<td>2,000,000</td>
<td>2,500,000</td>
<td>2,000,000</td>
<td>2,000,000</td>
</tr>
<tr>
<td>France</td>
<td>3,000,000</td>
<td>2,000,000</td>
<td>3,000,000</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>5,000,000</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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Country</th>
<th>Region</th>
<th>Revenue</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>UK</td>
<td></td>
<td>7,000,000</td>
<td>12,000,000</td>
</tr>
</tbody>
</table>
Previous([Revenue]);2;NotNull) 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></td>
<td></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></td>
<td>700</td>
</tr>
<tr>
<td>2008</td>
<td>Q3</td>
<td></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 on page 226
- Self operator on page 193

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.

RelativeValue

Description

Returns previous or subsequent values of an object

Function Group

Misc

Syntax

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:

```
RelativeValue(Revenue; ([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>

For detailed information on **RelativeValue**, see the "Comparing values using Web Intelligence functions" chapter in the *Using Functions, Formulas*.
ReportName

Description
Returns the name of a report

Function Group
Misc

Syntax
string ReportName()

Examples
ReportName() returns "Sales Report" if it is placed in a report called "Sales Report".

RowIndex

Description
Returns the number of a row

Function Group
Misc

Syntax
integerRowIndex()
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.

**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".

**Web Intelligence function and formula operators**

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

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.
Logical operators

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

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.

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.

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".

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.
- Because the Document Formatting Locale can affect the sort order of data, changing the locale can impact the result returned by the Between operator. (You set the Document Formatting Locale in the Web Intelligence Document Preferences tab in InfoView.)
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 on page 167
• Where operator on page 193

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".
Function-specific operators

Some Web Intelligence 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.

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 on page 51
• Distinct/All operators on page 188
• NoFilter on page 171
• All/Drill operators on page 186

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
**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**

\[
\text{Rank}([\text{Revenue}] ; ([\text{Country}]) ; \text{Top}) \text{ ranks countries by revenue from highest to lowest.}
\]

**Related Topics**

- [Rank](#) on page 155

---

**Break operator**

**Description**

The Break operator tells Percentage function to account for table breaks.

**Examples**

The formula \( \text{Percentage}([\text{Revenue}]) \) 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>
<tr>
<td>2005</td>
<td>Q2</td>
<td>20000</td>
<td>20%</td>
</tr>
<tr>
<td>2006</td>
<td>Q1</td>
<td>30000</td>
<td>30%</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>

**Related Topics**
- *Percentage* on page 60

### Distinct/All operators

The **Distinct/All operators** tell the **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**
- *Count* on page 51
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 on page 50
• Count on page 51
• RunningAverage on page 63
• RunningCount on page 65

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 on page 121
• RefValueUserResponse on page 119

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.

**Related Topics**
- *Interpolation* on page 54

### 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* on page 173

### NotOnBreak operator

**Description**

The NotOnBreak operator tells the Interpolation function to ignore section and block breaks.

**Related Topics**
- *Interpolation* on page 54

### 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* on page 54

**Row/Col operators**

**Description**

The **Row/Col** operators set the calculation direction of the following functions: Percentage, RunningAverage, RunningCount, RunningMax, RunningMin, RunningProduct, RunningSum.

**Notes**

With the **Row** operator, Web Intelligence calculates each value in the row as a percentage of the total value of all the rows in the embedding context. With the **Col** operator, Web Intelligence calculates each value in the column as a percentage of the total value of all the columns in the embedding context.

In a crosstab, Web Intelligence by default calculates the value in each cell as a percentage of the total value in the crosstab. With the **Row** operator, Web Intelligence calculates the values in the rows as percentages of the total value for the row. With the **Col** operator, Web Intelligence calculates the values in the columns as percentages of the total value in the column.

**Examples**

In a crosstab, \( \text{Percentage}([\text{Measure}]) \) 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>

With the ROW operator (or by default), Web Intelligence calculates the running aggregate by row. With the COL operator, Web Intelligence calculates the running aggregate by column.

In a crosstab, **RunningSum([Measure])** or **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, **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>
Self operator

Description
Refers the Previous function to the previous cell when it does not contain a report object.

Examples
5 + Previous(Self) returns the sequence 5, 10, 15, 20, 25, 30...
1 + 0.5 * Previous(Self) returns the sequence 1, 1.5, 1.75, 1.88...

Related Topics
• Previous on page 173

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* on page 183
- *Between operator* on page 184
- *Inlist operator* on page 185
- *Or operator* on page 183
- *Not operator* on page 184

**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.

**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>$8096123.60</td>
<td>$2660699.50</td>
</tr>
<tr>
<td>2002</td>
<td>$13232246.00</td>
<td>$4196120.00</td>
</tr>
<tr>
<td>2003</td>
<td>$15059142.60</td>
<td>$4006717.50</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 tells Web Intelligence to calculate the maximum sales revenue for each (Year,Quarter) combination, then output 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.

---

**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:

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

**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.

### Web Intelligence 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.
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.

