Using Functions, Formulas and Calculations in Web Intelligence

BusinessObjects Enterprise XI 3.0

2008-02-08

Contents

Contents

About this guide

chapter 1

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

chapter 2

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:

Calculation	Description
Sum	Calculates the sum of the selected data.
Count	Counts all rows for a measure object or count distinct rows for a dimension or detail object.
Average	Calculates the average of the data.
Minimum	Displays the minimum value of the selected data.
Maximum	Display the maximum value of the selected data.

Calculation	Description
Percentage	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. **Note:** 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.
Default	Applies the default aggregation function to a standard measure, or the database aggregation function to a smart measure.

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. (For more information, see *What are calculation contexts?* on page 24.)

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 `[Sales Revenue]/[Number Sold]` gives this value by dividing the revenue by the number of items sold in order to give the revenue per item.

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 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:

```
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)
```

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

```
Sum ([Difference Squared])/[Number of Observations] - 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 `Sum` function totals all the values in a measure and outputs the result. The formula `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 184
- *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 Average([Revenue]) appears in a cell as `=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 prototypes

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).

This description of a function's inputs and outputs it known as its prototype. Here is the prototype of the Abs function:

```
number Abs (number input_number)
```

This prototype tells you that the Abs function takes a single number (`input_number`) as input and returns a number as output.

The Formula Editor displays the function prototype 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:

Quarterly Revenues for Illinois

Year	Quarter	Sales revenue
2001	Q1	$256,454
	Q2	$241,458
	Q3	$107,006
	Q4	$133,306
2001	**Total**	**$738,223.80**

Year	Quarter	Sales revenue
2002	Q1	$334,297
	Q2	$254,722
	Q3	$230,573
	Q4	$331,067
2002	**Total**	**$1,150,658.80**

Year	Quarter	Sales revenue
2003	Q1	$255,658
	Q2	$354,724
	Q3	$273,186
	Q4	$250,517
2003	**Total**	**$1,134,085.40**

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 `Percentage ([Sales Revenue])`.

Year	Quarter	Sales revenue	Percentage
2001	Q1	$2660700	0.07
2001	Q2	$2279003	0.06
2001	Q3	$1367841	0.04
2001	Q4	$1788580	0.05
2002	Q1	$3326172	0.09
2002	Q2	$2840651	0.08
2002	Q3	$2879303	0.08
2002	Q4	$4186120	0.12
2003	Q1	$3742989	0.1
2003	Q2	$4006718	0.11
2003	Q3	$3953395	0.11
2003	Q4	$3356041	0.09
		Sum:	**1**

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.

2001

Year	Quarter	Sales revenue	Percentage
2001	Q1	$2660700	0.33
2001	Q2	$2279003	0.28
2001	Q3	$1367841	0.17
2001	Q4	$1788580	0.22
		Sum:	**1**

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.

2001	0.22

Year	Quarter	Sales revenue
2001	Q1	$2660700
2001	Q2	$2279003
2001	Q3	$1367841
2001	Q4	$1788580

2002	0.36

Year	Quarter	Sales revenue
2002	Q1	$3326172
2002	Q2	$2840651
2002	Q3	$2879303
2002	Q4	$4186120

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 [Sales Revenue]/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:

```
[Sales revenue]/(Sum([Sales revenue] In Report))
```

and the Percentage of Year column has the formula:

```
[Sales revenue]/(Sum([Sales revenue] In Section))
```

2001

Year	Quarter	Sales revenue	Percentage of Total	Percentage of Year
2001	Q1	$2660700	0.07	0.33
2001	Q2	$2279003	0.06	0.28
2001	Q3	$1367841	0.04	0.17
2001	Q4	$1788580	0.05	0.22

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. (For more information, see *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:

```
Sum(([Quantity sold] - Average([Quantity sold] ForEach [Quar
ter]) 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:

Variable	Formula
Average Sold	Average([Quantity Sold] In ([Quarter])) In Report
Number of Observations	Count([Quantity Sold] In ([Quarter])) In Report
Difference Squared	Power(([Quantity sold] - [Average Sold]);2)
Variance	Sum([Difference Squared] In ([Quarter]))/([Number of Observations] - 1)

The final formula is now

```
Sum ([Difference Squared])/[Number of Observations] - 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.

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 `[Sales`

`Revenue] - [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. The following table describes them:

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to

You use conditional operators with the If function, as in:

```
If ([Revenue] >= 10000;'High';'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

Logical operators are used in expressions that return True or False. You use such expressions in the If function. The Web Intelligence logical operators are AND, OR, NOT, Between and InList. For example, the formula

```
If ([Resort] = 'Bahamas Beach' OR [Resort]='Hawaiian Club';
'US'; 'France')
```

returns "US" if the resort is "Bahamas Beach or "Hawiian Club", "France" otherwise.

The formula

```
[Resort] = 'Bahamas Beach' OR [Resort]='Hawaiian Club'
```

returns True or False, True if the Resort variable is equal to 'Bahamas Beach' or 'Hawaiian Club', False otherwise.

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.

Understanding calculation contexts

chapter 3

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 (see *The input context* on page 24)
- The output context (see *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:

Report part	Input context
Section header and block footers	Year
Rows in the block	Year, Customer

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 is 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:

Year	Quarter	Sales revenue
	Q1	$2660699.50
	Q2	$2279003.00
	Q3	$1367840.70
	Q4	$1788580.40
2001		
	Min:	**$1367840.70**

Year	Quarter	Sales revenue
	Q1	$3326172.20
	Q2	$2840650.80
	Q3	$2879303.00
	Q4	$4186120.00
2002		
	Min:	**$2840650.80**

Year	Quarter	Sales revenue
	Q1	$3742988.90
	Q2	$4006717.50
	Q3	$3953395.30
	Q4	$3356041.10
2003		
	Min:	**$3356041.10**

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:

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

That is, the output context appears after the parentheses of the function whose output context you are specifying. In this case, the output context 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:

Year	Quarter	Sales revenue	Min by Year
2001	Q1	$2660699.50	$1367840.70
2001	Q2	$2279003.00	$1367840.70
2001	Q3	$1367840.70	$1367840.70
2001	Q4	$1788580.40	$1367840.70
2002	Q1	$3326172.20	$2840650.80
2002	Q2	$2840650.80	$2840650.80
2002	Q3	$2879303.00	$2840650.80
2002	Q4	$4186120.00	$2840650.80
2003	Q1	$3742988.90	$3356041.10
2003	Q2	$4006717.50	$3356041.10
2003	Q3	$3953395.30	$3356041.10
2003	Q4	$3356041.10	$3356041.10

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.

2005	**Total:8000**

Customer	Revenue
Harris	1000
Jones	3000
Walsh	4000
Total:	**8000**

Report total: 8000

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

Measure	Value	Context
Report total	20000	Total of all revenue in the report
Section header total	8000	Year

Measure	Value	Context
Customer total	1000, 3000, 4000	Year;Customer
Block footer total	8000	Year

Related Topics

- *What are calculation contexts?* on page 24
- *Modifying the default calculation context with extended syntax* on page 34

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:

When the calculation is in the...	The input context is	The output context is
Header	The dimensions and measures used to generate the body of the block	All the data is aggregated then the calculation function returns a single value
Body of the block	The dimensions and measures used to generate the current row	The same as the input context
Footer	The dimensions and measures used to generate the body of the block	All the data is aggregated then the calculation function returns a single value

Example: Default contexts in a vertical table

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

Year	Quarter	Sales revenue	$36387203
2001	Q1	$2660700	$2660699.50
2001	Q2	$2278693	$2278693.40
2001	Q3	$1367841	$1367840.70
2001	Q4	$1788580	$1788580.40
2002	Q1	$3326172	$3326172.20
2002	Q2	$2840651	$2840650.80
2002	Q3	$2879303	$2879303.00
2002	Q4	$4186120	$4186120.00
2003	Q1	$3742989	$3742988.90
2003	Q2	$4006718	$4006717.50
2003	Q3	$3953395	$3953395.30
2003	Q4	$3356041	$3356041.10
	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:

The calculation is in the...	The input context is...	The output context is...
Header	The dimensions and measures used to generate the body of the block.	All the data is aggregated, then the calculation function returns a single value.
Body of the block	The dimensions and measures used to generate the body of the block.	The same as the input context.

The calculation is in the...	The input context is...	The output context is...
Footer	The dimensions and measures used to generate the body of the block.	All the data is aggregated, then the calculation function returns a single value.
VBody footer	The dimensions and measures used to generate the current column.	All the data is aggregated, then the calculation function returns a single value.
HBody Footer	The dimensions and measures used to generate the current row.	All the data is aggregated, then the calculation function returns a single value.
VFooter	Same as footer.	All the data is aggregated, then the calculation function returns a single value.
HFooter	Same as footer.	All the data is aggregated, then the calculation function returns a single value.

Example: Default contexts in a crosstab

The following report shows the default contexts in a crosstab:

		FY2000	FY2000	FY2000	FY2000	1,115,730
		Q1	Q2	Q3	Q4	1,115,730
France	259,170	61,895	76,555	70,080	50,640	259,170
US	856,560	196,831	189,886	234,574	235,269	856,560
Sum:	1,115,730	258,726	266,441	304,654	285,909	1,115,730

Default contexts in a section

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

The calculation is in the...	The input context is...	The output context is...
Body	The dimensions and measures in the report, filtered to restrict the data to the section data.	All the data is aggregated, then the calculation function returns a single value.

Example: Default contexts in a section

The following report shows the default contexts in a crosstab:

2001	8,096,123.6

Quarter	Sales revenue	Section
Q1	$2,660,700	8,096,123.6
Q2	$2,279,003	8,096,123.6
Q3	$1,367,841	8,096,123.6
Q4	$1,788,580	8,096,123.6
Sum:	**8,096,123.6**	

2002	13,232,246

Quarter	Sales revenue	Section
Q1	$3,326,172	13,232,246
Q2	$2,840,651	13,232,246
Q3	$2,879,303	13,232,246
Q4	$4,186,120	13,232,246
Sum:	**13,232,246**	

2003	15,059,142.8

Quarter	Sales revenue	Section
Q1	$3,742,989	15,059,142.8
Q2	$4,006,718	15,059,142.8
Q3	$3,953,395	15,059,142.8
Q4	$3,356,041	15,059,142.8
Sum:	**15,059,142.8**	

Default contexts in a break

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

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:

Year	Quarter	$8096123
	Q1	$2660700
	Q2	$2279003
	Q3	$1367841
	Q4	$1788580
2001		
	Sum:	$8096124

Year	Quarter	$13232246
	Q1	$3326172
	Q2	$2840651
	Q3	$2879303
	Q4	$4186120
2002		
	Sum:	$13232246

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.

