PUBLIC

SAP BusinessObjects Business Intelligence Suite

Document Version: 4.3 Support Package 4 – 2023-12-07

Using functions, formulas and calculations in Web Intelligence

© 2024 SAP SE or an SAP affiliate company. All rights reserved.

THE BEST RUN SAP

Content

1 Using functions, formulas and calculations for data analysis. 3

1.1 Document History: Web Intelligence Functions, Formulas and Calculations. 3

1.2 About this guide. 4

1.3 Using standard and custom calculations. 4

Introducing the formula editor. 4

Using standard and custom calculations. 5

1.4 Understanding calculation contexts. 12

Understanding calculation contexts. 12

1.5 Calculating values with smart measures. 24

Calculating values with smart measures. 24

1.6 Functions, operators and keywords. 32

Functions. 32

Function and formula operators. 260

Extended syntax keywords. 278

Rounding and truncating numbers. 283

Referring to members and member sets in hierarchies. 284

1.7 Building custom functions. 285

Overview of external functions 285

Defining a custom calculation. 288

Examples. 294

#EXTERNAL error message. 296

Trace log message errors. 296

1.8 Troubleshooting formulas. 297

Automatic rewrite formula mechanism. 297

Formula error and information messages. 298

1.9 Comparing values using functions. 304

Comparing values using the Previous function. 304

Comparing values using the RelativeValue function. 305

1 Using functions, formulas and calculations for data analysis

1.1 Document History: Web Intelligence Functions, Formulas and Calculations

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

Version	Date	Description
SAP BusinessObjects Web Intelligence 4.3 SP3	December 2022	The following sections have been updated or added to the guide: • New Reverse [page 97], ElementLinkingFilters [page 158], InputControlFilter [page 159], DescriptionOf [page 231], FormulaOf [page 233], and Next [page 242] functions added. • Updated ReportFilter [page 163] and ToDate [page 123].
SAP BusinessObjects Web Intelligence 4.3 SP1	December 2020	The following sections have been updated or added to the guide: • New RPos [page 101] function added. • Updated Pos [page 95].

SAP BusinessObjects Web Intelligence 4.3	June 2020	The following sections have been updated or added to the guide:
		• New DocumentDescription [page 152], DocumentParentFolder [page 154], DocumentPath [page 155], and NumberOfColumns [page 140] functions added.
		• You can now add comments within the code of a formula.
		• Updated:
		  • Trim [page 103], LeftTrim [page 92], and RightTrim [page 100]. You can now specify the characters you want to remove.
		  • QuerySummary [page 161] and DataProviderType [page 131]. Both functions returns new data provider types.

1.2 About this guide

The Using Functions, Formulas and Calculations in Web Intelligence guide provides detailed information on the advanced calculation capabilities that you can use when you perform data analysis.

This guide also provides a syntax reference to the available functions and operators.

1.3 Using standard and custom calculations

1.3.1 Introducing the formula editor

The formula editor is the central place to create advanced calculations and variables.

It has been designed to facilitate objects manipulation and create formulas or variables quickly using its built-in code editor.

Code editor

The code editor offer multiple features to help you write formulas:

- Parenthesis matching
- Syntax analysis
- Color coding
- Auto-completion
- Keyboard shortcuts (`Ctrl` + `C`, `Ctrl` + `Z`, and so on)
- Line numbering

Use the dedicated toggle to enable line wrapping ( ) as well as syntax analysis and color coding ( ).

A simplified version of the code editor is available in the formula bar and side panels in Web Intelligence. For example, a simplified version of the code editor can be found in formula text fields for cells, blocks, and reports when you navigate to > > *Display Settings* > *Hide* > *Hide when formula is true*.

Objects, functions, and operators panels

Next to the text editor, three panels allow you to drag and drop objects quickly within the code editor: the *Objects* ( ) panel, the *Functions* ( ) panel, and the *Operators* ( ) panel. Each of these panels can be resized or hidden if necessary.

You can use the *Expand all* (resp. *Collapse all*) button on top of each panel to expand (resp. collapse) the side panel content.

To get help on any of the object available, hover over one of them to get access to a tooltip with detailed information. If you click one of the function or operator available, there's also a link at the bottom right-hand corner of the dialog that redirects you to its detailed documentation on the help portal. Hovering on an object brings up a tooltip with information taken from the objects dictionary.

1.3.2 Using standard and custom calculations

You can use standard calculation functions to make quick calculations on data.

If standard calculations are not sufficient for your needs, you can use the formula language to build custom calculations.

1.3.2.1 Standard calculations

You can use standard calculation functions to make quick calculations on data.

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.
Min	Displays the minimum value of the selected data.
Max	Displays the maximum value of the selected data.
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.

When you apply standard calculations to table columns, the calculation results appear in footers. One footer is added for each calculation.

1.3.2.2 Using formulas to build custom calculations

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

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

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

Example: Showing average revenue per sale

If you have a report with Sales Revenue and Number Sold objects and you want to add revenue per sale to the report, the calculation `[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.

1.3.2.2.1 Using variables to simplify formulas

Variables are useful to break down formulas into manageable parts and make them easier to read. They also make building a formula less error-prone.

You'll find the variables in the *Objects* pane, under the *Variables* section, along with other objects in the query.

Use the *Description* field to provide context and details about a specific variable. The description is displayed in the *Query Panel* when you hover over the variable. You can edit this description when creating, editing or renaming a variable.

1.3.2.3 Working with functions

A custom calculation sometimes contains only report objects, 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 Information

Function and formula operators [page 260]
Functions [page 32]

1.3.2.3.1 Including functions in cells

The text in report cells always begins with '=' .

Literal text appears in quotation marks, while formulas appear without quotation marks. For example, the formula 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.

1.3.2.3.2 Function syntax

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

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

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

Here is the syntax of the `Abs` function:

```
num Abs(number)
```

This syntax tells you that the `Abs` function takes a single number as input and returns a number as output.

1.3.2.3.3 Examples of functions

This topic offers examples of functions used in formulas.

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([Query 1];"Enter values for State:")
```

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

Quarterly Revenues for Illinois

Year	Quarter	Sales revenue
2004	Q1	$256,454
2004	Q2	$241,149
2004	Q3	$107,006
2004	Q4	$133,306
2004	**Total**	**$737,914**

Year	Quarter	Sales revenue
2005	Q1	$334,297
2005	Q2	$254,722
2005	Q3	$230,573
2005	Q4	$331,067
2005	**Total**	**$1,150,659**

Year	Quarter	Sales revenue
2006	Q1	$255,658
2006	Q2	$354,724
2006	Q3	$273,186
2006	Q4	$250,517
2006	**Total**	**$1,134,085**

Example: Calculating a percentage using the Percentage function

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

Year	Quarter	Sales revenue	Percentage
2004	Q1	$256,454	0.08
2004	Q2	$241,149	0.08
2004	Q3	$107,006	0.04
2004	Q4	$133,306	0.04
2005	Q1	$334,297	0.11
2005	Q2	$254,722	0.08
2005	Q3	$230,573	0.08
2005	Q4	$331,067	0.11
2006	Q1	$255,658	0.08
2006	Q2	$354,724	0.12
2006	Q3	$273,186	0.09
2006	Q4	$250,517	0.08
		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.

2004

Year	Quarter	Sales revenue	Percentage
2004	Q1	$256,454	0.35
2004	Q2	$241,149	0.33
2004	Q3	$107,006	0.15
2004	Q4	$133,306	0.18
		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.

2004	0.22

Year	Quarter	Sales revenue
2004	Q1	$256,454
2004	Q2	$241,149
2004	Q3	$107,006
2004	Q4	$133,306

2005	0.38

Year	Quarter	Sales revenue
2005	Q1	$334,297
2005	Q2	$254,722
2005	Q3	$230,573
2005	Q4	$331,067

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))
```

2004

Year	Quarter	Sales revenue	Percentage of Year	Percentage of Total
2004	Q1	$256,454	0.08	0.35
2004	Q2	$241,149	0.08	0.33
2004	Q3	$107,006	0.04	0.15
2004	Q4	$133,306	0.04	0.18

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

Related Information

Modifying the default calculation context with extended syntax [page 20]

1.3.2.3.3.1 Simplifying a variance formula with variables

Variance is a statistical term. The variance of a set of values measures the spread of those values around their average.

The `Var` function calculates the variance in one step, but manual calculation of variance provides a good example of how to simplify a complex formula using variables. To calculate the variance manually you need to:

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

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

```
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)
```

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 becomes the following:

```
Sum ([Difference Squared])/[Number of Observations] - 1)
```

This formula 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.

1.4 Understanding calculation contexts

1.4.1 Understanding calculation contexts

The calculation context is the data that a calculation takes into account to generate a result.

This means that the value given by a measure is determined by the dimensions used to calculate the measure.

A report contains two kinds of objects:

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

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

The calculation context of a measure has two components:

- the dimension or list of dimensions that determine the measure value

- the part of the dimension data that determines the measure value

The calculation context has two components:

- The input context
- The output context

Related Information

The input context [page 13]
The output context [page 14]

1.4.1.1 The input context

The input context of a measure or formula is the list of dimensions that feed into the calculation.