### 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>
</tbody>
</table>
### 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.

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

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]).

**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>When placed in...</td>
<td>References this data...</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------------------------------</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.

<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>$2,560,700</td>
<td>36,387,512.4</td>
</tr>
<tr>
<td>2001</td>
<td>Q2</td>
<td>$2,279,003</td>
<td>36,387,512.4</td>
</tr>
<tr>
<td>2001</td>
<td>Q3</td>
<td>$1,867,841</td>
<td>36,387,512.4</td>
</tr>
<tr>
<td>2001</td>
<td>Q4</td>
<td>$1,788,580</td>
<td>36,387,512.4</td>
</tr>
<tr>
<td>2002</td>
<td>Q1</td>
<td>$3,326,172</td>
<td>36,387,512.4</td>
</tr>
<tr>
<td>2002</td>
<td>Q2</td>
<td>$2,940,651</td>
<td>36,387,512.4</td>
</tr>
<tr>
<td>2002</td>
<td>Q3</td>
<td>$2,879,303</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>$3,742,989</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>$3,356,041</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]`).

**The Section keyword**

The following table describes the data referenced by the Section keyword depending on where it is placed in a report.
### How Web Intelligence rounds and truncates numbers

Several Web Intelligence 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.

---

#### Example: The Section keyword

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

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Sales revenue</th>
<th>Section Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>$2,660,700</td>
<td>8,095,814</td>
</tr>
<tr>
<td>Q2</td>
<td>$2,278,693</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>$1,788,580</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]).

---

<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>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</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 |

**Related Topics**
- *Round* on page 158
- *Truncate* on page 162
- *EuroConvertTo* on page 144
- *EuroConvertFrom* on page 142
- *EuroFromRoundError* on page 146
- *EuroToRoundError* on page 148
How Web Intelligence rounds and truncates numbers
Troubleshooting Web Intelligence formulas
Formula error and information messages

In some cases a Web Intelligence 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.

#COMPUTATION

#COMPUTATION occurs when a slicing dimension specified in the Relative Value function is no longer available in the calculation context of the block where the function is placed.

#COMPUTATION is also related to the misuse of context operators in a formula. For more information, see the Using Functions, Formulas and Calculations in Web Intelligence guide

Related Topics
• RelativeValue on page 178

#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.)

#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.

#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.
#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.

---

#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

---

#OVERFLOW

#OVERFLOW occurs when a calculation returns a value that is too large for Web Intelligence to handle. This value, in exponential form, is 1.7E308 (1.7 followed by 307 zeros).
#PARTIALRESULT

#PARTIALRESULT occurs when Web Intelligence was unable to retrieve all rows associated with a report object.

If #PARTIALRESULT occurs often in your reports and you have the appropriate security rights, modify the Max Rows Retrieved query property to allow Web Intelligence to retrieve more data. If you do not have the right to modify the query, see your Business Objects administrator.

If your report contains smart measures it is more likely to display #PARTIALRESULT because smart measures require Web Intelligence to retrieve larger amounts of data than classic measures.

#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.

#RECURSIVE

#RECURSIVE occurs when Web Intelligence cannot make 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, Web Intelligence returns #RECURSIVE because the placing of this formula in an autofit cell creates a circular dependency. Web Intelligence must know the exact size of the report before it can return a value from the function, but the size of the cell (which affects the size of the report) is determined by the cell content.

#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.

#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.
#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.

#UNAVAILABLE

#UNAVAILABLE appears when Web Intelligence cannot calculate the value of a smart measure.

This situation occurs when Web Intelligence cannot 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, Web Intelligence does not apply the query filter.

#ERROR

#ERROR is the default error message that covers all errors not covered by other error messages.
Calculating values with smart measures
Smart measures defined

"Smart measures" are measures whose values are calculated by the database (relational or OLAP) on which a Web Intelligence universe is based, rather than by Web Intelligence itself. A measure is defined as a smart measure in the universe when its data is aggregated in a way not supported by Web Intelligence.

To return values for smart measure, Web Intelligence generates a query to calculate the measure in all the calculation contexts required in a report. These contexts can change as the report is edited. As a result, Web Intelligence modifies the query at each data refresh after the required contexts have changed.

Smart measures behave differently from classic measures, which support a basic set of aggregation functions (Max, Min, Count, Sum, Average) that Web Intelligence can calculate 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), Web Intelligence initially displays Country, Region and Revenue in a block. If you then remove Region from the block, Web Intelligence is still able to calculate the total revenue for each country by summing the revenues for all the regions in the country.

Calculation contexts are represented by "grouping sets" in the query that Web Intelligence generates.

Grouping sets and smart measures

A "grouping set" is a set of dimensions that generates a result for a measure. When Web Intelligence returns data for a smart measure, the generated SQL 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)

Web Intelligence retrieves grouping sets by using the `UNION` operator in the query. If the database does not support `UNION`, Web Intelligence itself performs the unions.

Web Intelligence updates the grouping sets according to the calculation contexts required by the report, which can change in response to changes in the report structure.