Specifying input and output contexts in the same formula

Extended syntax context operators

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

Operator	Description
In	Specifies an explicit list of dimensions to use in the context.
ForEach	Adds dimensions to the default context
ForAll	Removes dimensions from the default context

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:

Year	Sales revenue	Max Quarterly Revenue
2001	$8096123.60	$2660699.50
2002	$13232246.00	$4186120.00
2003	$15059142.80	$4006717.50

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:

Year	Quarter	Sales revenue
	Q1	$2660700
	Q2	$2279003
	Q3	$1367841
	Q4	$1788580
2001		
	Max:	**2660699.5**

Year	Quarter	Sales revenue
	Q1	$3326172
	Q2	$2840651
	Q3	$2879303
	Q4	$4186120
2002		
	Max:	**4186120**

Year	Quarter	Sales revenue
	Q1	$3742989
	Q2	$4006718
	Q3	$3953395
	Q4	$3356041
2003		
	Max:	**4006717.5**

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

```
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:

Year	Sales revenue	Max Quarterly Revenue
2001	$8096123.60	$2660699.50
2002	$13232246.00	$4186120.00
2003	$15059142.80	$4006717.50

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:

Year	Quarter	Sales revenue	Yearly Total
2001	Q1	$2660700	$8096124
2001	Q2	$2279003	$8096124
2001	Q3	$1367841	$8096124
2001	Q4	$1788580	$8096124
2002	Q1	$3326172	$13232246
2002	Q2	$2840651	$13232246
2002	Q3	$2879303	$13232246
2002	Q4	$4186120	$13232246
2003	Q1	$3742989	$15059143
2003	Q2	$4006718	$15059143
2003	Q3	$3953395	$15059143
2003	Q4	$3356041	$15059143

To total revenues by year the input context needs to be (Year); by default it is (Year; Quarter). Therefore, you can remove Quarter from the input context by specifying ForAll ([Quarter]) in the formula, which looks like this:

```
Sum([Sales Revenue] ForAll ([Quarter]))
```

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

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

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

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:

When placed in...	References this data...
A block	All data in the report
A block break (header or footer)	All data in the report
A section (header, footer, or outside a block)	All data in the report
Outside any blocks or sections	All data in the report

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.

Year	Quarter	Sales revenue	Report Total
2001	Q1	$2,660,700	36,387,512.4
2001	Q2	$2,279,003	36,387,512.4
2001	Q3	$1,367,841	36,387,512.4
2001	Q4	$1,788,580	36,387,512.4
2002	Q1	$3,326,172	36,387,512.4
2002	Q2	$2,840,651	36,387,512.4
2002	Q3	$2,879,303	36,387,512.4
2002	Q4	$4,186,120	36,387,512.4
2003	Q1	$3,742,989	36,387,512.4
2003	Q2	$4,006,718	36,387,512.4
2003	Q3	$3,953,395	36,387,512.4
2003	Q4	$3,356,041	36,387,512.4

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

When placed in...	References this data...
A block	All data in the section
A block break (header or footer)	All data in the section
A section (header, footer, or outside a block)	All data in the section
Outside any blocks or sections	Not applicable

Example: The Section keyword

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

2001

Quarter	Sales revenue	Section Total
Q1	$2.660.700	8.095.814
Q2	$2.278.693	8.095.814
Q3	$1.367.841	8.095.814
Q4	$1.788.580	8.095.814

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:

When placed in...	References this data...
A block	Data in the part of a block delimited by a break
A block break (header or footer)	Data in the part of a block delimited by a break
A section (header, footer, or outside a block)	Not applicable
Outside any blocks or sections	Not applicable

Example: The Break keyword

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

Year	Quarter	Sales revenue	Break Total
2001	Q1	$2,660,700	8,096,123.6
	Q2	$2,279,003	8,096,123.6
	Q3	$1,367,841	8,096,123.6
	Q4	$1,788,580	8,096,123.6
2001			

The report has break on Year. The Break Total column has the formula:

```
Sum ([Sales Revenue]) In Break
```

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

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.

When placed in...	References this data...
A block	Data in the whole block, ignoring breaks, respecting filters
A block break (header or footer)	Data in the whole block, ignoring breaks, respecting filters
A section (header, footer, or outside a block)	Not applicable
Outside any blocks or sections	Not applicable

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.

2001

Quarter	Sales revenue	First Half Average	Yearly Average
Q1	$2,660,700	$2,469,851.25	$8,096,123.60
Q2	$2,279,003	$2,469,851.25	$8,096,123.60
Sum:	**4,939,702.5**		

2002

Quarter	Sales revenue	First Half Average	Yearly Average
Q1	$3,326,172	$3,083,411.50	$13,232,246.00
Q2	$2,840,651	$3,083,411.50	$13,232,246.00
Sum:	**6,166,823**		

2003

Quarter	Sales revenue	First Half Average	Yearly Average
Q1	$3,742,989	$3,874,853.20	$15,059,142.80
Q2	$4,006,718	$3,874,853.20	$15,059,142.80
Sum:	**7,749,706.4**		

The Yearly Average column has the formula

```
Average([Sales revenue] In Section)
```

and the First Half Average column has the formula

```
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:

When placed in...	References this data...
A block	Data in the block

When placed in...	References this data...
A block break (header or footer)	Data in the block
A section (header, footer, or outside a block)	Data in the section
Outside any blocks or sections	Data in the report

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.

Year	Quarter	Sales revenue	Body
2001	Q1	$2,660,700	2,660,699.5
	Q2	$2,279,003	2,279,003
	Q3	$1,367,841	1,367,840.7
	Q4	$1,788,580	1,788,580.4
2001		**8,096,123.6**	

The Body column has the formula

```
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:

Year	Quarter	Sales revenue	Percentage of Total
2001	Q1	$2660700	7.31
2001	Q2	$2279003	6.26
2001	Q3	$1367841	3.76
2001	Q4	$1788580	4.92
2002	Q1	$3326172	9.14
2002	Q2	$2840651	7.81
2002	Q3	$2879303	7.91
2002	Q4	$4186120	11.5
2003	Q1	$3742989	10.29
2003	Q2	$4006718	11.01
2003	Q3	$3953395	10.86
2003	Q4	$3356041	9.22
		Sum:	**100**

The formula for the Percentage of Total column is:

```
([Sales revenue]/(Sum([Sales revenue]) In Report)) * 100
```

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

```
([Sales revenue]/Sum([Sales revenue] ForAll ([Year];[Quar
ter]))) * 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:

Year	Quarter	Month	Sales revenue	Percentage of Total
2001	Q1	1	$1003541.20	26.13
2001	Q1	2	$630073.20	29.97
2001	Q1	3	$1027085.10	27.12
2001	Q2	4	$895259.80	28.1
2001	Q2	5	$865615.10	24.3
2001	Q2	6	$517818.50	21.77
2001	Q3	7	$525903.50	20.42
2001	Q3	8	$173756.40	11.11
2001	Q3	9	$668180.80	16.45
2001	Q4	10	$655206.40	18.04
2001	Q4	11	$484024.20	18.55
2001	Q4	12	$649349.80	21.01
2002	Q1	1	$1335401.90	34.77
2002	Q1	2	$609012.80	28.97
2002	Q1	3	$1381757.50	36.49
2002	Q2	4	$1068308.90	33.53
2002	Q2	5	$1081884.80	30.38
2002	Q2	6	$690457.10	29.03
2002	Q3	7	$801954.70	31.14
2002	Q3	8	$581093.50	37.15
2002	Q3	9	$1496254.80	36.84
2002	Q4	10	$1545871.80	42.57
2002	Q4	11	$1081915.30	41.47
2002	Q4	12	$1558332.90	50.43
2003	Q1	1	$1501366.70	39.09
2003	Q1	2	$863451.90	41.07
2003	Q1	3	$1378170.30	36.39
2003	Q2	4	$1222329.40	38.37
2003	Q2	5	$1614147.30	45.32
2003	Q2	6	$1170240.80	49.2
2003	Q3	7	$1247313.50	48.44
2003	Q3	8	$809365.40	51.74
2003	Q3	9	$1896716.40	46.7
2003	Q4	10	$1430300.10	39.39
2003	Q4	11	$1043098.80	39.98
2003	Q4	12	$882642.20	28.56
			Sum:	**1200**

Why is this? The problem lies in:

```
Sum ([Sales Revenue] ForAll ([Year];[Quarter))
```

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

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

Web Intelligence functions, operators and keywords

chapter 4

Web Intelligence functions

Web Intelligence divides functions into the following categories:

Category	Description
Aggregate	Aggregates data (for example by summing or averaging a set of values)
Character	Manipulates character strings
Date and Time	Returns date or time data
Document	Returns data about a document
Data Provider	Returns data about a document's data provider
Logical	Returns TRUE or FALSE
Numeric	Returns numeric data
Misc	Functions that do not fit into the above categories

Aggregate functions

Average

Description

Returns the average of a set of numeric values

Function Group

Aggregate

Syntax

```
number Average ([measure]; [INCLUDEMPTY])
```

Input

[measure]	Any measure

Output

The average of the set of numeric values

Example

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

Notes

- You can use extended syntax context operators with the Average function.
- 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.

Related Topics

- *IncludeEmpty operator* on page 192

Count

Description

Counts the number of occurrences of an item

Function Group

Aggregate

Syntax

```
integer Count([object]; [INCLUDEMPTY];[DISTINCT|ALL];)
```

Input

[object]	Any object in the report

Output

The number of occurrences of the item. Depending on the DISTINCT/ALL argument, this number represents the number of distinct occurrences (ignoring duplication) or the total number of occurrences (including duplication).

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.

Notes

- You can use extended syntax context operators with the Count() function.
- 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.

Related Topics

- *IncludeEmpty operator* on page 192
- *Distinct/All operator* on page 191

First

Description

Returns the first value in a set of values

Function Group

Aggregate

Syntax

```
expression_output_type First (expression)
```

Input

expression	Any expression

Output

The first value in the set

Example

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

Notes

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

Last

Description

Returns the last value in a set of values

Function Group

Aggregate

Syntax

```
expression_output_type Last(expression)
```

Input

expression	Any expression

Output

The last value in the set

Example

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

Notes

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

Max

Description

Returns the maximum value of a set of values

Function Group

Aggregate

Syntax

```
any_type Max([object])
```

Input

[object]	Any object in the report

Output

The highest value in the set of values

Example

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".

Note

You can use extended syntax context operators with the **Max()** function.

Median

Description

Returns the median of a set of numbers. The median is the middle number in the set.

Function Group

Aggregate

Syntax

```
number Median([measure])
```

Input

measure	Any measure

Output

The median of the set of numbers

Example

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

Notes

If the set of numbers has an even number of values, **Median()** takes the average of the middle two values

Min

Description

Returns the lowest value of a set of values

Function Group

Aggregate

Syntax

```
any_type Min([object])
```

Input

[object]	Any object in the report

Output

The lowest value in the set of values

Example

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"

Note

You can use extended syntax context operators with the Min() function.

Mode

Description

Returns the most frequently-occuring value in a set of values

Function Group

Aggregate

Syntax

```
expression_output_type Mode(expression)
```

Input

expression	Any expression

Examples

Mode([Revenue]) returns 200 if [Revenue] has the values 100, 200, 300, 200.

`Mode([Country])` returns the most frequently-occuring value of [Country].

Notes

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

Percentage

Description

Returns the ratio of a numeric value to another numeric value

Function Group

Aggregate

Syntax

```
number Percentage([measure];[BREAK];[ROW|COL])
```

Input

[measure]	Any measure in the report
BREAK	Account for table break (optional)
ROW| COL	The calculation direction (optional)

Output

The ratio of the measure value in the current calculation context to the measure value in the default embedding context.