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

Example: Specifying an input context

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

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

In other words, the section cells 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.

1.4.1.2 The output context

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

Example: Specifying an output context

The following report shows revenue by year and quarter, with a break on year, and the minimum revenue calculated by year:

Year	Quarter	Sales revenue
2004	Q1	$2,660,700
	Q2	$2,278,693
	Q3	$1,367,841
	Q4	$1,788,580
2004		
	Min:	**$1,367,841**

Year	Quarter	Sales revenue
2005	Q1	$3,326,172
	Q2	$2,840,651
	Q3	$2,879,303
	Q4	$4,186,120
2005		
	Min:	**$2,840,651**

Year	Quarter	Sales revenue
2006	Q1	$3,742,989
	Q2	$4,006,718
	Q3	$3,953,395
	Q4	$3,356,041
2006		
	Min:	**$3,356,041**

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 ([Sales revenue]) In ([Year])
```

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

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

Year	Quarter	Sales revenue	Min By Year
2004	Q1	$2,660,700	$1,367,841
2004	Q2	$2,278,693	$1,367,841
2004	Q3	$1,367,841	$1,367,841
2004	Q4	$1,788,580	$1,367,841
2005	Q1	$3,326,172	$2,840,651
2005	Q2	$2,840,651	$2,840,651
2005	Q3	$2,879,303	$2,840,651
2005	Q4	$4,186,120	$2,840,651
2006	Q1	$3,742,989	$3,356,041
2006	Q2	$4,006,718	$3,356,041
2006	Q3	$3,953,395	$3,356,041
2006	Q4	$3,356,041	$3,356,041

You can see that the Min By Year column contains the minimum revenues that appear in the break footers in the previous report.

Notice that in this example, the input context is not specified because it is the default context (Year, Quarter) for the block. In other words, the output context determines which revenue by year and quarter to output. In full, with both input and output formulas explicitly specified, the formula looks like this:

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

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

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

1.4.1.3 Default calculation contexts

A measure has a default calculation context depending on its place in the report.

The figures returned by a measure depend on the dimensions with which it is associated. This combination of dimensions represents the calculation context.

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

Example: Default contexts in a report

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

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 revenues in the report
Section cell total	8000	Year
Customer total	1000, 3000, 4000	Year;Customer
Block footer total	8000	Year

Related Information

Understanding calculation contexts [page 12]
Modifying the default calculation context with extended syntax [page 20]

1.4.1.3.1 Default contexts in a vertical table

A vertical table is a standard report table with headers at the top, data going from top to bottom and footers at the bottom.

The default contexts in a down table are:

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

1.4.1.3.2 Default contexts in a horizontal table

The default contexts for a horizontal table are the same as those for a vertical table.

A horizontal table is like a vertical table turned on its side.

The appearance of the horizontal table depends on the Preferred Viewing Locale you have selected in the BI launch pad preferences. Some locales, like the English locale, use the left-to-right (LTR) interface positioning, whereas others, like the Arabic locale, use the right-to-left (RTL) interface positioning.

In an LTR locale, headers appear at the left, data goes left to right and footers appear at the right. In an RTL locale, headers appear at the right, data goes right to left and footers appear at the left.

1.4.1.3.3 Default contexts in a crosstab table

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.
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.

The calculation is in the...	The input context is...	The output context is...
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

1.4.1.3.4 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 section:

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**	

1.4.1.3.5 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**

1.4.1.4 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.

Related Information

Extended syntax keywords [page 278]
Extended syntax operators [page 20]

1.4.1.4.1 Extended syntax 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.

1.4.1.4.1.1 In context operator

The `In context` operator specifies dimensions explicitly in a context.

Example: Using In to specify the dimensions in a context

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

Year	Sales revenue	Max Quarterly Revenue
2001	$8,096,123.60	$2,660,699.50
2002	$13,232,246.00	$4,186,120.00
2003	$15,059,142.80	$4,006,717.50

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
2001	Q1	$2,660,699.50
2001	Q2	$2,279,003.00
2001	Q3	$1,367,841.00
2001	Q4	$1,788,580.00
	Max:	$2,660,699.50
Year	**Quarter**	**Sales revenue**
	Q1	$3,326,172.00
	Q2	$2,840,651.00
	Q3	$2,879,303.00
	Q4	$4,186,120.00

Year	Quarter	Sales revenue
	Max:	$4,186,120.00

Year	Quarter	Sales revenue
	Q1	$3,742,989.00
	Q2	$4,006,717.50
	Q3	$3,953,395.00
	Q4	$3,356,041.00
	Max:	$4,006,717.50

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

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

1.4.1.4.1.2 ForEach context operator

The `ForEach` operator adds dimensions to a context.

Example: Using ForEach to add dimensions to a context

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

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]).

1.4.1.4.1.3 ForAll context operator

The `ForAll` context operator removes dimensions from a context.

Example: Using ForAll to remove dimensions from a context

You have a report showing Year, Quarter and Sales revenue and you want to add a column that shows the total revenue in each year, as shown in the following block:

Year	Quarter	Sales revenue	Yearly Revenue
2004	Q1	$2,660,700	$8,096,124
2004	Q2	$2,279,003	$8,096,124
2004	Q3	$1,367,841	$8,096,124
2004	Q4	$1,788,580	$8,096,124
2005	Q1	$3,326,172	$13,232,246
2005	Q2	$2,840,651	$13,232,246
2005	Q3	$2,879,303	$13,232,246
2005	Q4	$4,186,120	$13,232,246
2006	Q1	$3,742,989	$15,059,143
2006	Q2	$4,006,718	$15,059,143
2006	Q3	$3,953,395	$15,059,143
2006	Q4	$3,356,041	$15,059,143

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.

1.5 Calculating values with smart measures

1.5.1 Calculating values with smart measures

Smart measures are measures whose values are calculated by the database (relational or OLAP) on which a universe is based.

They differ from classic measures, which are calculated from the detailed values returned by the database. The data returned by smart measures is aggregated in ways not supported natively by the Web Intelligence component of the SAP BusinessObjects Business Intelligence platform.

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

When you edit such a report, automatically the #TOREFRESH message is inserted in the report reminding you that the report should be refreshed in order to reflect the changes. You can choose to update the report automatically by selecting the Auto-refresh document option in the Document properties dialog

> ⓘ Note
>
> The measure delegation is static and defined based on the report definition at design time. In some cases (formula based on “if [choice]= 1 then [dimension 1) else [dimension 2]”) the dimensional context is variable at run time. In this case the system cannot delegate the measure calculation and returns an empty value.

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

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

1.5.1.1 Grouping sets and smart measures

A grouping set is a set of dimensions that generates a result for a measure.

The generated SQL that returns the data in a smart measure includes grouping sets for all the aggregations of that measure that are included in the report.

Example: Grouping sets in a query

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

- Total smart measure value
- smart measure value by (Country, Region, City)
- smart measure value by (Country, City)
- smart measure value by (City)
- smart measure value by (Region, City)
- smart measure value by (Region)
- smart measure value by (Country, Region)
- smart measure value by (Country)

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

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

1.5.1.1.1 Management of grouping sets

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

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

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

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

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

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

1.5.1.2 Smart measures and the scope of analysis

When you build a query with a scope of analysis, the initial grouping set contains the result objects, but not the scope objects.

The query does not generate all the possible grouping sets from the combination of the result objects and the scope objects.

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

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

1.5.1.3 Smart measures and SQL

1.5.1.3.1 Grouping sets and the UNION operator

Some databases support grouping sets explicitly with the `GROUPING SETS` operator.

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

Example: Grouping sets retrieved with the UNION operator

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

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

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

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

```
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, the (Country) grouping set is required. 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
```

1.5.1.4 Smart measures and formulas

1.5.1.4.1 Smart measures and dimensions containing formulas

If a formula or variable appears as a dimension in the calculation context of a smart measure, and the formula determines the grouping set required by the measure, the values of the smart measure can be displayed.

For example, smart measures and dimensions now return values for:

- A URL created with hyperlink wizard.
- Simple concatenation on a dimension (or blank removal).
- When FormatDate is used on [date]

> **Note**
>
> The message #UNAVAILABLE is stil returned for the following functions: ForEach, ForAll, In, Where, Rank, Previous, RelativeValue, RelativeDate, TimeDim, and in the Aggregation function when Min, Max, Last, or First are used in the formula: if ([selection] =1) then [dim1] else [dim2]

1.5.1.4.2 Smart measures in formulas

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

For example, a report contains a block as follows:

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

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