---

**How Web Intelligence manages grouping sets**

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

For example, if you build a query containing the [Country], [Region] and [City] dimensions and the [Revenue] smart measure, Web Intelligence includes the (Country, Region, City) grouping set in the generated SQL. This grouping set always appears in the SQL. Web Intelligence adds and removes other grouping sets in response to changes in the report.

If you remove the [City] dimension from the block, Web Intelligence needs the (Country, Region) grouping set in order to return the revenue values. This grouping set is not yet available in the query SQL, so Web Intelligence displays `#TOREFRESH` in the [Revenue] cells. When you refresh the data, Web Intelligence is able to replace `#TOREFRESH` with the revenue values.

If you then replace the [City] dimension in the block, the (Country, Region) grouping set is no longer needed. Web Intelligence removes it from the query SQL and discards its values the next time you refresh the data.
Each time you refresh the report data, Web Intelligence updates the query SQL to include or discard grouping sets according to the calculation contexts required by the report.

In certain situations, Web Intelligence cannot display the value of a smart measure. In this case Web Intelligence displays #UNAVAILABLE in the measure cells.

**Smart measures and the scope of analysis**

When you build a query with a scope of analysis, Web Intelligence generates an initial grouping set that contains the result objects, but not the scope objects. Web intelligence does not generate all the possible grouping sets from the combination of the result objects plus 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, Web Intelligence retrieves the (Country) grouping set and displays [Country] and [Revenue] in a block.

**Smart measures and SQL**

**Grouping sets and the UNION operator**

Some databases support grouping sets explicitly with the GROUPING SETS operator. Web Intelligence 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 Web Intelligence universes support the \texttt{Sum} function.

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 \texttt{UNION} operator in the SQL.

If you remove the [City] dimension from the table, Web Intelligence needs the (Country, Region) grouping set to display the revenue (which appears 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 )
  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
```
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, Web Intelligence needs the (Country) grouping set. The generated SQL now includes three grouping sets as follows:

```
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 )
  AND ( region.region_id=city.region_id )
GROUP BY
  country.country_name,
  region.region_name
UNION
SELECT
  1 AS GID,
  country.country_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 )
  AND ( region.region_id=city.region_id )
GROUP BY
  country.country_name,
  region.region_name,
  city.city_name

Smart measures and formulas

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, Web Intelligence cannot display values for the smart measure. Web Intelligence cannot deduce the grouping set from a formula in this situation.

For example, a report contains a variable, Semester, with the formula
If [Quarter] = "Q1" or [Quarter] = "Q2" Then "H1" 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>
Smart measures in formulas

Web Intelligence can return a value for a smart measure when the smart measure is included in a formula, 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>
<tr>
<td>US</td>
<td>South</td>
<td>15000</td>
</tr>
<tr>
<td>US</td>
<td>East</td>
<td>14000</td>
</tr>
<tr>
<td>US</td>
<td>West</td>
<td>12000</td>
</tr>
</tbody>
</table>

If you include an additional column in the table with the formula

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

Web Intelligence initially returns #TOREFRESH because the formula requires the grouping set (Country). (The formula excludes regions from the calculation.) When you refresh the data, Web Intelligence adds the (Country) grouping set to the query and displays the measure values.

Smart measures and filters

Smart measures and filters on dimensions

If a filter is applied 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, Web Intelligence cannot return a value for the smart measure and displays #UNAVAILABLE.

This situation occurs because Web Intelligence cannot calculate the effect of the filter on the measure values. The only way to know its effect is to apply
the filter to the query. This carries the risk of impacting other reports based on the same query. As a result, Web intelligence does not apply the filter at the query level.

**Example: A smart measure and a filter on a dimension**

A query contains the [Country] and [Region] dimensions and the [Revenue] smart measure. [Country] and [Revenue] are displayed in a block. If you apply a report filter restricting the values of [Region] to "South East" or "South West", Web Intelligence displays #UNAVAILABLE in the [Revenue] cells.

---

**Smart measures and drill filters**

In general, Web Intelligence cannot 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] and Web Intelligence displays [Region], [Revenue] in the block and moves the filter on [Country] to the drill toolbar.

To do this, Web Intelligence adds the (Country, Region) grouping set to the query and retrieves all its data, then filters this data to display only those regions contained in the drilled country. Web Intelligence does not need to add a filter at the query level to filter regions based on their country.
Calculating values with smart measures
Smart measures and filters

Using Functions, Formulas and Calculations in Web Intelligence
Comparing values using Web Intelligence functions
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 on page 173
• RelativeValue on page 178
• Comparing values using the RelativeValue function on page 226

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 \([\text{Revenue}]\);
- The slicing dimension is \([\text{Year}]\);
- 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 tells Web Intelligence to return the revenue generated by the same sales person in the same quarter in the previous year.

Expressed as a calculation in words, the formula tells Web Intelligence to return the value of \([\text{Revenue}]\) (the expression) in the row where the value of \([\text{Year}]\) (the slicing dimension) is the previous value from the list of values of the \([\text{Year}]\) object, and where the values of \([\text{Quarter}]\) and \([\text{Sales Person}]\) (the sub-axis dimensions) are the same as in the current row.