Example

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

Year	Sales Revenue	Percentage
2001	1000	10
2002	5000	50
2003	4000	40

Sum:	**10000**	**100**

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)**

Year	Quarter	Sales Revenue	Percentage
2001	Q1	1000	10
	Q2	2000	20
	Q3	5000	50
	Q4	2000	20
2001	**Sum:**	10000	100

Year	Quarter	Sales Revenue	Percentage
2002	Q1	2000	20
	Q2	2000	20
	Q3	5000	50
	Q4	1000	10
2002	**Sum:**	10000	100

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).**

	Q1	Percentage	Q2	Percentage	Q3	Percentage	Q4	Percentage
2001	1000	10	2000	20	5000	50	2000	20
2002	2000	20	2000	20	5000	50	1000	10

Percentile

Description

Returns a percentile of a set of numbers

Function Group

Numeric

Syntax

```
number Percentile([measure]; number percentile)
```

Input

[measure]	Any measure
percentile	A percentage expressed as a decimal

Output

The percentile value

Example

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

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.

Product

Description

Returns the product of a set of numerical values

Function Group

Aggregate

Syntax

```
number Product([measure])
```

Input

[measure]	Any measure

Output

The product of the set of numeric values

Example

`Product([Measure])` returns 30 if [Measure] has the values 2, 3, 5

RunningAverage

Description

Returns the running average of a set of numbers

Function Group

Aggregate

Syntax

```
number RunningAverage([measure];[Row|Col];[IncludeEmpty];[(reset_dimensions)])
```

Input

[measure]	Any measure
Row|Col	The calculation direction (optional)
IncludeEmpty	Include empty values in the calculation (optional)
reset_dimensions	The list of dimensions used to reset the running average (optional)

Output

The running average of the set of numbers

Examples

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

Country	**Resort**	Revenue	Running Average
US	Hawaiian Club	1,479,660	835,420
US	Bahamas Beach	971,444	1,225,552
France	French Riviera	835,420	1,095,508

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

Country	Resort	Revenue	Running Average
US	Hawaiian Club	1,479,660	835,420
US	Bahamas Beach	971,444	1,225,552
France	French Riviera	835,420	835,420

Notes

- You can use extended syntax context operators with the **RunningAverage()** function.
- You can set the calculation direction with the Row and Col operators.
- If you apply a sort on the measure referenced by the **RunningAverage()** function, 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.

Related Topics

- *IncludeEmpty operator* on page 192
- *Row and Col operators* on page 194

RunningCount

Description

Returns the running count of a set of numbers

Function Group

Aggregate

Syntax

`number RunningCount(`*`measure`*`;[Row|Col];[IncludeEmpty];[(`*`reset_dimensions`*`)])`

Input

measure	Any measure
Row|Col	The calculation direction (optional)
IncludeEmpty	Include empty values in the calculation (optional)
reset_dimensions	The list of dimensions used to reset the running count (optional)

Output

The running count of the set of numbers

Examples

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

Country	Resort	Revenue	Running Count
US	Hawaiian Club	1,479,660	1
US	Bahamas Beach	971,444	2
France	French Riviera	835,420	3

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

Country	Resort	Revenue	Running Count
US	Hawaiian Club	1,479,660	1

Country	Resort	Revenue	Running Count
US	Bahamas Beach	971,444	2
France	French Riviera	835,420	1

Notes

- You can use extended syntax context operators with the `RunningCount()` function.
- You can set the calculation direction with the ROW and COL operators.
- If you apply a sort on the measure referenced by the `RunningCount()` function, 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.

Related Topics

- *IncludeEmpty operator* on page 192
- *Row and Col operators* on page 194
- *IncludeEmpty operator* on page 192
- *IncludeEmpty operator* on page 192

RunningMax

Description

Returns the running maximum of a set of numbers

Function Group

Aggregate

Syntax

```
any_type RunningMax([object];[Row|Col
];[(reset_dimensions)])
```

Input

[measure]	Any object
Row|Col	The calculation direction (optional)
reset_dimensions	The list of dimensions used to reset the running maximum (optional)

Output

The running maximum of the report object

Examples

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

Country	**Resort**	Revenue	Running Max
France	French Riviera	835,420	835,420
US	Bahamas Beach	971,444	971,444
US	Hawaiian Club	1,479,660	1,479,660

Notes

- You can use extended syntax context operators with the **RunningMax()** function.
- You can set the calculation direction with the ROW and COL operators.
- If you apply a sort on the measure referenced by the **RunningMax()** function, Web Intelligence applies the sort to the measure first, then calculates the running max.
- 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.

Related Topics

- *IncludeEmpty operator* on page 192
- *Row and Col operators* on page 194

RunningMin

Description

Returns the running minimum of a set of numbers

Function Group

Aggregate

Syntax

```
any_type RunningMin([object];[Row|Col
];[(reset_dimensions)])
```

Input

[object]	Any object
Row|Col	The calculation direction (optional)
reset_dimensions	The list of dimensions used to reset the running minimum (optional)

Output

The running minimum of the set of numbers

Examples

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

Country	**Resort**	Revenue	Running Max
France	French Riviera	835,420	835,420
US	Bahamas Beach	971,444	835,420
US	Hawaiian Club	1,479,660	835,420

Notes

- You can use extended syntax context operators with the **RunningMin()** function.
- You can set the calculation direction with the ROW and COL operators.
- If you apply a sort on the measure referenced by the **RunningMin()** function, Web Intelligence applies the sort to the measure first, then calculates the running max.
- 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.

Related Topics

- *IncludeEmpty operator* on page 192
- *Row and Col operators* on page 194

RunningProduct

Description

Returns the running product of a set of numbers

Function Group

Aggregate

Syntax

```
number RunningProduct([measure];[Row|Col];[(reset_dimensions)])
```

Input

[measure]	Any measure
Row|Col	The calculation direction (optional)
reset_dimensions	The list of dimensions used to reset the running product (optional)

Output

The running product of the set of numbers

Examples

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

Country of origin	**City**	Number of guests	Running Product
Japan	Kobe	6	6
Japan	Osaka	4	24
US	Chicago	241	5,784

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

Country of origin	**City**	Number of guests	Running Product
Japan	Kobe	6	6
Japan	Osaka	4	24
US	Chicago	241	5784

Notes

- You can use extended syntax context operators with the **RunningProduct()** function.
- You can set the calculation direction with the ROW and COL operators.
- If you apply a sort on the measure referenced by the **RunningProduct()** function, Web Intelligence applies the sort to the measure first, then calculates the running max.
- 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.

Related Topics

- *IncludeEmpty operator* on page 192
- *Row and Col operators* on page 194

RunningSum

Description

Returns the running sum of a set of numbers

Function Group

Aggregate

Syntax

```
number RunningSum([measure];[Row|Col
];[(reset_dimensions)])
```

Input

[measure]	Any measure
Row|Col	The calculation direction (optional)

reset_dimensions	The list of dimensions used to reset the running sum (optional)

Output

The running sum of the set of numbers

Example

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

Country	**Resort**	Revenue	Running Sum
France	French Riviera	835,420	835,420
US	Bahamas Beach	971,444	1,806,864
US	Hawaiian Club	1,479,660	3,286,524

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

Country	**Resort**	**Revenue**	Running Sum
France	French Riviera	835,420	835,420
US	Bahamas Beach	971,444	971,444
US	Hawaiian Club	1,479,660	2,451,104

Notes

- You can use extended syntax context operators with the **RunningSum()** function.
- 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.

Related Topics

- *IncludeEmpty operator* on page 192
- *Row and Col operators* on page 194

StdDev

Description

Returns the standard deviation of a set of numbers

Function Group

Aggregate

Syntax

`number StdDev(measure)`

Input

measure	Any measure or numeric dimension

Output

The standard deviation of the set of numbers

Example

If *measure* has the set of values (2, 4, 6, 8) `StdDev`(*measure*) returns 2.58

Notes

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

- finding the average of the set of numbers;
- subtracting the average from each number in the set and squaring the difference;
- summing all these squared differences;
- dividing this sum by (*number of numbers in the set* - 1);
- finding the square root of the result

- The standard deviation is the square root of the variance.
- You can use extended syntax context operators with the `StdDev()` function.

Related Topics

- *Var* on page 75

StdDevP

Description

Returns the population standard deviation of a set of numbers

Function Group

Aggregate

Syntax

```
number StdDevP([measure])
```

Input

[measure]	Any measure

Output

The population standard deviation of the set of numbers

Example

If **[measure]** has the set of values (2, 4, 6, 8) **StdDevP([Revenue])** returns 2.24

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 the **StdDevP()** function.

Sum

Description

Returns the total of a set of numbers

Function Group

Aggregate

Syntax

```
number Sum([measure])
```

Input

[measure]	Any measure

Output

The sum of the set of numbers

Example

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

Note

You can use extended syntax context operators with the **Sum()** function.

Var

Description

Returns the variation of a set of numbers

Function Group

Aggregate

Syntax

`number Var(`*`measure`*`)`

Input

measure	Any measure

Output

The variation of the set of numbers

Example

If *measure* has the set of values (2, 4, 6, 8) `Var`(*measure*) returns 6.67

Notes

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

- You can use extended syntax context operators with the `Var()` function.

Related Topics

- *StdDev* on page 72

VarP

Description

Returns the population variation of a set of numbers

Function Group

Aggregate

Syntax

`number VarP(`*measure*`)`

Input

measure	Any measure

Output

The population variation of the set of numbers

Example

`If` *measure* has the set of values (2, 4, 6, 8) `VarP`(*measure*) returns 5

Notes

- The population variation 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*).

- The population variation is the square of the population standard deviation .
- You can use extended syntax context operators with the `VarP()` function.

Related Topics

- *StdDevP* on page 73

Character functions

Asc

Description

Returns the ASCII value of a character

Function Group

Character

Syntax

```
integer Asc(string input_string)
```

Input

input_string	A single character

Remarks

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

Output

The ASCII value of the character

Example

`Asc("A")` returns 65

`Asc("ab")` returns 97

`Asc([Country])` returns 85 when the Country dimension is "US"

Char

Description

Returns the character associated with an ASCII value

Function Group

Character

Syntax

```
string Char(integer ascii_value)
```

Input

ascii_value	An ASCII value

Output

The character associated with the ASCII value

Example

`Char(123)` returns "{"

Concatenation

Description

Concatenates (joins) two character strings

Function Group

Character

Syntax

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

Input

first_string	The first string
second_string	The second string

Output

The concatenated string

Example

`Concatenation("First ";"Second")` returns "First Second"

Note

You can also use the '+' operator to concatenate strings. For example, `"First " + "Second"` returns "First Second".

Fill

Description

Builds a character string consisting of a string repeated a number of times

Function Group

Character

Syntax

```
string Fill(string repeating_string; integer
num_repeats)
```

Input

repeating_string	The repeating string
num_repeats	The number of times the string repeats

Output

The repeated string

Example

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

FormatDate

Description

Formats a date according to a supplied format

Function Group

Character

Syntax

```
string FormatDate(date date_to_format; string
date_format)
```

Input

date_to_format	Any date
date_format	The format to apply to the date

Output

The date (as a string) formatted according to the format specified in **date_format**

Example

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

Note

- 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 the FormatDate function.