```
[Revenue] ForAll ([Region])
```

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

Related Information

ForAll context operator [page 23]

1.5.1.5 Smart measures and filters

1.5.1.5.1 Restrictions concerning smart measures and filters

A smart measure can be evaluated in the body of a table when there is no filter in the table or in the parent context (a report filter).

The following table describes how smart measures are evaluated when filters are present.

How smart measures are evaluated when a filter is present in the report

When the filter is on a ...	The smart measure is evalueated this way
Measure	The smart measure will be correctly evaluated, but some rows will be removed from the table.
Dimension that is already part of the table axis	The smart measure will be correctly evaluated, but some rows will be removed from the table. The smart measure can be evaluated, because there is no aggregation after the filtering.
Dimension that is not part of the axis of the table, and when the filter operand is mono-value (the filter will return one value/row).	The smart measure will be correctly evaluated. The smart measure can be evaluated because there is no aggregation after the filtering.

When the filter is on a ...	The smart measure is evalueated this way
Dimension that is not part of the axis of the table, and if the filter operand is multivalue (the filter can return many values/rows).	The smart measure can't be evaluated (#UNAVAILABLE is displayed) because in this case, filtering is done before aggregation, and for one row of the table, aggregation is required.

1.5.1.5.2 Smart measures and filters on dimensions

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

This also applies when a report filter comes from an input control.

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

> ⓘ Note
>
> A multi-valued filter filters on multiple values using operators such as Greater Than, In List or Less Than. You can apply single-valued filters such as Equal To without generating the #UNAVAILABLE error.

> ⓘ Note
>
> There is a workaround for cases which do not require aggregation: Define the formula as variable whose qualification is a measure and be sure that the used dimension is included in the block with the variable (you can hide that column for a better display).

Example: A smart measure and a filter on a dimension

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

Country	Revenue
France	#UNAVAILABLE
US	#UNAVAILABLE
Sum:	#UNAVAILABLE

If you restrict Product to "Jackets" only, the values are displayed.

Country	Revenue
US	971,444
Sum:	971,444

> **Note**
>
> A multivalue filter on Country will return results because the filter is in the table.

1.5.1.5.3 Filtering smart measures

The value in the table footer must be the aggregation of what the user sees in the table.

If what user sees in the table is filtered locally, then the system cannot return delegated aggregation of what is locally filtered.

Example: Filtering a smart measure

Country	OrderAmountDel
Brazil	28,833.36
China	51,384.33
France	68,630.22
USA	3,529,511.14
Total:	3,678,359.05
Sum:	3,678,359.05

When the data in the following table is filtered by OrderAmountDel > 60,000

The table shows the rows for which the OrderAmountDel in the context of table (per country) is greater than 60,000:

Sum in footer calculates the sum of the visible rows;

Total returns #UNAVAILABLE because the calculation is pushing the aggregation to the back end, but because Web Intelligence has performed local filtering, the aggregation cannot be delegated.

Country	OrderAmountDel
France	68,630.22
USA	3,529,511.14
Total:	#UNAVAILABLE
Sum:	3,598,141.36

1.5.1.5.4 Smart measures and drill filters

A drill filter is a single valued filter.

You can drill using the drill bar directly.

1.5.1.5.5 Smart measures and nested OR filters

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

This is because the smart measure has to be aggregated locally after some local processing (for example, filtering; some specific Web Intelligence formula) and this is not delegated.

1.6 Functions, operators and keywords

1.6.1 Functions

Formula functions are divided into several categories.

> **Note**
>
> In the following languages, the functions are not translated: Chinese, Japanese, Hungarian, Polish, Turkish, Thai, and Russian. They appear in the interface in English.

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
Set	Returns sets of members from hierarchies

1.6.1.1 Custom formats

You can define how to display any numeric or date / time values through custom formats.

The following table lists the tokens you can use to create these custom formats:

Format Token	Description	Example
`#`	The corresponding digit. If the number has less digits than the number of # characters used to specify the format, no leading zeros are inserted.	'12345' with the format `#,##0` gives '12,345' (if your locale defines the grouping separator as a comma) or '12 345' (if your locale defines the grouping separator as a space)
`0`	The corresponding digit. If the number has less digits than the number of `0` characters used to specify the format, a leading zero(s) is inserted before the number.	'123' with the format `#0,000` gives '0,123'
`,`	The grouping separator as defined by your locale.	'1234567' with the format `#,##0` gives '1,234,567' (if you locale defines the grouping separator as a comma) or '1 234 567' (if your locale defines the grouping separator as a non-breaking space)
`.`	The decimal separator as defined by your locale.	'12.34' with the format `#.#0` gives '12.34' (if your locale defines the decimal separator as a period) or '12,34' (if your locale defines the decimal separator as a comma)
`[%]%`	Displays a percentage sign (%) after the result and multiplies the result by 100.	0.50 becomes 50%.
`%`	The % sign after the result, but does not multiply the result by 100.	0.50 becomes 0.50%
	A non-breaking space ( )	'1234567' with the format `#  ##0` gives '1234 567'
`1, 2, 3, a, b, c, $, £, €` (and so on)	The alphanumeric character.	'705.15' with the format `$#.#0` gives '$705.15' or with the format #,#0 € gives '705,15 €' **Note** Alphanumeric characters should be delimited by single quotes, otherwise they can be interpreted as formatting characters. For example, # # will result in '123 4' while '#' # will result in '# 1234'
`COMPACT`	To round a numeric value and display it with an abbreviated suffix. The suffix is locale dependent.	-1234 with the format `COMPACT` gives -1K in "English (United States)" locale.

Format Token	Description	Example
`LONG_COMPACT`	To round a numeric value and display it with a suffix. The suffix is displayed in full name and is locale dependent.	-1234 with the format `COMPACT` gives -1 thousand in "English (United States)" locale.
`CURRENCY`	To display the value as a currency and apply the preferred viewing locale's rules for monetary values.	-1234 with the format `CURRENCY` gives -¤1,234.00 in "English (United States)" locale.
`ACCOUNTING_CURRENCY`	To display the value as a currency and apply the preferred viewing locale's rules for accounting values.	-1234 with the format `ACCOUNTING_CURRENCY` gives (¤1,234.00) in "English (United States)" locale.
`CURRENCY_COMPACT`	To display the value as a currency with an abbreviated suffix.	-1234 with the format `CURRENCY_COMPACT` gives -¤1K in "English (United States)" locale.
`[MIN_DEC:n]`	Used with COMPACT, LONG_COMPACT and CURRENCY_COMPACT to define the minimal number of decimals to display. Its value is 0 by default.	-1234 with the format `COMPACT[MIN_DEC:5]` gives -1.23400K in "English (United States)" locale.
`[MAX_DEC:n]`	Used with COMPACT, LONG_COMPACT and CURRENCY_COMPACT to define the maximum number of decimals to display. Its value is equal to MIN_DEC by default.	-1234 with the format `COMPACT[MAX_DEC:2]` gives -1.23K in "English (United States)" locale.
`[CURRENCY:c]`	Used with CURRENCY, ACCOUNTING_CURRENCY and CURRENCY_COMPACT to define the currency symbol. Its value is ¤ by default.	-1234 with the format `CURRENCY` gives -$1,234.00 in "English (United States)" locale.
`[Red], [Blue], [Green], [Yellow], [Gray], [White], [Dark Red], [Dark Blue], [Dark Green]`	The value in the specified color.	'150' with the format `#,##0[Red]` gives '150' in red text, `#,##0[Blue]` gives '150' in blue text.