Slicing dimensions and the RelativeValue function

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:

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>
</tbody>
</table>
Expressed as a business question, the formula tells Web Intelligence to return the revenue generated by the same sales person in the previous quarter.

Expressed as a calculation in words, the formula tells Web Intelligence to return 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.

To find the comparative value of revenue, Web Intelligence uses the list of values of the slicing dimensions:

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Sales Person</th>
<th>Revenue</th>
<th>RelativeValue</th>
</tr>
</thead>
<tbody>
<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>

The sort order of the slicing dimensions determines the output of the function. The * in the tables show the sort order.

Related Topics
• RelativeValue on page 178
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>RelativeValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td></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>
<tr>
<td>Q1</td>
<td>Jones</td>
<td>4000</td>
<td></td>
</tr>
</tbody>
</table>
To find the comparative value of revenue, Web Intelligence uses the list of values of the slicing dimensions:

<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.
Order of slicing dimensions

Because the sort order of the list of values of the slicing dimensions determines the output of `RelativeValue`, the order in which the slicing dimensions are specified impacts the output of the function.

**Example: Order of slicing dimensions**

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>
<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 tells Web Intelligence to display 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>
<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 function call is changed to:

RelativeValue([Revenue];([Quarter];[Year]);-1)

The sort order of the slicing dimensions becomes:

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2007</td>
</tr>
<tr>
<td>Q1</td>
<td>2008</td>
</tr>
<tr>
<td>Q2</td>
<td>2007</td>
</tr>
<tr>
<td>Q2</td>
<td>2008</td>
</tr>
<tr>
<td>Q3</td>
<td>2007</td>
</tr>
<tr>
<td>Q3</td>
<td>2008</td>
</tr>
<tr>
<td>Q4</td>
<td>2007</td>
</tr>
<tr>
<td>Q4</td>
<td>2008</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>
<tr>
<td>2008</td>
<td>Q2</td>
<td>Smith</td>
<td>3000****</td>
<td>2000***</td>
</tr>
<tr>
<td>2008</td>
<td>Q3</td>
<td>Smith</td>
<td>2700******</td>
<td>1500*****</td>
</tr>
<tr>
<td>2008</td>
<td>Q4</td>
<td>Smith</td>
<td>6800******</td>
<td>3000******</td>
</tr>
</tbody>
</table>

Expressed as a business question, the formula now tells Web Intelligence to display 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.

**Related Topics**
- *RelativeValue* on page 178

**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:

<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>Q4</td>
<td>Smith</td>
<td>3000</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>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>Q4</td>
<td>Jones</td>
<td>1700</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Q3</td>
<td>Jones</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Q1</td>
<td>Smith</td>
<td>5000**</td>
<td>1500*</td>
</tr>
<tr>
<td>2008</td>
<td>Q2</td>
<td>Smith</td>
<td>3000***</td>
<td>5000**</td>
</tr>
<tr>
<td>2008</td>
<td>Q4</td>
<td>Smith</td>
<td>6800****</td>
<td>3000***</td>
</tr>
<tr>
<td>2008</td>
<td>Q3</td>
<td>Smith</td>
<td>2700</td>
<td>6800****</td>
</tr>
</tbody>
</table>

The sorted list of 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>Q4</td>
</tr>
<tr>
<td>2007</td>
<td>Q3</td>
</tr>
<tr>
<td>2008</td>
<td>Q1</td>
</tr>
</tbody>
</table>
Comparing values using Web Intelligence functions
Comparing values using the RelativeValue function

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>Q2</td>
<td>***</td>
</tr>
<tr>
<td>2008</td>
<td>Q4</td>
<td>****</td>
</tr>
<tr>
<td>2008</td>
<td>Q3</td>
<td></td>
</tr>
</tbody>
</table>

The * in the tables show the sort order.

Related Topics
• RelativeValue on page 178

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 on page 178
Online documentation library

Business Objects offers a full documentation set covering all products and their deployment. The online documentation library has the most up-to-date version of the Business Objects product documentation. You can browse the library contents, do full-text searches, read guides on line, and download PDF versions. The library is updated regularly with new content as it becomes available.

To access the online documentation library, visit http://help.sap.com/ and click Business Objects at the top of the page.

Additional developer resources

https://boc.sdn.sap.com/developer/library/

Online customer support

The Business Objects Customer Support web site contains information about Customer Support programs and services. It also has links to a wide range of technical information including knowledgebase articles, downloads, and support forums.

http://www.businessobjects.com/support/

Looking for the best deployment solution for your company?

Business Objects consultants can accompany you from the initial analysis stage to the delivery of your deployment project. Expertise is available in relational and multidimensional databases, in connectivities, database design tools, customized embedding technology, and more.

For more information, contact your local sales office, or contact us at:

http://www.businessobjects.com/services/consulting/

Looking for training options?