FormatNumber

Description

Returns a number formatted according to the format specified

Function Group

Character

Syntax

```
string FormatNumber(number number_to_format; string
number_format)
```

Input

number_to_format	Any number
number_format	The format to apply to the number

Output

The number formatted according to the format string

Example

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

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 the FormatNumber function.

HTMLEncode

Description

Applies HTML encoding rules to a string

Function Group

Character

Syntax

```
string HTMLEncode(string html)
```

Input

html	An html string

Output

The encoded string

Example

`HTMLEncode("http://www.businessobjects.com")` returns "http%3A%2F%2Fwww%2Ebusinessobjects%2Ecom "

InitCap

Description

Capitalizes the first letter of a string.

Function Group

Character

Syntax

```
string InitCap(string input_string)
```

Input

input_string	The input string

Output

The string with the first letter capitalized

Example

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

Left

Description

Returns a string consisting of the first n leftmost characters of an input string

Function Group

Character

Syntax

```
string Left(string input_string; integer
number_of_characters)
```

Input

input_string	The input string
number_of_characters	The number of characters to take from the left

Output

A string consisting of the first **number_of_characters** leftmost characters of the input string

Example

`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(string new_string; integer
output_length; string orig_string)
```

Input

new_string	The string to be added to the left of the original string

output_length	The length of the output string
original_string	The original string

Output

A string consisting of both strings concatenated

Examples

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

Notes

- If **output_length** is less than the lengths of **new_string** and **original_string**, **new_string** is truncated
- If **output_length** is greater than the lengths of **new_string** and **original_string**, **new_string** is repeated until the length is reached

LeftTrim

Description

Removes the leading (left-side) blanks from a string

Function Group

Character

Syntax

```
string LeftTrim(string input_string)
```

Input

input_string	The input string

Output

The input string stripped of left-side blanks

Example

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

Length

Description

Returns the length of a character string

Function Group

Character

Syntax

```
integer Length (string input_string)
```

Input

input_string	The input string

Output

The length of the string

Example

`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_string)
```

Input

input_string	The input string

Output

The input string in lower case

Example

`Lower("New York")` returns "new york"

Match

Description

Determines whether a string matches a pattern

Function Group

Character

Syntax

```
boolean Match(string input_string; string
pattern)
```

Input

input_string	The input string
pattern	The pattern to match

Output

True if the string matches the pattern; false otherwise

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

Note

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

Pos

Description

Returns the starting position of a text pattern in a string

Function Group

Character

Syntax

```
integer Pos(string input_string; string
pattern)
```

Input

input_string	The input string
pattern	The pattern to search for

Output

The starting position of the pattern in the string

Examples

`Pos("New York";"Ne")` returns 1

`Pos("New York, New York";"Ne")` returns 1

`Pos("New York"; "York")` returns 5

Note

- If the pattern occurs more than once, Match() returns the position of the first occurrence

Replace

Description

Replaces part of a string with another string

Function Group

Character

Syntax

```
string Replace(string input_string; string
string_to_replace; string replace_with)
```

Input

input_string	The input string
string_to_replace	The string within **input_string** to be replaced
replace_with	The string to replace **string_to_replace** with.

Output

The string with the part replaced

Example

```
Replace ("New YORK";"ORK";"ork") returns "New
York"
```

Right

Description

Returns string consisting of the first n right-most characters of an input string

Function Group

Character

Syntax

```
string Right(string input_string; integer
number_of_characters)
```

Input

input_string	The input string
number_of_characters	The number of characters to return from the right of the input string

Output

A string consisting of the first **number_of_characters** right-most characters of the input string

Example

`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(string new_string; integer
output_length; string orig_string)
```

Input

new_string	The string to be added to the right of the original string
output_length	The length of the output string
original_string	The original string

Output

A string consisting of both strings concatenated

Examples

`RightPad("New ";8;"York")` returns "New York"

`RightPad("New "; 6;"York")` returns "New Yo"

`RightPad("New "; 12;"York")` returns "New YorkYork"

Notes

- If **output_length** is less than the lengths of **new_string** and **original_string**, **new_string** is truncated

- If **output_length** is greater than the lengths of **new_string** and **original_string**, **new_string** is repeated until the length is reached

RightTrim

Description

Removes the trailing (right-side) blanks from a string

Function Group

Character

Syntax

```
string RightTrim(string input_string)
```

Input

input_string	The input string

Output

The input string stripped of trailing blanks

Example

`RightTrim([Country])` returns "France" if [Country] is "France "

Substr

Description

Extracts a string from a character string

Function Group

Character

Syntax

```
string SubStr (string input_string; integer start;
integer length)
```

Input

input_string	The input string
start	The position of the first character of the extracted string in the input string
length	The length of the extracted string

Output

The extracted string

Examples

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

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

Trim

Description

Removes the leading and trailing blanks from a string

Function Group

Character

Syntax

```
string Trim (string input_string)
```

Input

input_string	The input string

Output

The trimmed string

Example

`Trim (" Great Britain ")` returns "Great Britain"

Upper

Description

Converts a string to upper case

Function Group

Character

Syntax

```
string Upper(string input_string)
```

Input

input_string	The input string

Output

The input string in upper case

Example

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

UrlEncode

Description

Applied URL encoding rules to a string

Function Group

Character

Syntax

```
string UrlEncode(string html)
```

Input

html	An html string

Output

The encoded string

Example

`UrlEncode("http://www.businessobjects.com")` returns "http%3A%2F%2Fwww%2Ebusinessobjects%2Ecom"

WordCap

Description

Capitalizes the first letter of all the words in a string

Function Group

Character

Syntax

```
string WordCap(string input_string)
```

Input

input_string	The input string

Output

The input string with the first letter of every word capitalized

Example

`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()
```

Input

Nothing

Output

The current date

Example

`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()
```

Output

The current time

DayName

Description

Returns the name of the day in a date

Function Group

Date and Time

Syntax

```
string DayName(date input_date)
```

Input

input_date	Any date

Output

The name of the day in the date

Example

`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 number of the day in the month of a date

Function Group

Date and Time

Syntax

```
integer DayNumberOfMonth(date input_date)
```

Input

input_date	Any date

Output

The number of the day in the month

Example

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

DayNumberOfWeek

Description

Returns the number of the day in the week of a date

Function Group

Date and Time

Syntax

```
integer DayNumberOfWeek(date input_date)
```

Input

input_date	Any date

Output

The number of the day in the week

Example

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

Note

Web Intelligence always treats Monday as the first day of the week

DayNumberOfYear

Description

Returns the number of the day in the year in a date

Function Group

Date and Time

Syntax

```
integer DayNumberOfYear(date input_date)
```

Input

input_date	Any date

Output

The number of the day in the year

Example

`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

```
integer DaysBetween(date first_date; date
last_date)
```

Input

first_date	The first date in the range

last_date	The last date in the range

Output

The number of days between the dates

Example

`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 of the month in a date

Function Group

Date and Time

Syntax

```
date LastDayOfMonth(date input_date)
```

Input

input_date	Any date

Output

The date of the last day of the month in the input date

Example

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

LastDayOfWeek

Description

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

Function Group

Date and Time

Syntax

```
date LastDayOfWeek(date input_date)
```

Input

input_date	Any date

Output

The date of the last day of the week in the input date

Example

`LastDayOfWeek([Sale Date])` returns 14 May 2005 (a Saturday) if [Sale Date] is 11 May 2005.

Note

Web Intelligence always treats Sunday as the first day of the week.

Month

Description

Returns the name of the month in a date

Function Group

Date and Time

Syntax

```
string Month(date input_date)
```

Input

input_date	Any date

Output

The name of the month in the date

Example

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

MonthNumberOfYear

Description

Returns the number of the month in a date

Function Group

Date and Time

Syntax

```
integer MonthNumberOfYear(date input_date)
```

Input

input_date	Any date

Output

The number of the month in the date

Example

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

MonthsBetween

Description

Description

Returns the number of months between two dates

Function Group

Date and Time

Syntax

```
integer MonthsBetween(date first_date; date
last_date)
```

Input

first_date	The first date in the range
last_date	The last date in the range

Output

The number of months between the dates

Example

`MonthsBetween([Sale Date]; [Invoice Date])` returns 1 if [Sale Date] is 30 December 2005 and [Invoice Date] is 2 January 2006

Quarter

Description

Returns the number of the quarter in a date

Function Group

Date and Time

Syntax

```
integer Quarter(date input_date)
```

Input

input_date	Any date

Output

The number of the quarter in the date

Example

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

RelativeDate

Description

Returns a date that is the input date plus a number of days

Function Group

Date and Time

Syntax

```
date RelativeDate(date input_date; integer
num_days)
```

Input

first_date	The first date in the range
last_date	The last date in the range

Output

The date represented by input_date + num_days

Example

`RelativeDate[Reservation Date];2)` returns 17 December 2005 when [Reservation Date] is 15 December 2005

ToDate

Description

Returns a character string formatted according to a date format

Function Group

Date and Time

Syntax

```
date ToDate(string input_string; string format)
```

Input

input_string	The input string
format	The format of the input date

Output

The date formatted according to the format supplied in **format**.

Example

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

ToNumber

Description

Returns a string as a number

Function Group

Numeric

Syntax

```
number ToNumber(string number_string)
```

Input

number_string	The string to be converted to a number

Output

The string as a number

Example

`ToNumber("45")` returns 45

Note

If **number_string** is not a number, **ToNumber()** returns #ERROR

Week

Description

Returns the number of the week in the year in a date

Function Group

Date and Time

Syntax

```
integer Week(date input_date)
```

Input

input_date	Any date

Output

The number of the week in the date

Example

`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 number of the year in a date

Function Group

Date and Time

Syntax

```
integer Year(date input_date)
```

Input

input_date	Any date

Output

The number of the year in the date

Example

`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(object data_provider)
```

Input

data_provider	The data provider

Output

The list of parameters

Examples

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

Note

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

DataProvider

Description

Returns the name of the data provider containing a variable

Function Group

Data Provider

Syntax

```
string DataProvider (variable [any_variable])
```

Input

any_variable	Any variable in a data provider

Output

The name of the data provider

Example

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

DataProviderKeyDate

Description

Returns the keydate of a data provider

Function Group

Data Provider

Syntax

```
date DataProviderKeyDate(object data_provider)
```

Input

data_provider	A data provider

Output

The keydate for the data provider

Example

`DataProviderKeyDate([Sales])` returns 3 August 2007 if the keydate for the Sales data provider is 3 August 2007

Notes

- You must enclose the name of the data provider in square brackets
- The returned keydate is formatted according to the document locale

DataProviderKeyDateCaption

Description

Returns the caption of the data provider keydate

Function Group

Data Provider

Syntax

```
string DataProviderKeyDateCaption(object data_provider)
```

Input

data_provider	A data provider

Output

The caption of the keydate

Example

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

Note

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

DataProviderSQL

Description

Returns the SQL generated by a data provider

Function Group

Data Provider

Syntax

```
string DataProviderSQL (object data_provider)
```

Input

data_provider	A data provider

Output

The SQL generated by the data provider

Example

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

Note

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

DataProviderType

Description

Returns the type of a data provider

Function Group

Data Provider

Syntax

```
string DataProviderType (object data_provider)
```

Input

data_provider	A data provider

Output

"Universe" if the data provider is a universe; "Personal data" if the data provider is a personal data provider