Day/date tokens	`(day, date)`	
`d`	The number of the day in the month with no leading zeros. If the date for day is less than two characters, the date displays without a zero before it.	The first day of a month with the format `d` gives '1'
`dd`	The number of the day with leading zeros. If the date for day is less than two characters, the date displays with a zero before it.	The first day of a month with the format `dd` gives '01'
`ddd`	The name of the day abbreviated. The first letter is capitalized if the selected locale uses capitalized day names.	'Monday' with the format `ddd` gives 'Mon' in English, in French, lundi gives lun.
`Dddd`	Forced the capitalization of the day name, for any locale.	'Monday' with the format `Dddd` gives 'Mon' in English, in French, lundi gives Lun.

Format Token	Description	Example
dddd	The name of the day in full. The first letter is capitalized if the selected locale uses capitalized day names.	'Monday' with the format dddd gives 'Monday' in English. In French, the day is lundi.
DDDD	The name of the day in full, in uppercase.	'Monday' with the format DDDD gives 'MONDAY' in English. In French, the day is LUNDI.
dddd dd	The day of the week followed by a space and the number of the day.	'Monday' with the format dddd dd gives 'Monday 01'
Calendar tokens	(week, month, year)	
M	The number of the month with no leading zeros. If the number for month is less than two characters, the number displays without a zero before it.	'January' with the format M gives '1'
MM	The number of the month with leading zeros. If the number for month is less than two characters, the number displays with a zero before it.	'January' with the format MM gives '01'
mmm	The name of the month abbreviated. The first letter is capitalized if the selected locale uses capitalization.	'January' with the format mmm gives Jan in English. In French, this is 'jan'.
Mmmm	The name of the month abbreviated. The first letter is capitalized for all locales.	'January' with the format mmm gives Jan in English. In French, this is 'Jan'.
mmmm	The name of the month in full. The first letter is capitalized if the selected locale used capitalization.	'January' with the format mmmm gives January in English, janvier in French
MMMM	The name of the month in full all in uppercase.	'January' with the format MMMM gives JANUARY in English, JANVIER in French
ww	The week number of the year.	For the 9th of January 2015, the ww format gives '02', because it is the seventh week of the year 2015.
w	The week number of the year without leading zero.	For the 9th of January 2015, the w format gives '2', because it is the seventh week of the year 2015.
W	The week number of the month.	For the 9th of January 2015, the W format gives '2', because it is the second week of January.
yy	The last two digits for year.	'2003' with the format yy gives '03'
yyyy	All four digits for year.	'2003' with the format yyyy gives '2003'
Time tokens	(hours, minutes, seconds, am/pm)	
hh:mm:ss a	The hour with no leading zeros and the minutes and seconds with leading zeros. The "a" character displays AM or PM after the time when available.	'21:05:03' with the format hh:mm:ss a gives '9:05:03 PM' for English locale

Format Token	Description	Example
H	The hour according to the 24-hour clock, starting at 0. No leading zero for single figure hours.	'21:00' with the format H gives '21'. Possible values are 0-23.
HH	The hour according to the 24-hour clock, starting at 0.	'21:00' with the format HH gives '21'. Possible values are 00-23.
k	The hour according to the 24-hour clock, starting at 1. No leading zero for single figure hours.	'21:00' with the format k gives '21'. Possible values are 1-24.
kk	The hour according to the 24-hour clock, starting at 01.	'21:00' with the format kk gives '21'. Possible values are 01-24.
hh	The hour according to the 12-hour clock.	'21:00' with the format hh gives '09'
HH:mm	The hour and minutes with a zero in front of a single-digit hour.	'7:15 am' with the format HH:mm gives '07:15'
HH:mm:ss	The hour, minutes, and seconds with a zero in front of a single-digit hour.	'7:15 am' with the format HH:mm:ss gives '07:15:00'
mm:ss	The minutes, and seconds with a zero in front of a single-digit hour.	'07:15:03' with the format mm:ss gives '15:03'
x	Time zone in hours.	-08, +0530, +00
xx	Time zone in hours minutes.	-0800, +0530, +0000
xxx	Time zone in hours:minutes.	-08:00, +05:30, +00:00
xxxx	Time zone in hours minutes seconds.	-0800, +075228, +0000
xxxxx	Time zone in hours:minutes:seconds.	-08:00, +07:52:28, +00:00
X	Same as x, except that it displays "Z" when the time zone is UTC.	-08, +0530, Z
XX	Same as xx, except that it displays "Z" when the time zone is UTC.	-0800, +0530, Z
XXX	Same as xxx, except that it displays "Z" when the time zone is UTC	-08:00, +05:30, Z
XXXX	Same as xxxx, except that it displays "Z" when the time zone is UTC.	-0800, +075228, Z
XXXXX	Same as xxxxx, except that it displays "Z" when the time zone is UTC.	-08:00, +07:52:28, Z
VV	Time zone ID.	America/Los_Angeles
O	Time zone in hours from GMT.	GMT-8
OOOO	Time zone in hours and minutes from GMT (replaces the former 'z' format).	GMT-08:00

Format Token	Description	Example
z	The time zone name. If the time zone has no name, z displays the time difference.	CEST or PST. If the zone has a name. If there's no name, z displays the time difference as follows: +02, +530,... **Note** Documents created before the 4.3 release that use the previous z format are automatically converted so that the result displayed in 4.3 stays the same. The older z is interpreted as the 0000 listed in the table.
[TIMEZONE:t]	Used to specify the time zone of a date/time value (by default a date time in Web Intelligence is in UTC time zone). The supported time zones are listed below.	For the 1st of January 2015 12:00:00 AM, HH':'mm':'ss[TIMEZONE:US/Eastern] z gives 19:00:00 EST

List of time zones with the [TIMEZONE:t] token:

Africa/Abidjan	America/Grand_Turk	Asia/Baghdad	Australia/Perth	Europe/Ulyanovsk
Africa/Accra	America/Grenada	Asia/Bahrain	Australia/Queensland	Europe/Uzhgorod
Africa/Addis_Ababa	America/Guadeloupe	Asia/Baku	Australia/South	Europe/Vaduz
Africa/Algiers	America/Guatemala	Asia/Bangkok	Australia/Sydney	Europe/Vatican
Africa/Asmara	America/Guayaquil	Asia/Barnaul	Australia/Tasmania	Europe/Vienna
Africa/Asmera	America/Guyana	Asia/Beirut	Australia/Victoria	Europe/Vilnius
Africa/Bamako	America/Halifax	Asia/Bishkek	Australia/West	Europe/Volgograd
Africa/Bangui	America/Havana	Asia/Brunei	Australia/Yancowinna	Europe/Warsaw
Africa/Banjul	America/Hermosillo	Asia/Calcutta	Brazil/Acre	Europe/Zagreb
Africa/Bissau	America/Indiana/Indianapolis	Asia/Chita	Brazil/DeNoronha	Europe/Zaporozhye
Africa/Blantyre	America/Indiana/Knox	Asia/Choibalsan	Brazil/East	Europe/Zurich
Africa/Brazzaville	America/Indiana/Marengo	Asia/Chongqing	Brazil/West	GB
Africa/Bujumbura	America/Indiana/Petersburg	Asia/Chungking	CET	GB-Eire
Africa/Cairo	America/Indiana/Tell_City	Asia/Colombo	CST6CDT	GMT
Africa/Casablanca	America/Indiana/Vevay	Asia/Dacca	Canada/Atlantic	GMT+0
Africa/Ceuta	America/Indiana/Vincennes	Asia/Damascus	Canada/Central	GMT-0

Africa/Conakry	America/Indiana/Winamac	Asia/Dhaka	Canada/Eastern	GMT0
Africa/Dakar	America/Indianapolis	Asia/Dili	Canada/Mountain	Greenwich
Africa/Dar_es_Salaam	America/Inuvik	Asia/Dubai	Canada/Newfoundland	HST
Africa/Djibouti	America/Iqaluit	Asia/Dushanbe	Canada/Pacific	Hongkong
Africa/Douala	America/Jamaica	Asia/Famagusta	Canada/Saskatchewan	Iceland
Africa/El_Aaiun	America/Jujuy	Asia/Gaza	Canada/Yukon	Indian/Antananarivo
Africa/Freetown	America/Juneau	Asia/Harbin	Chile/Continental	Indian/Chagos
Africa/Gaborone	America/Kentucky/Louisville	Asia/Hebron	Chile/EasterIsland	Indian/Christmas
Africa/Harare	America/Kentucky/Monticello	Asia/Ho_Chi_Minh	Cuba	Indian/Cocos
Africa/Johannesburg	America/Knox_IN	Asia/Hong_Kong	EET	Indian/Comoro
Africa/Juba	America/Kralendijk	Asia/Hovd	EST	Indian/Mahe
Africa/Kampala	America/La_Paz	Asia/Irkutsk	EST5EDT	Indian/Maldives
Africa/Khartoum	America/Lima	Asia/Istanbul	Egypt	Indian/Mauritius
Africa/Kigali	America/Los_Angeles	Asia/Jakarta	Eire	Indian/Mayotte
Africa/Kinshasa	America/Louisville	Asia/Jayapura	Etc/GMT	Indian/Reunion