From traditional classroom learning to targeted e-learning seminars, we can offer a training package to suit your learning needs and preferred learning style. Find more information on the Business Objects Education web site:

http://www.businessobjects.com/services/training
Send us your feedback

Do you have a suggestion on how we can improve our documentation? Is there something you particularly like or have found useful? Drop us a line, and we will do our best to ensure that your suggestion is included in the next release of our documentation:

mailto:documentation@businessobjects.com

Note:
If your issue concerns a Business Objects product and not the documentation, please contact our Customer Support experts. For information about Customer Support visit: http://www.businessobjects.com/support/.

Business Objects product information

For information about the full range of Business Objects products, visit: http://www.businessobjects.com.
Index

#COMPUTATION error message 178, 208
#CONTEXT error message 208, 209
#DATASYNC error message 208, 209
#DIV/0 error message 209
#ERROR error message 142, 144, 146, 148, 162, 213
#INCOMPATIBLE error message 208, 209, 210
#MULTIVALUE error message 180, 210
#OVERFLOW error message 210
#PARTIALRESULT error message 211
  and smart measures 211
#RANK error message 211
#RECURSIVE error message 211
#SECURITY error message 212
#SYNTAX error message 212
#TOREFRESH error message 213, 217, 222
  and smart measures 213
#UNAVAILABLE error message 213, 222
  and smart measures 213

A

Abs function 140
All operator 171, 186, 188
And operator 20, 183
Asc function 78
ASCII values 78
  returning characters associated with 79
  average
    calculating a running average 63
Average function 50, 189
  using with extended syntax keywords 42, 199
Average standard calculation 10

B

calculating a base 10 logarithm 153
calculating a base n logarithm 152
Between operator 20, 183, 184
Block keyword 42, 199
BlockName function 163
blocks
  displaying the names of 163
Body keyword 43, 200
boolean expressions
  linking with And operator 183
  returning the opposite of 184
boolean values
  identifying 134
  testing 167, 168
Bottom operator 155, 187
Break keyword 41, 201
Break operator 187
breaks
  default calculation contexts in 33
  building custom calculations using formulas 11

C

calculating a base 10 logarithm 153
calculating a base n logarithm 152
calculating a cosine 141
calculating a factorial 150
calculating a maximum value 56
calculating a median 57
calculating a minimum value 58
calculating a mode 59
calculating a natural logarithm 152
Index

calculating a percentage 14, 60
calculating a percentile 62
calculating a population standard deviation 75
calculating a population variance 77
calculating a product 63
calculating a running average 63
calculating a running count 65
calculating a running maximum 67
calculating a running minimum 69
calculating a running product 70
calculating a running sum 72
calculating a square root 160
calculating a standard deviation 74
calculating a sum 76
calculating a tangent 161
calculating a variance 76
calculating an average 50
calculating an exponential function 149
calculating rounding errors 146, 148
calculating the absolute value of a number 140
calculating the sine of an angle 159
calculation contexts
  and smart measures 222
  changing with extended syntax 27
default 27
defined 24
input context 24
output context 24, 25
calculations
  custom 10
  standard 10
Ceil function 140
cells
  including functions in 14
  including text in 14
Char function 79
character strings
  applying URL encoding rules to 96
  calculating length of 87
  capitalizing first letter of 84
  capitalizing all first letters in 97
  capitalizing first letters 97
character strings (continued)
  converting to lowercase 88
  extracting sections from 94
  joining/concatenating 20, 80, 182
  matching to a pattern 88
  padding with other strings 85, 92
  removing leading spaces from 86, 94
  removing trailing spaces from 93, 94
  repeating 81
  replacing parts of 90
  returning leftmost characters of 85
  returning rightmost characters of 91
  transforming to uppercase 95
  turning to numbers 162
characters
  displaying from ASCII values 79
  returning ASCII values of 78
charts
  displaying the names of 163
Col operator 63, 67, 69, 70, 72, 191
ColumnNumber function 164
columns
  displaying the numbers of 164
  comparing values using Previous 226
  comparing values using RelativeValue 226, 228, 230, 232, 234, 236
concatenating character strings 20, 80, 182
Concatenation function 80
conditional operators 20, 182
Connection function 109
context operators 21
converting from euros 142
converting to euros 144
Cos function 141
cosine 141
Count function 51, 186, 188, 189
Count standard calculation 10
counting rows in tables 170
counting values 65
crosstabs
  and the RelativeValue function 236
default calculation contexts in 30
currencies
  converting between European currencies  142, 144
CurrentDate function 97
CurrentTime function 98