Example

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

Note

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

LastExecutionDate

Description

Returns the date on which a data provider was last refreshed

Function Group

Data Provider

Syntax

`date LastExecutionDate(object data_provider)`

Input

data_provider	A data provider (optional)

Output

The date on which the data provider was last refreshed

Example

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

Notes

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

Related Topics

- *DataProvider* on page 110

LastExecutionDuration

Description

Returns the time in seconds that a data provider took to return its data the last time it was run

Function Group

Data Provider

Syntax

```
number LastExecutionDuration(object data_provider)
```

Input

data_provider	A data provider

Output

The time in seconds that the data provider took to return its data the last time it was run

Example

`LastExecutionDuration([Sales])` returns 3 is the Sales data provider took 3 second to return its data the last time it was run

Note

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

LastExecutionTime

Description

Returns the time at which a data provider was last refreshed

Function Group

Data Provider

Syntax

`date LastExecutionTime(object data_provider)`

Input

data_provider	A data provider (optional)

Output

The time at which the data provider was last refreshed

Example

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

Notes

- If your report has one data provider only you can omit the *data_provider* 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

Related Topics

- *DataProvider* on page 110

NumberOfDataProviders

Description

Returns the number of data providers in a report

Function Group

Data Provider

Syntax

```
integer NumberOfDataProviders()
```

Output

The number of data providers in the report

Example

`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

```
integer NumberOfRows(object data_provider)
```

Input

data_provider	A data provider

Output

The number of rows in the data provider

Example

`NumberOfRows(Query 1)` returns 10 if the *Query 1* data provider has 10 rows

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.

Related Topics

- *DataProvider* on page 110

RefValueDate

Description

Returns the date of the reference data used for data tracking

Function Group

Data Provider

Syntax

```
date RefValueDate()
```

Input

None

Output

The date of the reference data

RefValueUserReponse

Description

Returns the data entered in response to a prompt at the time when the reference data was the current data

Function Group

Data Provider

Syntax

```
string RefValueUserResponse(object data_provider;string
prompt_text;Index)
```

Input

data_provider	The data provider (optional)
prompt_text	The text that appears in the prompt
Index	Tells the function to return the database primary keys of the prompt values selected rather than the values themselves

Output

The data entered in the prompt

Example

`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 .

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 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.

UniverseName

Description

Returns a the name of the Universe on which a data provider is based

Function Group

Data Provider

Syntax

`string UniverseName(object data_provider)`

Input

data_provider	The data provider

Output

The name of the universe

Example

`UniverseName(Query 1)` returns "eFashion" if the data provider is based on the eFashion universe

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.

Related Topics

- *DataProvider* on page 110

UserResponse

Description

Returns the data entered in response to a prompt

Function Group

Data Provider

Syntax

```
string UserResponse(object data_provider; string
prompt_text; Index)
```

Input

data_provider	The data provider (optional)
prompt_text	The text that appears in the prompt

Index	Tells the function to return the database primary keys of the prompt values selected rather than the values themselves

Output

The data entered in the prompt

Example

`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.

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.

Document functions

DocumentAuthor

Description

Returns the InfoView logon of the person who created the document

Function Group

Document

Syntax

```
string DocumentAuthor()
```

Input

None

Output

The document author's InfoView logon

Example

`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

```
date DocumentCreationDate()
```

Input

None

Output

The date on which the document was created.

DocumentCreationTime

Description

Returns the time on which a document was created.

Function Group

Document

Syntax

```
time DocumentCreationTime()
```

Input

None

Output

The time on which the document was created.

DocumentDate

Description

Returns the date on which a document was last saved.

Function Group

Document

Syntax

```
date DocumentDate()
```

Input

None

Output

The date on which the document was last saved

Example

`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()
```

Input

None

Output

The document name

Example

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

DocumentPartiallyRefreshed

Description

Determines whether a document is fully or partially refreshed

Function Group

Document

Syntax

```
boolean DocumentPartiallyRefreshed()
```

Input

None

Output

True if the document is partially refreshed; false if it is fully refreshed

Example

`DocumentPartiallyRefreshed()` returns True if Web Intelligence is still retrieving document data

Note

`DocumentPartiallyRefreshed()` returns a boolean value that you can use in the *If()* on page 173 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.

DocumentTime

Description

Returns the time a document was last saved

Function Group

Document

Syntax

```
time DocumentTime()
```

Input

None

Output

The time the document was last saved

Example

`DocumentTime()` returns 15:45 if the document was last saved at 15:45

Note

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

DrillFilters

Description

Returns drill filters applied to a document or object in drill mode

Function Group

Document

Syntax

```
string DrillFilters([object]; [string
separator])
```

Input

[object]	Any object
separator	The drill filter separator (optional)

Output

The drill filters applied to the variable or object

Examples

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

`DrillFilters()` returns "US - 1999" if the document has a filter restricting [Country] to "US" and [Year] to 1999

`DrillFilters("/")` returns "US / 1999" if the document has a filter restricting [Country] to "US" and [Year] to 1999

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

Notes

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

LastPrintDate

Description

Returns the date on which the document was last printed

Function Group

Document

Syntax

```
date LastPrintDate()
```

Input

None

Output

The date on which the document was last printed (or nothing if the document has never been printed)

Examples

`LastPrintDate()` returns 12 December 2005 if the document was last printed on 12 December 2005

PromptSummary

Description

Returns the prompt text and related user response for all prompts in a document

Function Group

Document

Syntax

```
string PromptSummary()
```

Input

None

Output

The prompt text and related user response for all the prompts in the document

Example

`PromptSummary()` example output:

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

Enter Customer (optional):
```

QuerySummary

Description

Returns information about the queries in a document

Function Group

Document

Syntax

```
string QuerySummary(object data_provider)
```

Input

data_provider	A data provider (optional)

[none]

Output

The query information

Example

`QuerySummary([Query 1])` example output:

```
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
```

Notes

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

ReportFilter

Description

Returns the report filters applied to an object or report

Function Group

Document

Syntax

```
string ReportFilter([object]|)
```

Input

[object]	Any report object

Output

The report filters applied to the object or report

Examples

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

ReportFilterSummary

Description

Returns information about the report filters in a document

Function Group

Document

Syntax

```
string ReportFilterSummary([string report_name])
```

Input

report_name	The name of the report (optional)

Output

The information about the report filters

Example

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

`ReportFilterSummary("Last Quarter Sales")` returns information about the report filters in the "Last Quarter Sales" report.

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))
```

ReportName

Description

Returns the name of a report

Function Group

Document

Syntax

```
string ReportName()
```

Input

None

Output

The name of the report in which the function is placed

Example

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

UserResponse

Description

Returns the data entered in response to a prompt

Function Group

Data Provider

Syntax

```
string UserResponse(object data_provider; string
prompt_text; Index)
```

Input

data_provider	The data provider (optional)
prompt_text	The text that appears in the prompt

Index	Tells the function to return the database primary keys of the prompt values selected rather than the values themselves

Output

The data entered in the prompt

Example

`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.

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.

Logical functions

Even

Description

Determines whether a number is even

Function Group

Logical

Syntax

```
boolean Even(number input_number)
```

Input

input_number	Any number

Output

True if the number is even; false otherwise

Examples

`Even(4)` returns true.

`Even(23.2)` returns false.

`Even(24.2)` returns true.

Notes

`Even()` returns a boolean value that you can use in the If function. If you place the function directly into a column, Web Intelligence converts the return value to an integer. You can format this number using a Boolean number format.

IsDate

Description

Determines whether a variable has the date data type

Function Group

Logical

Syntax

`boolean IsDate(object)`

Input

object	Any object

Output

True if the object is of data type date; false if not.

Examples

`IsDate(Reservation Date)` returns True if *Reservation Date* has the date data type

`If(IsDate(Reservation Date;"Date";"Not a date")` returns "Date" if *Reservation Date* has the date data type

Note

`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. You can format this number using a Boolean number format.

Related Topics

- *If* on page 173

IsError

Description

Determines whether a variable returns an error

Function Group

Logical

Syntax

`boolean IsError(object)`

Input

object	Any object

Output

True if the object returns an error; false otherwise

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(Sales Revenue);"Error";"No Error")` returns "Error" if the *Sales Revenue* variable returns an error

Note

`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.

Related Topics

- *If* on page 173

IsLogical

Description

Determines whether an object has a Boolean data type

Function Group

Logical

Syntax

`boolean IsLogical(object)`

Input

object	Any object

Output

True if the object has a boolean data type; false otherwise

Examples

`IsLogical(IsString)` returns true if the *IsString* variable has the formula **IsString(*Country*)**

`If (IsLogical(IsString;"Logical";"Not logical")` returns "Logical" if the *IsString* variable has the formula `IsString`(*Country*)

Note

`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.

Related Topics

- *If* on page 173

IsNull

Description

Determines whether a variable is null

Function Group

Logical

Syntax

`boolean IsNull(`*`object`*`)`

Input

object	Any object

Output

True if the object is null; false otherwise

Examples

`IsNull`(*Revenue*) returns false if the *Revenue* variable is not null.

`IsNull`(*Average Guests*) returns true if the *Average Guests* variable is null.

`If`(`IsNull`(*Sales Revenue*);"Null";"Not Null") returns "Null" if the *Sales Revenue* variable is null.

Note

`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.

Related Topics

- *If* on page 173

IsNumber

Description

Determines whether a variable is a number

Function Group

Logical

Syntax

`boolean IsNumber(`*`object`*`)`

Input

object	Any object

Output

True if the object is a number; false otherwise

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 (`*`Sales Revenue`*`);"Number";"Not a number")` returns "Number" if the *Sales Revenue* variable is numerical

Note

`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.

Related Topics

- *If* on page 173

IsPromptAnswered

Description

Returns whether a prompt has been answered or not

Function Group

Boolean

Syntax

```
boolean IsPromptAnswered (object data_provider; string prompt)
```

Input

data_provider	The data provider (optional)
prompt	The text associated with a prompt

Output

True if the prompt has been answered; false if not

Example

`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.

Note

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

IsString

Description

Determines whether a variable is a string

Function Group

Logical

Syntax

```
boolean IsString(object)
```

Input

object	Any object

Output

True if the object is a string; false otherwise

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`*`Country`*`);"String";"Not a string")` returns "String" if the *Country* variable is a string.

Note

`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.

Related Topics

- *If* on page 173

IsTime

Description

Determines whether a variable is a time variable

Function Group

Logical

Syntax

`boolean IsTime(`*`object`*`)`

Input

object	Any object

Output

True if the variable is a time variable; false otherwise

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(Reservation Time);"Time";"Not time")` returns "Time" if the *Reservation Time* variable is a time variable.

Note

`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.

Related Topics

- *If* on page 173

Odd

Description

Determines whether a number is odd

Function Group

Logical

Syntax

`boolean Odd(number input_number)`

Input

input_number	Any number

Output

True if the number is odd; false otherwise

Example

`Odd(5)` returns true.

`Odd(23.2)` returns true.

`Odd(24.2)` returns false.

`If(Odd(Sales Revenue);'Odd';'Even')` returns 'Odd' if *Sales Revenue* has an odd value.

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

Related Topics

- *If* on page 173

Numeric functions

Abs

Description

Returns the absolute value of a number (the number's value without the sign)

Function Group

Numeric

Syntax