Africa/Lagos	America/Lower_Princes	Asia/Jerusalem	Etc/GMT+0	Iran
Africa/Libreville	America/Maceio	Asia/Kabul	Etc/GMT+1	Israel
Africa/Lome	America/Managua	Asia/Kamchatka	Etc/GMT+10	Jamaica
Africa/Luanda	America/Manaus	Asia/Karachi	Etc/GMT+11	Japan
Africa/Lubumbashi	America/Marigot	Asia/Kashgar	Etc/GMT+12	Kwajalein
Africa/Lusaka	America/Martinique	Asia/Kathmandu	Etc/GMT+2	Libya
Africa/Malabo	America/Matamoros	Asia/Katmandu	Etc/GMT+3	MET
Africa/Maputo	America/Mazatlan	Asia/Khandyga	Etc/GMT+4	MST
Africa/Maseru	America/Mendoza	Asia/Kolkata	Etc/GMT+5	MST7MDT
Africa/Mbabane	America/Menominee	Asia/Krasnoyarsk	Etc/GMT+6	Mexico/BajaNorte
Africa/Mogadishu	America/Merida	Asia/Kuala_Lumpur	Etc/GMT+7	Mexico/BajaSur
Africa/Monrovia	America/Metlakatla	Asia/Kuching	Etc/GMT+8	Mexico/General
Africa/Nairobi	America/Mexico_City	Asia/Kuwait	Etc/GMT+9	NZ
Africa/Ndjamena	America/Miquelon	Asia/Macao	Etc/GMT-0	NZ-CHAT
Africa/Niamey	America/Moncton	Asia/Macau	Etc/GMT-1	Navajo
Africa/Nouakchott	America/Monterrey	Asia/Magadan	Etc/GMT-10	PRC
Africa/Ouagadougou	America/Montevideo	Asia/Makassar	Etc/GMT-11	PST8PDT

Africa/Porto-Novo	America/Montreal	Asia/Manila	Etc/GMT-12	Pacific/Apia
Africa/Sao_Tome	America/Montserrat	Asia/Muscat	Etc/GMT-13	Pacific/Auckland
Africa/Timbuktu	America/Nassau	Asia/Nicosia	Etc/GMT-14	Pacific/Bougainville
Africa/Tripoli	America/New_York	Asia/Novokuznetsk	Etc/GMT-2	Pacific/Chatham
Africa/Tunis	America/Nipigon	Asia/Novosibirsk	Etc/GMT-3	Pacific/Chuuk
Africa/Windhoek	America/Nome	Asia/Omsk	Etc/GMT-4	Pacific/Easter
America/Adak	America/Noronha	Asia/Oral	Etc/GMT-5	Pacific/Efate
America/Anchorage	America/North_Dakota/Beulah	Asia/Phnom_Penh	Etc/GMT-6	Pacific/Enderbury
America/Anguilla	America/North_Dakota/Center	Asia/Pontianak	Etc/GMT-7	Pacific/Fakaofo
America/Antigua	America/North_Dakota/New_Salem	Asia/Pyongyang	Etc/GMT-8	Pacific/Fiji
America/Araguaina	America/Ojinaga	Asia/Qatar	Etc/GMT-9	Pacific/Funafuti
America/Argentina/Buenos_Aires	America/Panama	Asia/Qostanay	Etc/GMT0	Pacific/Galapagos
America/Argentina/Catamarca	America/Pangnirtung	Asia/Qyzylorda	Etc/Greenwich	Pacific/Gambier
America/Argentina/ComodRivadavia	America/Paramaribo	Asia/Rangoon30m	Etc/UCT	Pacific/Guadalcanal
America/Argentina/Cordoba	America/Phoenix	Asia/Riyadh	Etc/UTC	Pacific/Guam
America/Argentina/Jujuy	America/Port-au-Prince	Asia/Saigon	Etc/Universal	Pacific/Honolulu
America/Argentina/La_Rioja	America/Port_of_Spain	Asia/Sakhalin	Etc/Zulu	Pacific/Johnston
America/Argentina/Mendoza	America/Porto_Acre	Asia/Samarkand	Europe/Amsterdam	Pacific/Kiritimati
America/Argentina/Rio_Gallegos	America/Porto_Velho	Asia/Seoul	Europe/Andorra	Pacific/Kosrae
America/Argentina/Salta	America/Puerto_Rico	Asia/Shanghai	Europe/Astrakhan	Pacific/Kwajalein
America/Argentina/San_Juan	America/Punta_Arenas	Asia/Singapore	Europe/Athens	Pacific/Majuro
America/Argentina/San_Luis	America/Rainy_River	Asia/Srednekolymsk	Europe/Belfast	Pacific/Marquesas
America/Argentina/Tucuman	America/Rankin_Inlet	Asia/Taipei	Europe/Belgrade	Pacific/Midway
America/Argentina/Ushuaia	America/Recife	Asia/Tashkent	Europe/Berlin	Pacific/Nauru
America/Aruba	America/Regina	Asia/Tbilisi	Europe/Bratislava	Pacific/Niue

America/Asuncion	America/Resolute	Asia/Tehran	Europe/Brussels	Pacific/Norfolk
America/Atikokan	America/Rio_Branco	Asia/Tel_Aviv	Europe/Bucharest	Pacific/Noumea
America/Atka	America/Rosario	Asia/Thimbu	Europe/Budapest	Pacific/Pago_Pago
America/Bahia	America/Santa_Isabel	Asia/Thimphu	Europe/Busingen	Pacific/Palau
America/Bahia_Banderas	America/Santarem	Asia/Tokyo	Europe/Chisinau	Pacific/Pitcairn
America/Barbados	America/Santiago	Asia/Tomsk	Europe/Copenhagen	Pacific/Pohnpei
America/Belem	America/Santo_Domingo	Asia/Ujung_Pandang	Europe/Dublin	Pacific/Ponape
America/Belize	America/Sao_Paulo	Asia/Ulaanbaatar	Europe/Gibraltar	Pacific/Port_Moresby
America/Blanc-Sablon	America/Scoresbysund	Asia/Ulan_Bator	Europe/Guernsey	Pacific/Rarotonga
America/Boa_Vista	America/Shiprock	Asia/Urumqi	Europe/Helsinki	Pacific/Saipan
America/Bogota	America/Sitka	Asia/Ust-Nera	Europe/Isle_of_Man	Pacific/Samoa
America/Boise	America/St_Barthelemy	Asia/Vientiane	Europe/Istanbul	Pacific/Tahiti
America/Buenos_Aires	America/St_Johns	Asia/Vladivostok	Europe/Jersey	Pacific/Tarawa
America/Cambridge_Bay	America/St_Kitts	Asia/Yakutsk	Europe/Kaliningrad	Pacific/Tongatapu
America/Campo_Grande	America/St_Lucia	Asia/Yangon	Europe/Kiev	Pacific/Truk
America/Cancun	America/St_Thomas	Asia/Yekaterinburg	Europe/Kirov	Pacific/Wake
America/Caracas	America/St_Vincent	Asia/Yerevan	Europe/Lisbon	Pacific/Wallis
America/Catamarca	America/Swift_Current	Atlantic/Azores	Europe/Ljubljana	Pacific/Yap
America/Cayenne	America/Tegucigalpa	Atlantic/Bermuda	Europe/London	Poland
America/Cayman	America/Thule	Atlantic/Canary	Europe/Luxembourg	Portugal
America/Chicago	America/Thunder_Bay	Atlantic/Cape_Verde	Europe/Madrid	ROC
America/Chihuahua	America/Tijuana	Atlantic/Faeroe	Europe/Malta	ROK
America/Coral_Harbour	America/Toronto	Atlantic/Faroe	Europe/Mariehamn	Singapore
America/Cordoba	America/Tortola	Atlantic/Jan_Mayen	Europe/Minsk	Turkey
America/Costa_Rica	America/Vancouver	Atlantic/Madeira	Europe/Monaco	UCT
America/Creston	America/Virgin	Atlantic/Reykjavik	Europe/Moscow	US/Alaska
America/Cuiaba	America/Whitehorse	Atlantic/South_Georgia	Europe/Nicosia	US/Aleutian
America/Curacao	America/Winnipeg	Atlantic/St_Helena	Europe/Oslo	US/Arizona
America/Danmarkshavn	America/Yakutat	Atlantic/Stanley	Europe/Paris	US/Central

America/Dawson	America/Yellowknife	Australia/ACT	Europe/Podgorica	US/East-Indiana
America/Daw-son_Creek	Antarctica/Macquarie	Australia/Adelaide	Europe/Prague	US/Eastern
America/Denver	Antarctica/McMurdo	Australia/Brisbane	Europe/Riga	US/Hawaii
America/Detroit	Antarctica/Palmer	Australia/Broken_Hill	Europe/Rome	US/Indiana-Starke
America/Dominica	Antarctica/South_Pole	Australia/Canberra	Europe/Samara	US/Michigan
America/Edmonton	Arctic/Longyearbyen	Australia/Currie	Europe/San_Marino	US/Mountain
America/Eirunepe	Asia/Aden	Australia/Darwin	Europe/Sarajevo	US/Pacific
America/El_Salvador	Asia/Almaty	Australia/Eucla	Europe/Saratov	US/Pacific-New
America/Ensenada	Asia/Amman	Australia/Hobart	Europe/Simferopol	US/Samoa
America/Fort_Nelson	Asia/Anadyr	Australia/LHI	Europe/Skopje	UTC
America/Fort_Wayne	Asia/Aqtau	Australia/Lindeman	Europe/Sofia	Universal
America/Fortaleza	Asia/Aqtobe	Australia/Lord_Howe	Europe/Stockholm	W-SU
America/Glace_Bay	Asia/Ashgabat	Australia/Melbourne	Europe/Tallinn	WET
America/Godthab	Asia/Ashkhabad	Australia/NSW	Europe/Tirane	Zulu
America/Goose_Bay	Asia/Atyrau	Australia/North	Europe/Tiraspol	

1.6.1.2 Aggregate functions

1.6.1.2.1 Aggregate

Description

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

Function Group

Aggregate

Syntax

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

Input

Parameter	Description	Type	Required
measure	Any measure	Measure	Yes
member_set	The member set used to calculate the aggregation	Member set	No

Notes

- You can use extended syntax context operators with `Aggregate`.
- If you include `member_set`, `Aggregate` returns the aggregate value of the measure for all members in the member set.
- `member_set` can include multiple sets separated by semicolons (;).
- The list of member sets must be enclosed in {}.
- If the member set expression does not specify a precise member or node, the hierarchy referenced must be present in the table, then the member set expression references the current member in the hierarchy in the table. If the hierarchy is not in the table, the function returns the message #MULTIVALUE.
- Delegated measure aggregation returns #TOREFRESH when the required aggregation is not available in the query. The user has to refresh the document to get the new level of aggregation. This occurs for example when using the filter bar when the user selects a value before "all values" and vice versa when selecting "all values" before a selected value.

Examples

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

Related Information

Referring to members and member sets in hierarchies [page 284]

1.6.1.2.2 Member selection in aggregate functions

Description

For certain aggregate functions you can define a member selection to define the aggregation context when the block contains a hierarchy.

Function Group

Aggregate

Syntax