CurrentUser function 164
custom calculations 10
  using formulas to build 11

data
  refreshing 217
data providers 117
  displaying number of rows in 117
  displaying the universe name 120
  viewing SQL generated by 112
DataProvider function 109
DataProviderKeyDate function 110
DataProviderKeyDateCaption function 111
DataProviderSQL function 112
DataProviderType function 112
dates
  calculating relative dates 106
  formatting 82, 107
  identifying 132
DayName function 98
DayNumberOfMonth function 99
DayNumberOfWeek function 100
DayNumberOfYear function 100
DaysBetween function 101
default calculation contexts
  in breaks 33
  in crosstabs 30
  in horizontal tables 30
  in sections 32
  in vertical tables 29
  modifying with extended syntax 34
Default standard calculation 10
dimensions
  adding to the calculation context 37, 196
  and #DATASYNC error message 209
dimensions (continued)
  and #INCOMPATIBLE error message 210
  and grouping sets 221
  removing from the calculation context 37, 197
  slicing dimensions 226, 228
  specifying in calculation context 35, 195
displaying column numbers 164
displaying function syntax 14
displaying page numbers 173
displaying prompt responses 121
displaying the author of documents 122
displaying the creation date of documents 122
displaying the last date a document was saved 123
displaying the last time a document was saved 125, 127
displaying the names of charts 163
displaying the names of report objects 170
displaying the names of tables 163
displaying the number of pages in a report 172
displaying the row number 180
displaying universe names 120
Distinct operator 188
DocumentAuthor function 122
DocumentDate function 123
DocumentName function 124
DocumentPartiallyRefreshed function 124
documents
  displaying the author of 122
  displaying the creation date of 122
  displaying the last print time of 127
  displaying the last save date of 123
  displaying the last save time of 125
  displaying the name of 124
  viewing queries in 128
DocumentTime function 125
drill filters 186
  and smart measures 223
  displaying 186
  ignoring 186
Drill operator 171, 186
DrillFilter function 125

E

error messages
  #COMPUTATION 178, 208
  #CONTEXT 208, 209
  #DATASYNC 208, 209
  #DIV/0 209
  #ERROR 142, 144, 146, 148, 162, 213
  #INCOMPATIBLE 208, 209, 210
  #MULTIVALUE 180, 210
  #OVERFLOW 210
  #PARTIALRESULT 211
  #RANK 211
  #RECURSIVE 211
  #SECURITY 212
  #SYNTAX 212
  #TOREFRESH 213, 217, 222
  #UNAVAILABLE 213, 222
errors
  identifying 133
EuroConvertFrom function 142, 204
EuroConvertTo function 144, 204
EuroFromRoundError function 146, 204
European currencies
  converting between 142, 144
euros
  converting from 142
  converting to 144
EuroToRoundError function 146, 204
Even function 131
even numbers
  identifying 131, 139
Exp function 149
exponential function 149
extended syntax (continued)
  In operator 35, 194, 195
  modifying default calculation context with 34
  Report keyword 39, 44, 202
  Section keyword 40, 203
extended syntax keywords 38, 198
  making reports generic with 44
  using with Average function 42, 199
  using with Sum function 39, 40, 41, 43, 200, 201, 202, 203
F
Fact function 150
factorial 150
Fill function 81
filters
  and smart measures 223
  block 129
  displaying all 186
  displaying drill filters 186
  drill 186
  drill filters 223
  on dimensions 222
  report 129, 186
  section 129
filters on dimensions
  affect on smart measures of 222
First function 53
Floor function 151
ForAll operator 37, 44, 197
ForceMerge function 165
ForEach operator 37, 196
FormatDate function 82
FormatNumber function 82
formatting numbers 82
Formula Editor
  displaying function syntax in 14
  formula error messages 208
formulas
  and smart measures 221
formulas (continued)
   building custom calculations using 11
   error messages generated by 208
   simplifying with variables 12, 18
   smart measures in 222
   use of operators in 19, 181
free-standing cells
   and #MULTIVALUE error message 210
function categories 50
function syntax
   example of 14
   how Web Intelligence displays 14
functions
   Abs 140
   Asc 78
   Average 42, 50, 189, 199
   BlockName 163
   Cell 140
   Char 79
   ColumnNumber 164
   Concatenation 80
   Connection 109
   Cos 141
   Count 51, 186, 188, 189
   CurrentDate 97
   CurrentTime 98
   CurrentUser 164
   DataProvider 109
   DataProviderKeyDate 110
   DataProviderKeyDateCaption 111
   DataProviderSQL 112
   DataProviderType 112
   DayName 98
   DayNumberOfMonth 99
   DayNumberOfWeek 100
   DayNumberOfYear 100
   DaysBetween 101
   defined 13
   DocumentAuthor 122
   DocumentCreationDate 122
   DocumentCreationDate function 122
   DocumentCreationTime 123
functions (continued)
   DocumentCreationTime function 123
   DocumentDate 123
   DocumentName 124
   DocumentPartiallyRefreshed 124
   DocumentTime 125
   DrillFilters 125
   EuroConvertFrom 142, 204
   EuroConvertTo 144, 204
   EuroFromRoundError 146, 204
   EuroToRoundError 148, 204
   Even 131
   examples of 14
   Exp 149