```
number Abs (number input_number)
```

Input

input_number	A set of numeric values (for example, a measure)

Output

The absolute value of the number

Examples

`Abs(25)` returns 25

`Abs(-11)` returns 11

Ceil

Description

Returns a number rounded up to the next whole number

Function Group

Numeric

Syntax

```
number Ceil(number input_number)
```

Input

input_number	Any numeric variable

Output

The number rounded up to the next whole number

Examples

`Ceil(2.4)` returns 3

`Ceil(3.1)` returns 4

`Ceil(-3.1)` returns -3

Cos

Description

Returns the cosine of a number, where the number is an angle in radians

Function Group

Numeric

Syntax

```
number Cos(number angle)
```

Input

angle	An angle in radians

Output

The cosine of the angle

Examples

`Cos(180)` returns -0.6

EuroConvertFrom

Description

Converts an amount in euros to another currency

Function Group

Numeric

Syntax

```
number EuroConvertFrom(number amount; string code;
integer num_decimals)
```

Input

amount	The amount in euros
code	The ISO code of the target currency
num_decimals	The number of decimal places in the converted amount

Output

The amount in the target currency

Example

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

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

`EuroConvertFrom(1000.04 ;"DEM";2)` returns 1955.91

`EuroConvertFrom(1000.04 ;"DEM";1)` returns 1955.90

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. These currencies are:

BEF	Belgian franc
DEM	German mark
GRD	Greek drachma
ESP	Spanish peseta
FRF	French franc
IEP	Irish punt
ITL	Italian lira
LUF	Luxembourg franc
NLG	Dutch guilder
ATS	Austrian schilling
PTS	Portugese escudo
FIM	Finnish mark

EuroConvertTo

Description

Converts an amount in another currency to euros

Function Group

Numeric

Syntax

```
number EuroConvertFrom(number amount; string code;
integer num_decimals)
```

Input

amount	The amount in the original currency
code	The ISO code of the original currency
num_decimals	The number of decimal places in the converted amount

Output

The amount in Euros

Example

`EuroConvertTo(6559;"FRF";2)` returns 99.91

`EuroConvertTo(6559;"FRF";1)` returns 99.90

`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. These currencies are:

BEF	Belgian franc
DEM	German mark

GRD	Greek drachma
ESP	Spanish peseta
FRF	French franc
IEP	Irish punt
ITL	Italian lira
LUF	Luxembourg franc
NLG	Dutch guilder
ATS	Austrian schilling
PTS	Portugese escudo
FIM	Finnish mark

EuroFromRoundError

Description

Returns the rounding error in a non-euro-to-euro calculation

Function Group

Numeric

Syntax

```
number EuroFromRoundError(number amount; string
code; integer num_decimals)
```

Input

amount	The amount in euros

code	The ISO code of the target currency
num_decimals	The number of decimal places in the converted amount

Output

The rounding error in the calculation

Example

`EuroFromRoundErr(1000;"FRF";2)` returns 0

`EuroFromRoundErr(1000;"FRF";1)` returns 0.03

`EuroFromRoundErr(1000.04;"DEM";2)` returns 0

`EuroFromRoundErr(1000.04;"DEM";1)` returns -0.01

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. These currencies are:

BEF	Belgian franc
DEM	German mark
GRD	Greek drachma
ESP	Spanish peseta
FRF	French franc
IEP	Irish punt
ITL	Italian lira
LUF	Luxembourg franc

NLG	Dutch guilder
ATS	Austrian schilling
PTS	Portugese escudo
FIM	Finnish mark

EuroToRoundError

Description

Returns the rounding error in a euro-to-non-euro calculation

Function Group

Numeric

Syntax

```
number EuroToRoundError(number amount; string code;
integer num_decimals)
```

Input

amount	The amount in euros
code	The ISO code of the target currency
num_decimals	The number of decimal places in the converted amount

Output

The rounding error in the calculation

Examples

`EuroToRoundErr(6559;"FRF";2)` returns 0

`EuroToRoundErr(6559;"FRF";1)` returns -0.01

`EuroToRoundErr(1955;"DEM";2)` returns 0

`EuroToRoundErr(1955;"DEM";1)` returns 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. These currencies are:

BEF	Belgian franc
DEM	German mark
GRD	Greek drachma
ESP	Spanish peseta
FRF	French franc
IEP	Irish punt
ITL	Italian lira
LUF	Luxembourg franc
NLG	Dutch guilder
ATS	Austrian schilling
PTS	Portugese escudo
FIM	Finnish mark

Exp

Description

Returns e (2.718...) raised to a power

Function Group

Numeric

Syntax

```
number Exp(number power)
```

Input

power	The power to which you want to raise the numeric variable

Output

e (2.718...) raised to the power

Example

`Exp(2.2)` returns 9.03

Fact

Description

Returns the factorial of an integer

Function Group

Numeric

Syntax

```
integer Fact(integer input_number)
```

Input

input_number	Any integer

Output

The factorial of the number

Example

`Fact(4)` returns 24

Floor

Description

Returns a number rounded down to the nearest integer.

Function Group

Numeric

Syntax

```
integer Floor(number input_number)
```

Input

input_number	Any number

Output

The number rounded down to the nearest integer

Example

`Floor(24.4)` returns 24

Interpolation

Description

Calculates empty measure values by interpolation

Function Group

Numeric

Syntax

```
number Interpolation(measure;[interpolation_method];[NotOn
Break];[Row|Col]
)
```

Input

[measure]	Any measure
interpolation_method	The interpolation method (optional): • `PointToPoint` - point-to-point interpolation. This is the default interpolation method when you do not supply the argument. • `Linear` - linear regression with least squares interpolation
NotOnBreak	Prevents the function from resetting the calculation on block and section breaks. (Optional.)
Row | Col	The calculation direction (optional)

Output

The list of values returned by the measure with missing values supplied by interpolation

Example

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

Day	Value	Interpolation([Value])
Monday	12	12
Tuesday	14	14
Wednesday		15
Thursday	16	16
Friday		17
Saturday		18
Sunday	19	19

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.
- 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.

Ln

Description

Returns the natural logarithm of a number

Function Group

Numeric

Syntax

```
number Ln(number input_number)
```

Input

input_number	Any number

Output

The natural logarithm of the input number

Example

`Ln(10)` returns 2

Log

Description

Returns the logarithm of a number in a specified base

Function Group

Numeric

Syntax

```
number Log(number input_number; number base)
```

Input

input_number	Any number
base	The base

Output

The logarithm of the input number in the specified base

Example

`Log(125;5)` returns 3

Log10

Description

Returns the base 10 logarithm of a number

Function Group

Numeric

Syntax

```
number Log10(number input_number)
```

Input

input_number	A number

Output

The base 10 logarithm of the input number

Example

`Log10(100)` returns 2

Mod

Description

Returns the modulus (remainder) of a number divided by another number

Function Group

Numeric

Syntax

```
number Mod(number dividend; number divisor)
```

Input

dividend	The dividend
divisor	The divisor

Output

The remainder when the dividend is divided by the divisor

Example

`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

```
number Power(number input_number; number power)
```

Input

input_number	The input number
power	The power

Output

The number raised to the power

Example

`Power(10;2)` returns 100

Rank

Description

Ranks a measure by a dimension or set of dimensions

Function Group

Numeric

Syntax

```
integer Rank(measure; (dimensions); [TOP|BOTTOM]; [(reset_dimensions)])
```

Input

measure	A measure
dimensions	A dimension or list of dimensions (optional)

TOP|BOTTOM	Ranking order (optional) TOP - descending; BOTTOM - ascending
reset_dimensions	The list of dimensions used to reset the ranking (optional)

Output

The measure ranking based on the ranking dimension(s)

Examples

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

Country	Revenue	Rank
France	835,420	2
US	2,451,104	1

In the following table the rank is given by **Rank([Revenue];([Country]);BOTTOM)**. The BOTTOM argument means that the measures are ranked in descending order.

Country	Revenue	Rank
France	835,420	1
US	2,451,104	2

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

Country	Resort	Revenue	Rank

France	French Riviera	835,420	3
US	Bahamas Beach	971,444	2
US	Hawaiian Club	1,479,660	1

In the following table the rank is given by **Rank([Revenue];([Country];[Year]);([Country])).**

The rank is reset on the Country dimension.

Country	Year	Revenue	Rank
France	FY1998	295,940	1
France	FY1999	280,310	2
France	FY2000	259,170	3
US	FY1998	767,614	3
US	FY1999	826,930	2
US	FY2000	856,560	1

Notes

- If you do not specify a ranking dimension(s), 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

Related Topics

- *Bottom operator* on page 191

- *Top operator* on page 196

Round

Description

Rounds a number to a specified number of decimal places

Function Group

Numeric

Syntax

```
number Round (number input_number; integer
num_places)
```

Input

input_number	The number to be rounded
num_places	The number of decimal places to which the number is to be rounded

Output

The number rounded to the specified number of decimal places

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

Notes

- If **num_places** > 0, **input_number** is rounded to num_places decimal places
- If **num_places** = 0, **input_number** is rounded to the nearest integer
- If **num_places** < 0, the decimal point is moved **num_places** to the left, then the number is rounded, then the decimal point is moved **num_places** to the right

Sign

Description

Returns the sign of a number

Function Group

Numeric

Syntax

```
number Sign(number input_number)
```

Input

input_number	The number whose sign you want to determine

Output

The sign of the number (-1 = negative; 0 = zero; 1 = positive)

Example

`Sign(3)` returns 1

`Sign(-27.5)` returns -1

Sin

Description

Returns the sine of an angle in radians

Function Group

Numeric

Syntax

```
number Sin(number angle)
```

Input

angle	The angle in radians

Output

The sine of the angle

Example

`Sin(234542)` returns -0,116992.

Sqrt

Description

Returns the square root of a number

Function Group

Numeric

Syntax

```
number Sqrt(number input_number)
```

Input

input_number	The number whose square root you want to find

Output

The square root of the number

Example

`Sqrt(25)` returns 5

Tan

Description

Returns the tangent of an angle

Function Group

Numeric

Syntax

```
number Tan(number angle)
```

Input

angle	The angle

Output

The tangent of the angle

Example

`Tan(90)` returns -2

Truncate

Description

Returns a number truncated to n decimal places

Function Group

Numeric

Syntax

```
number Truncate(number input_number; integer
num_places)
```

Input

input_number	The input number
num_places	The number of decimal places

Output

The input number truncated to **num_places** decimal places

Example

`Truncate(3.423;2)` returns 3.42

Misc functions

BlockName

Description

Returns the name of a block

Function Group

Misc

Syntax

```
string BlockName()
```

Input

None

Output

The name of the block in which the function is placed

Example

`BlockName()` returns "Block1" if it is placed in a block called "Block1"

ColumnNumber

Description

Returns the number of the column in a table

Function Group

Misc

Syntax

```
integer ColumnNumber()
```

Input

None

Output

The column number

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()
```

Input

None

Output

The current user

Example

`CurrentUser()` returns "gkn" if the current user InfoView login is "gkn"

ForceMerge

Description

Forces Web Intelligence to account for synchronized dimensions in measure calculations when the synchronized dimensions do not appear in the calculation context of the measure

Function Group

Misc

Syntax

```
number ForceMerge([measure])
```

Input

measure	Any measure

Output

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

Example

`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.