```
=AggregationFunction([my object];{memberselection})
```

Input

Parameter	Description	Type	Required
AggregationFunction	Must be one of the following: • Aggregate • Average • Coun • Ma • Mi • Sum	Aggregate function	Yes
my object	Dimension or a measure	Dimension or Measure	Yes
memberselection	A defined member, or a calculated set of member using set functions. The memberselection must be enclosed in curly brackets. Each part of the member set is separated with semicolon `{[member one];[member two];CalculatedMemberSet()}` Where CalculatedMemberSet uses one of the set functions:	Member or a calculated set of members using Set functions.	Yes

Parameter	Description	Type	Required
	• Ancestor • Descendant • Lag • Children • Parent • Siblings		

Description

The set functions use Object, Level, or Member as parameters. If you specify only Object and the object is a hierarchical object present in the block, then it will use the current member. You can also define a specific member using the following syntax :

```
[HierarchicalOBject]&[RootMember].[ChildMember].[ChildMember]
```

For Microsoft and Essbase .UNX sources you can select select a Level :

```
[HierarchicalOBject].[LevelName]
```

Examples

The following examples are all taken from an English language data source.

Example

In the following sample, you want to get the internet sales difference between Year 2002 and Year 2001:

```
=Sum([Internet Sales].[Internet Sales Amount];{[Calendar].
[Date.Calendar]&[All Periods].[CY 2002]}) + Sum([Internet Sales].[Internet
Sales Amount];{[Calendar].[Date.Calendar]&[All Periods].[CY 2001]})
```

Or either select two members in the member selection :

```
=Sum([Internet Sales].[Internet Sales Amount];{[Calendar].
[Date.Calendar]&[All Periods].[CY 2002]};[Calendar].[Date.Calendar]&[All
Periods].[CY 2001]})
```

Date.Calendar	Internet Sales Amount
All Periods	29,358,677.22
CY 2001	3,266,373.66
CY 2002	6,530,343.53
CY 2003	9,791,060.3
CY 2004	9,770,899.74

{CY 2001;CY 2002}
9,796,717.18

Example

In the following sample, you have a product hierarchy, and you want to know the internet sales for all products related to bikes. But two of them are in a different branch:

```
=Sum([Query 3].[Internet Sales].[Internet Sales Amount];{[Product Model
Categories]&[All Products].[Accessories].[Bike Racks];[Product Model
Categories]&[All Products].[Accessories].[Bike Stands];[Product Model
Categories]&[All Products].[Bikes]})
```

Bikes Amount	28,397,095.65
⊟ All Products	29,358,677.22
⊟ Accessories	700,759.96
⊞ Bike Racks	39,360
⊞ Bike Stands	39,591
⊞ Bottles and Cages	56,798.19
⊞ Cleaners	7,218.6
⊞ Fenders	46,619.58
⊞ Helmets	225,335.6
⊞ Hydration Packs	40,307.67
⊞ Tires and Tubes	245,529.32
⊟ Bikes	28,318,144.65
⊞ Mountain Bikes	9,952,759.56
⊞ Road Bikes	14,520,584.04
⊞ Touring Bikes	3,844,801.05
⊟ Clothing	339,772.61
⊞ Caps	19,688.1
⊞ Gloves	35,020.7
⊞ Jerseys	172,950.68
⊞ Shorts	71,319.81
⊞ Socks	5,106.32
⊞ Vests	35,687

Example

In the following sample, you want to compare Internet Sales Amount between North America Area countries, comparing first of all Canada and USA, and then with other countrys worldwide:

Firstly, get the total for the North American countries, for this sample, you are only interested in Canada and USA :

```
=Sum([Query 2].[Internet Sales].[Internet Sales Amount];{[Customer Geography]&[All Customers].[Canada];[Customer Geography]&[All Customers].[United States]})
```

Customer Geography		Internet Sales Amount
All Customers	11,367,634.37	29,358,677.22
Australia	11,367,634.37	9,061,000.58
Canada	11,367,634.37	1,977,844.86
France	11,367,634.37	2,644,017.71
Germany	11,367,634.37	2,894,312.34
United Kingdom	11,367,634.37	3,391,712.21
United States	11,367,634.37	9,389,789.51

Then you want to compare all countries with North America :

```
=[Query 2].[Internet Sales].[Internet Sales Amount] / Sum([Query 2].[Internet Sales].[Internet Sales Amount];{[Customer Geography]&[All Customers].[Canada];[Customer Geography]&[All Customers].[United States]})
```

Customer Geography		Internet Sales Amount
All Customers	258.27%	29,358,677.22
Australia	79.71%	9,061,000.58
Canada	17.40%	1,977,844.86
France	23.26%	2,644,017.71
Germany	25.46%	2,894,312.34
United Kingdom	29.84%	3,391,712.21
United States	82.60%	9,389,789.51

We can see that the global world total of customers is two and a half times that of North America, and that Australia is 80% compared to North America.

Related Information

Aggregate [page 41]

1.6.1.2.3 Average

Description

Returns the average value of a measure

Function Group

Aggregate

Syntax

```
num Average(measure[;member_set][;IncludeEmpty])
```

Input

Parameter	Description	Type	Required
measure	Any measure	Measure	Yes
member_set	A set of members	Member set	No
IncludeEmpty	Includes empty rows in the calculation	Keyword	No (Empty rows excluded by default)

Notes

- You can use extended syntax context operators with `Average`.
- If you include `member_set`, `Average` returns the average value of the measure for all members in the member set.
- `member_set` can include multiple sets separated by semicolons (;).
- The list of member sets must be enclosed in {}.
- If the member set expression does not specify a precise member or node, the hierarchy referenced must be present in the table, then the member set expression references the current member in the hierarchy in the table. If the hierarchy is not in the table, the function returns the message #MULTIVALUE.
- Delegated measure aggregation returns #TOREFRESH when the required aggregation is not available in the query. The user has to refresh the document to get the new level of aggregation. This occurs for

example when using the filter bar when the user selects a value before “all values” and vice versa when selecting “all values” before a selected value.

- A delegated measure given against a group returns #UNAVAILABLE as it requires local aggregation (aggregation of the measure value of the grouped values).
  Even when you force local aggregaton on a delegated measure against an "if then else" formula or group value it will still return the #MULTIVALUE message.

Examples

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

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

Related Information

Referring to members and member sets in hierarchies [page 284]
IncludeEmpty operator [page 269]

1.6.1.2.4 Count

Description

Returns the number of values in a set of values

Function Group

Aggregate

Syntax

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

Input

Parameter	Description	Type	Required
aggregated_data	Any dimension, measure, hierarchy, level or member set	Dimension, measure, hierarchy, member set	Yes
member_set	The member set used to calculate the count	Member set	No
IncludeEmpty	Includes empty values in the calculation	Keyword	No
Distinct|All	Includes distinct values only (default for dimensions) or all values (default for measures) in the calculation	Keyword	No

Notes

- You can use extended syntax context operators with `Count`.
- If you specify `IncludeEmpty` as the second argument, the function takes empty (null) values into consideration in the calculation.
- If you do not specify the `Distinct|All` parameter, the default values are `Distinct` for dimensions and `All` for measures.
- If you include `member_set`, `Count` restricts the count to the number of values in `member_set`.
- `member_set` can include multiple sets separated by semicolons (;).
- The list of member sets must be enclosed in {}.
- If the member set expression does not specify a precise member or node, the hierarchy referenced must be present in the table, then the member set expression references the current member in the hierarchy in the table. If the hierarchy is not in the table, the function returns the message #MULTIVALUE.
- Delegated measure aggregation returns #TOREFRESH when the required aggregation is not available in the query. The user has to refresh the document to get the new level of aggregation. This occurs for example when using the filter bar when the user selects a value before *All values* and vice versa when selecting *All values* before a selected value.
- A delegated measure given against a group returns #UNAVAILABLE as it requires local aggregation (aggregation of the measure value of the grouped values).
  Even when you force local aggregaton on a delegated measure against an "if then else" formula or group value it will still return the #MULTIVALUE message.

> **Note**
>
> In very specific workflows, merged object values don't aggregate the same way in XI 3.x and 4.x.
>
> In XI 3.x, the aggregation of the values of merged object members, in the context of that merged object (using the ForEach() formula), results in a filtered list of the aggregated values: those which match the merged values.
>
> In 4.x, the same workflow results in the full list of the aggregated values: no filter is applied.

Examples

`Count("Test")` returns 1

`Count([City];Distinct)` returns 5 if there are 5 different cities in a list of cities, even if there are more than 5 rows in the list due to duplication.

`Count([City];All)` returns 10 if there are 10 cities in a list of cities, even though some are duplicated.

`Count ([City];IncludeEmpty)` returns 6 if there are 5 cities and one blank row in a list of cities.

`Count([Product];{[Geography]&[State]})` returns the total number of products at the [State] level in the [Geography] hierarchy.

Related Information

IncludeEmpty operator [page 269]
Distinct/All operators [page 268]

1.6.1.2.5 First

Description

Returns the first value in a data set

Function Group

Aggregate

Syntax