   Fact 150
   Fill 81
   First 53
   Floor 151
   ForceMerge 165
   FormatDate 82
   FormatNumber 82
   function syntax 14
   GetContentLocale 166
   GetLocale 166
   HTMLEncode 83
   If 20, 168, 182
   including in cells 14
   InitCap 84
   Interpolation 54, 189, 190
   Interpolation function 190
   IsDate 132
   IsError 133
   IsLogical 134
   IsNull 135
   IsNumber 136
   IsPromptAnswered 113
   IsString 137
   IsTime 138
   Last 56
   LastDayOfMonth 102
   LastDayOfWeek 102
   LastExecutionDate 114
Index

functions (continued)
   LastExecutionDuration 115
   LastExecutionTime function 116
   LastPrintDate 127
   Left 85
   LeftPad 85
   LeftTrim 86
   Length 87
   LineNumber 170
   Ln 152
   Log 152
   Log10 153
   Lower 88
   Match 88
   Max 37, 56, 196
   Median 57
   Min 58
   mixing with text in cells 14
   Mod 154
   Mode 59
   Month 103
   MonthNumberOfYear 104
   MonthsBetween 104
   NameOf 170
   NoFilter 171, 186
   NumberOfDataProviders 117
   NumberOfPages 172, 211
   NumberOfRows 117
   Odd 139
   Page 173
   Percentage 14, 60, 187, 191
   Percentage function 191
   Percentile 62
   Pos 89
   Power 154
   Previous 173, 190, 193, 211, 226
   Product 63
   PromptSummary 127
   Quarter 105
   QuerySummary 128
   Rank 155, 187
   RefValue 177

functions (continued)
   RefValueDate 118
   RefValueUserResponse 119, 189
   RelativeDate 106
   RelativeValue 178, 208, 226, 230, 232, 234, 236
   Replace 90
   ReportFilter 129
   ReportFilterSummary 129
   ReportName 180
   Right 91
   RightPad 92
   RightTrim 93
   Round 158, 204
   RowIndex 180
   RunningAverage 63, 189, 191
   RunningAverage function 191
   RunningCount 65, 189, 191
   RunningCount function 191
   RunningMax 67, 191
   RunningMax function 191
   RunningMin 69, 191
   RunningMin function 69, 191
   RunningProduct 70, 191
   RunningProduct function 191
   RunningSum 72, 191
   RunningSum function 191
   Sign 159
   Sin 159
   Sqrt 160
   StdDev 74
   StdDevP 75
   Substr 94
   Sum 14, 37, 39, 40, 41, 43, 44, 76, 197, 200, 201, 202, 203
   Sum function 76
   Tan 161
   ToDate 107
   ToNumber 162
   Trim 94
   Truncate 162, 204
   UniqueNameOf 181
functions (continued)
  UniverseName 120
  Upper 95
  URLEncode 96
  UserResponse 14, 121, 189
  Var 76
  VarP 77
  Week 107
  WordCap 97
  Year 108

G
GetContentLocale function 166
GetLocale function 166
grouping sets 216
  and scope of analysis 218
  and the UNION operator 218
  defined 216
  example of management of 218
  management in Web Intelligence 217

H
horizontal tables
  default calculation contexts in 30
HTMLEncode function 83

I
identifying boolean values 134
identifying dates 132
identifying errors 133
identifying even numbers 131, 139
identifying null values 135
identifying numbers 136
identifying odd numbers 131, 139
identifying strings 137
identifying time values 138
If function 20, 168, 182
If...Then...Else condition 167

J
joining character strings 20, 80, 182

K
keywords
  Block 42, 199
  Body 43, 200
  Break 41, 201
  making reports generic with 44
  Report 39, 44, 202
  Section 40, 203

L
Last function 56
LastDayOfMonth function 102
LastDayOfWeek function 102
Index

LastExecutionDate function 114
LastExecutionDuration function 115
LastExecutionTime function 116
LastPrintDate function 127
leading spaces
   removing from character strings 86, 94
Left function 85
LeftPad function 85
LeftTrim function 86
Length function 87
Linear operator 54, 189
LineNumber function 170
Ln function 152
Log function 152
Log10 function 153
logarithms 152, 153
logical operators 20, 183
Lower function 88

M

Match function 88
mathematical operators 20, 182
Max function 56
   using with context operators 37, 196
Maximum standard calculation 10
maximum values
   calculating a running maximum 67
measure values
   interpolating 54, 190
measures
   and default calculation contexts 27
   returning previous values of 178
   returning subsequent values of 178
   smart measures defined 216
median 57
Median function 57
Min function 58
Minimum standard calculation 10
minimum values
   calculating a running minimum 69
Mod function 154
mode 59
Mode function 59
Month function 103
MonthNumberOfYear function 104
MonthsBetween function 104
multiplying numbers 63, 70

N

NameOf function 170
natural logarithms 152
NoFilter function 171, 186
NoNull operator 173, 190
Not operator 20, 183, 184
NotOnBreak operator 54, 190
null values
   identifying 135
NumberOfDataProviders function 117
NumberOfPages function 172
   and #RECURSIVE error message 211
NumberOfRows function 117
numbers
   calculating a modulus 154
determining if odd or even 131, 139
dividing 154
formatting 82
identifying 136
modulus 154
multiplying 63, 70
raising to a power 154
rounding 140, 158
rounding down 151
rounding down numbers 151
rounding numbers 158
truncating 162
truncating numbers 162
turning character strings to 162

O

objects
   displaying the name of 181
objects (continued)
  viewing filters on 129
Odd function 139
odd numbers
  identifying 131, 139
operators
  All 171, 186, 188