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.

GetContentLocale

Description

Returns the document locale

Function Group

Misc

Syntax

```
string GetContentLocale()
```

Input

None

Output

The locale of the document content

Example

`GetContentLocale()` returns "fr_FR" if the content locale is "French (France)"

GetLocale

Description

Returns the current locale

Function Group

Misc

Syntax

```
string GetLocale()
```

Input

None

Output

The current locale

Example

`GetLocale()` returns "en_US" if the current locale is "en_US"

If

Description

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

Function Group

Misc

Syntax

```
If boolean_expression Then true_expression {Else false_expression | ElseIf boolean_expression Then true_expression Else false_expression}
```

Input

boolean_expression	An expression that returns true or false
true_expression	The expression whose value is returned if `boolean_expression` is true
false_expression	The expression whose value is returned if `boolean_expression` is false

Output

The value of `true_expression` or `false_expression`

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.

Notes

- `true_expression` and `false_expression` can mix datatypes (as in the second example above) except for date/time and number data types. If the formula mixes date/time and number datatypes it returns **#DATATYPE**.
- You can place parentheses around `boolean_expression`, `true_expression` and `false_expression`.
- You can use the boolean operators (**AND**, **Between**, **Inlist**, **OR**, **NOT**) with the If function.
- Web Intelligence still supports the original syntax for the If function. See the link below for further details.

Related Topics

- *And operator* on page 186
- *Between operator* on page 188
- *Inlist operator* on page 189
- *Or operator* on page 186
- *Not operator* on page 187

LineNumber

Description

Returns the line number in a table

Function Group

Misc

Syntax

```
integer LineNumber()
```

Input

None

Output

The line number

Example

`LineNumber()` returns 2 when the function appears at the second line in a table

Note

Numbering of the lines in a table starts with the header, which is line 1.

NameOf

Description

Returns the name of an object

Function Group

Misc

Syntax

```
string NameOf([object])
```

Input

[object]	Any object

Output

The name of the object

Example

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

Note:

Web Intelligence uses the NameOf() function in column and row headers in reports

NoDrillFilter

Description

Ignores drill filters when performing a calculation

Function Group

Misc

Syntax

```
expression_output_type NoDrillFilter(expression)
```

Input

expression	Any expression

Output

The result of the expression, ignoring drill filters.

NoFilter

Description

Tells Web Intelligence to ignore filters when calculating an expression.

Function Group

Misc

Syntax

```
expression_output_type NoFilter(expression;[All|Drill])
```

Input

expression	Any expression
All/Drill (optional)	• Not specified - ignore report and block filters • All - ignore all filters • Drill - ignore report filters and drill filters

Output

The result of the expression, ignoring filters.

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.

Notes

- `NoFilter (expression;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 expressions to which `NoFilter(expression;Drill)` is applied.

NumberOfPages

Description

Returns the number of pages in a report

Function Group

Misc

Syntax

```
integer NumberOfPages()
```

Output

The number of pages in the report

Example

`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()
```

Output

The number of the current page

Example

`Page()` returns 2 if it appears in the second page of the report

Previous

Description

Returns the previous value of an object

Function Group

Misc

Syntax

`any_type Previous(object|SELF;(reset_dimensions))`

Input

object| SELF	Any object
reset_dimensions	The list of dimensions used to reset the calculation (optional)

Output

The previous value of the object

Examples

`Previous(Country)` returns the following values in the following table:

Country	Revenue	Previous
US	5,000,000	

UK	2,000,000	UK
France	2,100,000	US

`Previous(Revenue)` returns the following values in the following table:

Country	Revenue	Previous
US	5,000,000	
UK	2,000,000	5,000,000
France	2,100,000	2,000,000

`Previous(Revenue;(Country))` returns the following values in the following table:

Country	Region	Revenue	Previous
US	North	5,000,000	
	South	7,000,000	5,000,000
UK	North	3,000,000	
	South	4,000,000	3,000,000

`Previous(Revenue)` returns the following values in the following crosstab:

	2004	Previous	2005	Previous
US	5,000,000		6,000,000	5,000,000
UK	2,000,000	6,000,000	2,500,000	2,000,000
France	3,000,000	2,500,000	2,000,000	3,000,000

`Previous(`*`Revenue`*`)` returns the following values in the following table with a break on *Country*:

Country	Region	Revenue	Previous
US	North	5,000,000	
	South	7,000,000	5,000,000
US		12,000,000	

Country	Region	Revenue	Previous
UK	North	3,000,000	7,000,000
	South	4,000,000	3,000,000
UK		7,000,000	12,000,000

2 * `Previous`(**SELF**) returns the sequence 2, 4, 6, 8, 10...

Notes

- You can use extended syntax context operators with the `Previous()` function.
- The **SELF** operator allows you to refer to the previous value of a cell when it contains content other than one report object
- You must always place dimensions in parentheses even if there is only one dimension in the list of reset dimensions.
- When you specify a set of reset dimensions you must separate them with semi-colons.
- Web Intelligence applies the `Previous()` function after applying all report, section and block filters.
- You cannot apply a filter on a formula that uses **Previous().**
- Web Intelligence applies the `Previous()` function 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.

- Web Intelligence calculates previous values in a crosstab in a Z pattern (left to right on each row, with the last value on each row carried over to the next).
- `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.

Related Topics

- *Self operator* on page 196

RefValue

Description

Returns the reference value of a report object when data tracking is activated

Function Group

Misc

Syntax

```
object_data_type RefValue([object])
```

Input

None

Output

The value of the report object

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.

RowIndex

Description

Returns the number of a row

Function Group

Misc

Syntax

```
integer RowIndex()
```

Input

None

Output

The number of the row

Example

`RowIndex()` returns 0 when it appears on the first row of a table

Notes

- Row numbering starts at 0
- **RowIndex()** returns #MULTIVALUE when placed in a table header or footer

UniqueNameOf

Description

Returns the unique name of an object

Function Group

Misc

Syntax

```
string UniqueNameOf([object])
```

Input

[object]	Any object

Output

The unique name of the object

Example

`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 `[Sales Revenue] - [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. The following table describes them:

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to

You use conditional operators with the If function, as in:

```
If ([Revenue] >= 10000;‘High’;‘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

Logical operators are used in expressions that return True or False. You use such expressions in the If function. The Web Intelligence logical operators are AND, OR, NOT, Between and InList. For example, the formula

```
If ([Resort] = ‘Bahamas Beach’ OR [Resort]=’Hawaiian Club’;
‘US’; ‘France’)
```

returns “US” if the resort is “Bahamas Beach or “Hawiian Club”, “France” otherwise.

The formula

```
[Resort] = ‘Bahamas Beach’ OR [Resort]=’Hawaiian Club’
```

returns True or False, True if the Resort variable is equal to ‘Bahamas Beach’ or ‘Hawaiian Club’, False otherwise.

And operator

Description

Returns true if all boolean expressions in a list are true

Operator type

Boolean

Syntax

```
<boolean_expression> AND
<boolean_expression> AND <boolean_expression>...
```

Input

<boolean_expression>	Any boolean expression

Output

True if all the expressions in the list are true; false otherwise

Example

`[Country] = "US" AND [Revenue] Between (1000000;2000000) AND [Year] = "2005"` returns true if Country = "US", Revenue = 1500000 and Year = "2005"

Or operator

Description

Returns true if at least one boolean expression in a list is true

Operator type

Boolean

Syntax

```
<boolean_expression> OR
<boolean_expression> OR <boolean_expression>...
```

Input

<boolean_expression>	Any boolean expression

Output

True if at least one expression in the list istrue; false otherwise

Example

`[Country] = "US" OR [Revenue] Between (1000000;2000000) OR [Year] = "2005"` returns true if Country = "US", Revenue = 1500000 and Year = "2004"

Not operator

Description

Returns the opposite of a boolean expression

Operator type

Boolean

Syntax

```
NOT (<boolean_expression>)
```

Input

<boolean_expression>	Any boolean expression

Output

The opposite of the boolean expression

Example

`NOT ([Country] = "US")` returns false if Country="US"

Between operator

Description

Determines whether a variable is between two values

Operator type

Boolean

Syntax

```
boolean Between([firstvalue];[secondvalue])
```

Input

secondvalue	Any report variable
firstvalue	Any report variable

Output

True if the variable is between the two values; false otherwise

Example

`[Sales Revenue] Between (10000;20000)` returns true if the sales revenue is between 10000 and 20000

`If ([Sales Revenue] Between (200000;500000);"Medium Rev enue";"Low/High Revenue")` returns "Medium Revenue" if [Sales Revenue] is 300000

Notes

- You use the Between operator 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.)

Related Topics

- *If* on page 173
- *Where operator* on page 197
- *And operator* on page 186
- *Inlist operator* on page 189
- *Or operator* on page 186
- *Not operator* on page 187

Inlist operator

Description

Determines whether a variable is in a list of values

Operator type

Boolean

Syntax

```
boolean Inlist(list_of_values)
```

Input

list_of_values	A list of values

Output

True if the variable is in the list of values; false otherwise

Example

`[Country] Inlist ("US";"UK')` returns true if [Country] is "US"

`Average([Revenue]) Where ([Country] Inlist ("US";"UK"))` returns the average revenue of the US and the UK

Note

- It is the combination of **object + InList** that returns a boolean value, not **InList** alone. You must be aware of this when combining **InList** with other boolean operators. For example, you use **NOT** like this:

NOT ([object] InList(list))

The syntax **[object] NOT InList(list)** is incorrect and returns an error.

- You use the **InList** operator with the **If** function and the **Where** operator.

Related Topics

- *If* on page 173
- *Where operator* on page 197

Function-specific operators

Some Web Intelligence functions can take specific operators as arguments. For example, the Previous function can take the SELF operator.

All/Drill operator

The All/Drill operator controls 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

Related Topics

- *NoFilter* on page 176

Bottom operator

Description

Sets the `Rank()` function to rank from bottom to top

Syntax

`integer Rank(`*`argument_list`*`)`

Input

argument_list	List of arguments for the Rank function including Bottom

Output

Causes the ranking to rank from bottom to top

Example

`Rank(`*`Revenue`*`;(`*`Country`*`);TOP)`

ranks US as 2 and UK as 1 if the associated revenues are 1,000,000 and 900,000.

Related Topics

- *Rank* on page 161

Distinct/All operator

Description

Tells Web Intelligence whether to take account of distinct values in a calculation

Syntax

`Function (argument_list)`

Input

argument_list	A function argument list including DISTINCT or ALL

Example

`Count([Revenue]; DISTINCT)` returns 3 if [Revenue] has the values (5;5;6;4)

`Count([Revenue]; ALL)` returns 4 if [Revenue] has the values (5;5;6;4)

Related Topics

- *Count* on page 51

IncludeEmpty operator

Description

Tells Web Intelligence to take account of empty values when making a calculation

Syntax

```
Function (argument_list)
```

Input

argument_list	A function argument list including the IncludeEmpty operator

Example

`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 61
- *RunningCount* on page 63

Index operator

You use the `Index` operator in the `UserResponse` function. When you include the operator, Web Intelligence returns the database primary keys of prompt values instead of the values themselves.