```
input_type First(dimension|measure)
```

Input

Parameter	Description	Type	Required
dimension|measure	Any dimension or measure	Dimension or measure	Yes

Notes

- When placed in a break footer, `First` returns the first value in the break.
- When placed in a table footer, `First` returns the first value in the table.
- When placed in a table body, the result of `First` is unpredictable and depends on the order of the data set in the data source.

Examples

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

1.6.1.2.6 Interpolation

Description

Calculates empty measure values by interpolation

Function Group

Aggregate

Syntax

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

Input

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

Notes

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

Examples

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

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

Related Information

Linear operator [page 270]
PointToPoint operator [page 271]

1.6.1.2.7 Last

Description

Returns the last value in a data set

Function Group

Aggregate

Syntax

```
input_type Last(dimension|measure)
```

Input

Parameter	Description	Type	Required
dimension|measure	Any dimension or measure	Dimension or measure	Yes

Notes

- When placed in a table footer, `Last` returns the last value in the in the break.
- When placed in a table footer, `Last` returns the last value in the table.
- When placed in a table body, the result of `Last` is unpredictable and depends on the order of the data set in the data source.
- For technical reasons, `Last` can return a null value when the input parameter is a merged object.

Examples

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

1.6.1.2.8 Max

Description

Returns the largest value in a set of values

Function Group

Aggregate

Syntax

```
input_type Max(aggregated_data[;member_set])
```

Input

Parameter	Description	Type	Required
aggregated_data	Any dimension, measure, hierarchy, level or member set	Dimension, measure, hierarchy, level or member set	Yes
member_set	A set of members	Member set	No

Notes

- You can use extended syntax context operators with `Max`.
- If you include `member_set`, `Max` returns the maximum value of the aggregated data for all members in the member set.
- `member_set` can include multiple sets separated by semicolons (;).
- The list of member sets must be enclosed in {}.
- If the member set expression does not specify a precise member or node, the hierarchy referenced must be present in the table, then the member set expression references the current member in the hierarchy in the table. If the hierarchy is not in the table, the function returns the message #MULTIVALUE.
- Delegated measure aggregation returns #TOREFRESH when the required aggregation is not available in the query. The user has to refresh the document to get the new level of aggregation. This occurs for example when using the filter bar when the user selects a value before “all values” and vice versa when selecting “all values” before a selected value.
- A delegated measure given against a group returns #UNAVAILABLE as it requires local aggregation (aggregation of the measure value of the grouped values).
  Even when you force local aggregaton on a delegated measure against an "if then else" formula or group value it will still return the #MULTIVALUE message.

Examples

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

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

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

1.6.1.2.9 Median

Description

Returns the median (middle value) of a measure

Function Group

Aggregate

Syntax

```
num Median(measure)
```

Input

Parameter	Description	Type	Required
measure	Any measure	Measure	Yes

Notes

If the set of numbers has an even number of values, `Median` takes the average of the middle two values.

Examples

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

1.6.1.2.10 Min

Description

Returns the smallest value in a set of values

Function Group

Aggregate

Syntax

```
input_type Min(aggregated_data[;member_set])
```

Input

Parameter	Description	Type	Required
aggregated_data	Any dimension, measure, hierarchy, level or member set	Dimension, measure, hierarchy, level or member set	Yes
member_set	A set of members	Member set	No

Notes

- You can use extended syntax context operators with `Min`.
- If you include `member_set`, `Min` returns the minimum value of the aggregated data for all members in the member set.
- `member_set` can include multiple sets separated by semicolons (;).
- The list of member sets must be enclosed in {}.
- If the member set expression does not specify a precise member or node, the hierarchy referenced must be present in the table, then the member set expression references the current member in the hierarchy in the table. If the hierarchy is not in the table, the function returns the message #MULTIVALUE.
- Delegated measure aggregation returns #TOREFRESH when the required aggregation is not available in the query. The user has to refresh the document to get the new level of aggregation. This occurs for example when using the filter bar when the user selects a value before “all values” and vice versa when selecting “all values” before a selected value.
- A delegated measure given against a group returns #UNAVAILABLE as it requires local aggregation (aggregation of the measure value of the grouped values).
  Even when you force local aggregaton on a delegated measure against an "if then else" formula or group value it will still return the #MULTIVALUE message.

Examples

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

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

`Min([Sales Revenue];{[Geography]&[US].children})` returns the lowest sales revenue for a US state if [US] is a member in the [Geography] hierarchy with levels [Country] > [State] > [City].

1.6.1.2.11 Mode

Description

Returns the most frequently-occurring value in a data set

Function Group

Aggregate

Syntax

```
input_type Mode(dimension|measure)
```

Input

Parameter	Description	Type	Required
dimension|measure	Any dimension or measure	Measure	Yes

Notes

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

Examples

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

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

1.6.1.2.12 Percentage

Description

Expresses a measure value as a percentage of its embedding context

Function Group

Aggregate

Syntax

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

Input

Parameter	Description	Type	Required
measure	Any measure	Measure	Yes
Break	Accounts for table breaks	Keyword	No
Row|Col	Sets the calculation direction	Keyword	No

Examples

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 % column has the formula `Percentage([Sales Revenue];Row)`

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

1.6.1.2.13 Percentile

Description

Returns the nth percentile of a measure

Function Group

Numeric

Syntax

```
num Percentile(measure;percentile)
```

Input

Parameter	Description	Type	Required
measure	Any measure	Measure	Yes
percentile	A percentage expressed as a decimal	Number	Yes

Notes

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

Examples

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

1.6.1.2.14 Product

Description

Multiplies the values of a measure

Function Group

Aggregate

Syntax

```
num Product(measure)
```

Input

Parameter	Description	Type	Required
measure	Any measure	Measure	Yes

Examples

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

1.6.1.2.15 RunningAverage

Description

Returns the running average of a measure

Function Group

Aggregate

Syntax

```
num RunningAverage(measure[;Row|Col][;IncludeEmpty][;(reset_dims)])
```

To reset at each section the RunningAverage, we recommend the following syntax:

```
num RunningAverage(measure;section)
```

Input

Parameter	Description	Type	Required
measure	Any measure	Measure	Yes
Row|Col	Sets the calculation direction	Keyword	No
IncludeEmpty	Includes empty values in the calculation	Keyword	No
reset_dims	Resets the calculation on the specified dimensions	Dimension list	No
section	Dimension on which the section is set	Keyword	Yes in the case of a section reset

Notes

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

Examples

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

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

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

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

In an example where you are using `RunningAverage` in a section on [Quarter], using the formula `RunningAverage([Sales revenue];([Quarter])),` you receive the following results:

Q1

City	Sales revenue	Running Average
New York	$1,987,114.70	$1,987,114.70
Houston	$1,544,627.80	$1,765,871.25
Los Angeles	$1,129,177.60	$1,553,640.03

Q2

City	Sales revenue	Running Average
New York	$2,028,090.70	$2,028,090.70
Houston	$1,380,838.20	$1,704,464.45
Los Angeles	$980,405.30	$1,463,111.40

Related Information

IncludeEmpty operator [page 269]
Row/Col operators [page 272]

1.6.1.2.16 RunningCount

Description

Returns the running count of a number set

Function Group

Aggregate

Syntax

```
num RunningCount(dimension|measure[;Row|Col][;IncludeEmpty][;(reset_dims)])
```

To reset at each section the RunningCount, we recommend the following syntax:

```
num RunningCount(dimension|measure;section)
```

Input

Parameter	Description	Type	Required
dimension|measure	Any dimension or measure	Dimension or measure	Yes
Row|Col	Sets the calculation direction	Keyword	No
IncludeEmpty	Includes empty values in the calculation	Keyword	No
reset_dims	Resets the calculation on the specified dimensions	Dimension list	No
section	Dimension on which the section is set	Keyword	Yes in the case of a section reset

Notes

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

Examples

`RunningCount([Sales revenue])` returns these results in the following table:

Country	*Resort*	**Sales revenue**	**Running Count**
US	Hawaiian Club	1,479,660	1

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

In an example where you are using `RunningCount` in a section on [Week], using the formula `RunningCount([Lines];([Week]))` and with an input control on [Sales revenue] limiting the list to revenues over $30,000, returns the following results:

Week 1

Lines	*Sales revenue*	*Running Count*
Sweat-T-Shirts	$186,191	1
Shirt Waist	$139,082	2
Dresses	$70,931	3

Week 2

Lines	*Sales revenue*	*Running Count*
Accessories	$344,617	1
Sweat-T-Shirts	$196,976	2
Shirt Waist	$105,597	3
Dresses	$76,290	4
Sweaters	$68,364	5

Notice that in Week 1 there are three lines with revenue that exceeded $30,000, while in Week 2, there are five product lines that exceeded $30,000.

Related Information

IncludeEmpty operator [page 269]
Row/Col operators [page 272]
IncludeEmpty operator [page 269]
IncludeEmpty operator [page 269]

1.6.1.2.17 RunningMax

Description

Returns the running maximum of a dimension or measure

Function Group

Aggregate

Syntax