  And 20, 183
  Between 20, 183, 184
  Bottom 155, 187
  Break 187
  Col 63, 65, 67, 69, 70, 72, 191
conditional
  context 21
  defined 19, 181
Distinct 188
Drill 171, 186
ForAll 37, 44, 197
ForEach 37, 196
function-specific 21, 186
function-specific operators 21, 186
In 35, 195
IncludeEmpty 65, 189
Index 121, 189
Index operator 189
Inlist 20, 183
InList 185
Linear 54, 189
logical 20, 183
mathematical 20, 182
NoNull 173, 190
Not 20, 183, 184
NotOnBreak 54, 190
Or 20, 183
PointToPoint 54, 190
Row 63, 65, 67, 69, 70, 72, 191
Self 173, 193
Top 155, 187
Where 193
Or operator 20, 183
output context
  defined 25

P
Page function 173
pages
  displaying number of in reports 172
pages numbers
  displaying in reports 173
Percentage function 14, 60
Percentage standard calculation 10
percentile 62
Percentile function 62
PointToPoint operator 54, 190
population standard deviation 75
population variance 77
Pos function 89
Power function 154
Previous function 173, 193
  and #RANK error message 211
  comparing values using 226
Previous function 190
Product function 63
products
  calculating a running product 70
prompts
  displaying responses to 14, 119, 121
PromptSummary function 127

Q
Quarter function 105
queries
  viewing summary of 128
query properties
  Max Rows Retrieved 211
QuerySummary function 128

R
Rank function 155, 187
ranking
  and running aggregate functions 211
ranking data 155
Index

reference data
  returning date of 118
refreshing data 217
RefValue function 177
RefValueDate function 118
RefValueUserResponse function 119, 189
RelativeDate function 106
RelativeValue function 178
  and #COMPUTATION error message 208
  and crosstabs 236
  and sections 230
  and slicing dimensions 228, 230, 234
  comparing values using 226, 228, 230, 232, 234, 236
Replace function 90
report filters 129, 186
  ignoring 186
  viewing summary of 129
Report keyword 39, 44, 202
ReportFilter function 129
ReportFilterSummary function 129
ReportName function 180
reports
  displaying number of pages in 172
  displaying page numbers in 173
  displaying the names of objects in 170
  viewing names of 180
Right function 91
RightPad function 92
RightTrim function 93
Round function 158, 204
rounding errors 146, 148
rounding numbers 140
rounding values 204
Row operator 63, 67, 69, 70, 72, 191
RowIndex function 180
rows
  counting all 186
  counting distinct 186
  counting in tables 170
  displaying number of in data provider 117
  displaying the row number 180
running aggregate functions
  and #RANK error message 211
  and ranking 211
running count 65
RunningAverage function 63, 189
RunningCount function 65, 189
RunningMax function 67
RunningProduct function 70
RunningSum function 72

S
scope of analysis
  and grouping sets 218
  and smart measures 218
Section keyword 40, 203
sections
  and slicing dimensions 230
  and the RelativeValue function 230
  default calculation contexts in 32
Self operator 173, 193
Sign function 159
simplifying formulas with variables 12, 18
Sin function 159
sine 159
slicing dimensions 226
  and sections 230
  impact of sort order of 228, 232, 234
  order of 232
smart measures
  affect of filters on 222
  and #PARTIALRESULT error message 211
  and #TOREFRESH error message 213
  and #UNAVAILABLE error message 213
  and context operators 222
  and drill filters 223
  and formulas 221, 222
  and grouping sets 216
  and scope of analysis 218
  and standard calculations 10
  and variables 221
defined 216
smart measures *(continued)*
  impossibility of calculating 213
sort order
  and slicing dimensions 228, 232, 234
sorts
  and slicing dimensions 228, 232, 234
SQL 112
  and data refresh 217
  and grouping sets 216, 217
  GROUPING SETS operator 218
  UNION operator 216, 218
Sqrt function 160
square root 160
standard calculations 10
standard deviation 74
StdDev function 74
StdDevP function 75
strings
  identifying 137
Substr function 94
Sum function 14
  using with context operators 37, 197
  using with extended syntax keywords 39, 40, 41, 43, 44, 200, 201, 202, 203
Sum standard calculation 10

T

tables
  counting rows in 170
  displaying the names of 163
  horizontal 30
  vertical 29
Tan function 161
tangents 161
testing boolean values 167, 168
time values
  identifying 138
ToDate function 107
toNumber function 162
Top operator 155, 187

trailing spaces
  removing from character strings 93, 94
Trim function 94
Truncate function 162, 204
truncating values 204

U

UNION operator 218
UniqueNameOf function 181
UniverseName function 120
universes
  displaying the names of 120
Upper function 95
URLEncode function 96
URLs
  applying encoding rules to 96
UserResponse function 14, 121, 189
Using Functions, Formulas and Calculations guide
  about 8

V

values
  comparing using Previous 226
  comparing using RelativeValue 226, 228, 230, 232, 234, 236
  counting 51, 65
  rounding 204
  truncating 204
Var function 76
variables
  and smart measures 221
  simplifying formulas with 12, 18
variance 76
VarP function 77
vertical tables
  default calculation contexts in 29