Related Topics

- *UserResponse* on page 133

Linear operator

Description

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.

NotOnBreak operator

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

PointToPoint operator

Description

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.

RegLeastSquares operator

Description

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.

Row and Col operators

Description
You use the ROW and COL operators to set the calculation direction of the following functions: `Percentage()`, `RunningAverage()`, `RunningCount()`, `RunningMax()`, `RunningMin()`, `RunningProduct()`, `RunningSum()`.

ROW and COL operators and the Percentage() function

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.

In a crosstab, **Percentage([Measure])** gives the following result:

Measure	Percentage	Measure	Percentage
100	10%	500	50%

200	20%	200	20%

Percentage([Measure];ROW) gives the following result:

Measure	Percentage	Measure	Percentage
100	16.7%	500	83.3%
200	50%	200	50%

Percentage([Measure];COL) gives the following result:

Measure	Percentage	Measure	Percentage
100	33.3%	500	83.3%
200	66.6%	200	16.7%

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.

ROW and COL operators and running aggregate functions

In a crosstab, **RunningSum([Measure])** or **RunningSum([Measure];ROW)** gives the following:

Measure	RunningSum	Measure	RunningSum
100	100	200	300
400	700	250	950

In a crosstab, **RunningSum([Measure];COL)** gives the following:

Measure	RunningSum	Measure	RunningSum
100	100	200	700
400	500	250	950

Self operator

Description

Allows the *Previous* on page 179 function to refer to the previous cell value when the cell contains content other than one report object

Syntax

```
Previous(SELF)
```

Example

`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 179

Top operator

Description

Sets the `Rank()` function to rank from top to bottom

Syntax

```
integer Rank (argument_list)
```

Input

argument_list	List of arguments for the Rank function including TOP

Output

Causes the ranking to rank from top to bottom

Example

```
Rank([Revenue];([Country]);TOP)
```

ranks US as 1 and UK as 2 if the associated revenues are 1,000,000 and 900,000.

Related Topics

- *Rank* on page 161

Where operator

Description

Restricts the data used to calculate the measure expression.

Syntax

```
[measure_expression] Where [boolean_expression]
```

[measure_expression]	Any measure expression
[boolean_expression]	Any boolean expression on dimensions or measures

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.

Note
You can use the boolean operators with the Where operator.

Related Topics
- *And operator* on page 186
- *Between operator* on page 188
- *Inlist operator* on page 189
- *Or operator* on page 186
- *Not operator* on page 187

StepLine operator

Description

Tells the `Interpolation` function to use step line interpolation to supply missing measure values.

Step line interpolation calculates missing values by using the previous value as the missing value.

Extended syntax context operators

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

Operator	Description
In	Specifies an explicit list of dimensions to use in the context.
ForEach	Adds dimensions to the default context
ForAll	Removes dimensions from the default context

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:

Year	Sales revenue	Max Quarterly Revenue
2001	$8096123.60	$2660699.50
2002	$13232246.00	$4186120.00
2003	$15059142.80	$4006717.50

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:

Year	Quarter	Sales revenue
	Q1	$2660700
	Q2	$2279003
	Q3	$1367841
	Q4	$1788580
2001		
	Max:	**2660699.5**

Year	Quarter	Sales revenue
	Q1	$3326172
	Q2	$2840651
	Q3	$2879303
	Q4	$4186120
2002		
	Max:	**4186120**

Year	Quarter	Sales revenue
	Q1	$3742989
	Q2	$4006718
	Q3	$3953395
	Q4	$3356041
2003		
	Max:	**4006717.5**

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

```
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:

Year	Sales revenue	Max Quarterly Revenue
2001	$8096123.60	$2660699.50
2002	$13232246.00	$4186120.00
2003	$15059142.80	$4006717.50

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:

Year	Quarter	Sales revenue	Yearly Total
2001	Q1	$2660700	$8096124
2001	Q2	$2279003	$8096124
2001	Q3	$1367841	$8096124
2001	Q4	$1788580	$8096124
2002	Q1	$3326172	$13232246
2002	Q2	$2840651	$13232246
2002	Q3	$2879303	$13232246
2002	Q4	$4186120	$13232246
2003	Q1	$3742989	$15059143
2003	Q2	$4006718	$15059143
2003	Q3	$3953395	$15059143
2003	Q4	$3356041	$15059143

To total revenues by year the input context needs to be (Year); by default it is (Year; Quarter). Therefore, you can remove Quarter from the input context by specifying ForAll ([Quarter]) in the formula, which looks like this:

```
Sum([Sales Revenue] ForAll ([Quarter]))
```

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

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

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

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.

When placed in...	References this data...
A block	Data in the whole block, ignoring breaks, respecting filters
A block break (header or footer)	Data in the whole block, ignoring breaks, respecting filters
A section (header, footer, or outside a block)	Not applicable
Outside any blocks or sections	Not applicable

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.

2001

Quarter	Sales revenue	First Half Average	Yearly Average
Q1	$2,660,700	$2,469,851.25	$8,096,123.60
Q2	$2,279,003	$2,469,851.25	$8,096,123.60
Sum:	**4,939,702.5**		

2002

Quarter	Sales revenue	First Half Average	Yearly Average
Q1	$3,326,172	$3,083,411.50	$13,232,246.00
Q2	$2,840,651	$3,083,411.50	$13,232,246.00
Sum:	**6,166,823**		

2003

Quarter	Sales revenue	First Half Average	Yearly Average
Q1	$3,742,989	$3,874,853.20	$15,059,142.80
Q2	$4,006,718	$3,874,853.20	$15,059,142.80
Sum:	**7,749,706.4**		

The Yearly Average column has the formula

```
Average([Sales revenue] In Section)
```

and the First Half Average column has the formula

```
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:

When placed in...	References this data...
A block	Data in the block

When placed in...	References this data...
A block break (header or footer)	Data in the block
A section (header, footer, or outside a block)	Data in the section
Outside any blocks or sections	Data in the report

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.

Year	Quarter	Sales revenue	Body
2001	Q1	$2,660,700	2,660,699.5
	Q2	$2,279,003	2,279,003
	Q3	$1,367,841	1,367,840.7
	Q4	$1,788,580	1,788,580.4
2001		**8,096,123.6**	

The Body column has the formula

```
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.

The Break keyword

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

When placed in...	References this data...
A block	Data in the part of a block delimited by a break
A block break (header or footer)	Data in the part of a block delimited by a break
A section (header, footer, or outside a block)	Not applicable
Outside any blocks or sections	Not applicable

Example: The Break keyword

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

Year	Quarter	Sales revenue	Break Total
2001	Q1	$2,660,700	8,096,123.6
	Q2	$2,279,003	8,096,123.6
	Q3	$1,367,841	8,096,123.6
	Q4	$1,788,580	8,096,123.6
2001			

The report has break on Year. The Break Total column has the formula:

```
Sum ([Sales Revenue]) In Break
```

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

The Report keyword

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

When placed in...	References this data...
A block	All data in the report

When placed in...	References this data...
A block break (header or footer)	All data in the report
A section (header, footer, or outside a block)	All data in the report
Outside any blocks or sections	All data in the report

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.

Year	Quarter	Sales revenue	Report Total
2001	Q1	$2,660,700	36,387,512.4
2001	Q2	$2,279,003	36,387,512.4
2001	Q3	$1,367,841	36,387,512.4
2001	Q4	$1,788,580	36,387,512.4
2002	Q1	$3,326,172	36,387,512.4
2002	Q2	$2,840,651	36,387,512.4
2002	Q3	$2,879,303	36,387,512.4
2002	Q4	$4,186,120	36,387,512.4
2003	Q1	$3,742,989	36,387,512.4
2003	Q2	$4,006,718	36,387,512.4
2003	Q3	$3,953,395	36,387,512.4
2003	Q4	$3,356,041	36,387,512.4

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

When placed in...	References this data...
A block	All data in the section
A block break (header or footer)	All data in the section
A section (header, footer, or outside a block)	All data in the section
Outside any blocks or sections	Not applicable

Example: The Section keyword

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

2001

Quarter	Sales revenue	Section Total
Q1	$2.660.700	8.095.814
Q2	$2.278.693	8.095.814
Q3	$1.367.841	8.095.814
Q4	$1.788.580	8.095.814

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]).

Troubleshooting Web Intelligence formulas

chapter 5

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.

#CONTEXT

#CONTEXT is related to the #INCOMPATIBLE and #DATASYNC error messages, both of which occur when a block contains a non-existent aggregation context. In the case of #INCOMPATIBLE the aggregation context is non-existent because the dimensions are incompatible; in the case of #DATASYNC the aggregation context is non-existent because the dimensions are from multiple unsynchronized data providers.

Example: Non-existent aggregation 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.

#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 approrpiate 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

f 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

chapter 6

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.

Related Topics

- *#UNAVAILABLE* on page 214

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 SET` operator. Web Intelligence uses multiple result sets and the `UNION` operator to simulate the effect of the `GROUPING SETS` clause.

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 `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 `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:

```
SELECT
  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:

Semester	Revenue
H1	#UNAVAILABLE
H2	#UNAVAILABLE

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:

Country	Region	Revenue
US	North	10000
US	South	15000
US	East	14000
US	West	12000

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

```
[Revenue] 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

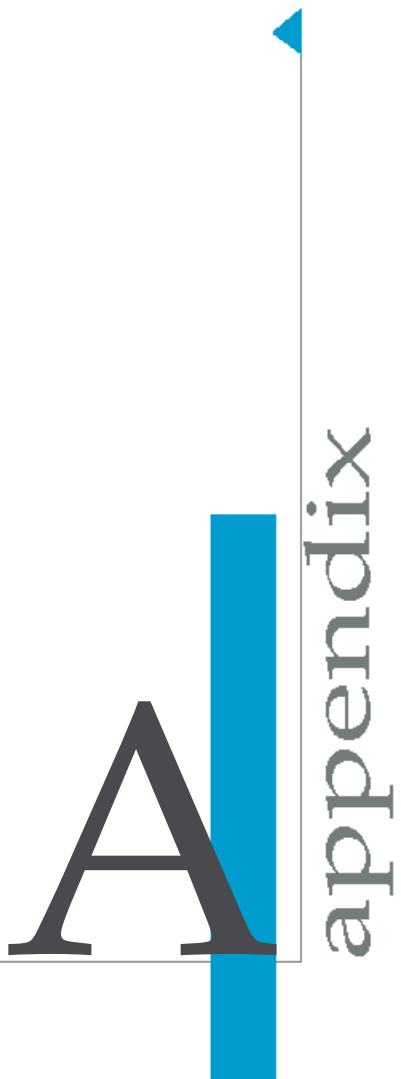
In general, Web Intelligence cannot return values for smart measures when a filter is applied to a dimension that impacts the calculation of the measure. Dimensions filtered by drill filters are an exception to this rule.

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.

Get More Help



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.

http://support.businessobjects.com/documentation/product_guides/

Additional developer resources

http://devlibrary.businessobjects.com

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

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Index