```
input_type RunningMax(dimension|measure[;Row|Col][;(reset_dims)])
```

To reset at each section the RunningMax, we recommend the following syntax:

```
num RunningMax(measure;section)
```

Input

Parameter	Description	Type	Reqiured
dimension|measure	Any dimension or measure	Dimension or measure	Yes
Row|Col	Sets the calculation direction	Keyword	No
reset_dims	Resets the calculation on the specified dimensions	Dimension list	No
section	Dimension on which the section is set	Keyword	Yes in the case of a section reset

Notes

- You can use extended syntax context operators with `RunningMax`.
- You can set the calculation direction with the `Row` and `Col` operators.
- If you apply a sort on the measure referenced by `RunningMax`, the running maximum is calculated after the measure is sorted .
- You must always place dimensions in parentheses even if there is only one dimension in the list of reset dimensions.

- When you specify a set of reset dimensions you must separate them with semi-colons.
- `RunningMax` does not automatically reset the max after a block break or new section.

Examples

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

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

In an example where you are using `RunningMax` in a section on [City], using the formula `RunningMax([Sales revenue];([City]))`, you receive the following results:

Austin		
Quarter	*Sales revenue*	*Running Max*
Q1	$775,482.70	$775,482.70
Q2	$667,850.30	$775,482.70
Q3	$581,470.40	$775,482.70
Q4	$674,869.80	$775,482.70

Boston		
Quarter	*Sales revenue*	*Running Max*
Q1	$312,896.40	$312,896.40
Q2	$291,431.00	$312,896.40
Q3	$249,529.00	$312,896.40
Q4	$429,850.20	$429,850.20

Related Information

IncludeEmpty operator [page 269]
Row/Col operators [page 272]

1.6.1.2.18 RunningMin

Description

Returns the running minimum of a dimension or measure

Function Group

Aggregate

Syntax

```
input_type RunningMin(dimension|measure;[Row|Col];[(reset_dims)])
```

To reset at each section the RunningMin, we recommend the following syntax:

```
num RunningMin(measure;section)
```

Input

Parameter	Description	Type	Required
dimension|detail|measure	Any dimension or measure	Dimension or measure	Yes
Row|Col	Sets teh calculation direction	Keyword	No
reset_dims	Resets the calculation on the specified dimensions	Dimension list	No
section	Dimension on which the section is set	Keyword	Yes in the case of a section reset

Notes

- You can use extended syntax context operators with `RunningMin`.
- You can set the calculation direction with the `Row` and `Col` operators.
- If you apply a sort on the measure referenced by `RunningMin`, the running minimum is calculated after the measure is sorted.
- You must always place dimensions in parentheses even if there is only one dimension in the list of reset dimensions.

- When you specify a set of reset dimensions you must separate them with semi-colons.
- `RunningMin` does not automatically reset the minimum after a block break or new section.

Examples

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

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

In an example where you are using `RunningMin` in a section on [City], using the formula `RunningMin([Sales revenue];([City]))`, you receive the following results:

Austin		
Quarter	*Sales revenue*	*Running Min*
Q1	$775,482.70	$775,482.70
Q2	$667,850.30	$667,850.30
Q3	$581,470.40	$581,470.40
Q4	$674,869.80	$581,470.40

Boston		
Quarter	*Sales revenue*	*Running Min*
Q1	$312,896.40	$312,896.40
Q2	$291,431.00	$291,431.00
Q3	$249,529.00	$249,529.00
Q4	$429,850.20	$249,529.00

Related Information

IncludeEmpty operator [page 269]
Row/Col operators [page 272]

1.6.1.2.19 RunningProduct

Description

Returns the running product of a measure

Function Group

Aggregate

Syntax

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

Input

Parameter	Description	Type	Required
measure	Any measure	Measure	Yes
Row|Col	Sets the calculation direction	Keyword	No
reset_dims	Resets the calculation on the specified dimensions	Dimension list	No

Notes

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

Examples

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

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

Related Information

IncludeEmpty operator [page 269]
Row/Col operators [page 272]

1.6.1.2.20 RunningSum

Description

Returns the running sum of a measure

Function Group

Aggregate

Syntax

```
num RunningSum(measure[;Row|Col][;(reset_dims)])
```

To reset at each section the RunningSum, we recommend the following syntax:

```
num RunningSum(measure;section)
```

Input

Parameter	Description	Type	Required
measure	Any measure	Measure	Yes
Row|Col	Sets the calculation direction	Keyword	No
reset_dims	Resets the calculation on the specified dimensions	Dimension list	No
section	Dimension on which the section is set	Keyword	Yes in the case of a section reset

Notes

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

Example

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

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

In an example where you are using `RunningSum` in a section on [Quarter], using the formula `RunningSum([Sales revenue];([Quarter]))`, you receive the following results:

Q1		
City	*Sales revenue*	*Running Sum*
New York	$1,987,114.70	$1,987,114.70
Houston	$1,544,627.80	$3,531,742.50
Los Angeles	$1,129,177.60	$4,660,920.10

Q2		
City	*Sales revenue*	*Running Sum*
New York	$2,028,090.70	$2,028,090.70
Houston	$1,380,838.20	$3,408,928.90
Los Angeles	$980,405.30	$4,389,334.20

Related Information

IncludeEmpty operator [page 269]
Row/Col operators [page 272]

1.6.1.2.21 StdDev

Description

Returns the standard deviation of a measure

Function Group

Aggregate

Syntax

```
num StdDev(measure)
```

Input

Paramter	Description	Type	Required
measure	Any measure	Measure	Yes

Notes

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

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

Examples

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

Related Information

Var [page 79]

1.6.1.2.22 StdDevP

Description

Returns the population standard deviation of a measure

Function Group

Aggregate

Syntax

```
num StdDevP(measure)
```

Input

Paramter	Description	Type	Required
measure	Any measure	Measure	Yes

Notes

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

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

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

Examples

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

1.6.1.2.23 Sum

Description

Returns the sum of a measure

Function Group

Aggregate

Syntax

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

Input

Parameter	Description	Type	Required
measure	Any measure	Measure	Yes
member_set	A set of members	Member set	No

Notes

- You can use extended syntax context operators with `Sum`.
- If you include `member_set`, `Sum` returns the sum of the measure for all members in the member set.
- `member_set` can include multiple sets separated by semicolons (;).
- The list of member sets must be enclosed in {}.
- If the member set expression does not specify a precise member or node, the hierarchy referenced must be present in the table, then the member set expression references the current member in the hierarchy in the table. If the hierarchy is not in the table, the function returns the message #MULTIVALUE.
- Delegated measure aggregation returns #TOREFRESH when the required aggregation is not available in the query. The user has to refresh the document to get the new level of aggregation. This occurs for example when using the filter bar when the user selects a value before “all values” and vice versa when selecting “all values” before a selected value.
- When migrating from XIR2 to XIR3, aggregation functions containing IN and WHERE clauses in XI2 queries should be included into Sum function definitely by using parenthesis as follows:
  In XIR2, the formula: `=Sum([Measure] In ([Dim 1];[Dim 2])) In ([Dim 1]) Where ([Dim 3]="Constant")`
  From XI3 onwards, modify the declaration: `=Sum(([Measure] In ([Dim 1];[Dim 2])) In ([Dim 1]) Where ([Dim 3]="Constant"))`
- A delegated measure given against a group returns #UNAVAILABLE as it requires local aggregation (aggregation of the measure value of the grouped values).
  Even when you force local aggregaton on a delegated measure against an "if then else" formula or group value it will still return the #MULTIVALUE message.

Examples

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

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

1.6.1.2.24 Var

Description

Returns the variance of a measure

Function Group

Aggregate

Syntax

```
num Var(measure)
```

Input

Paramter	Description	Type	Required
measure	Any measure	Measure	Yes

Notes

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

- finding the average of the set of numbers
- subtracting the average from each number in the set and squaring the difference
- summing all these squared differences

- dividing this sum by (`<number of numbers in the set>` - 1)

The variance is the square of the standard deviation.

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

Examples

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

Related Information

StdDev [page 75]

1.6.1.2.25 VarP

Description

Returns the population variance of a measure

Function Group

Aggregate

Syntax

```
num VarP(measure)
```

Input

Paramter	Description	Type	Required
measure	Any measure	Measure	Yes

Notes

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

- finding the average of the set of numbers
- subtracting the average from each number in the set and squaring the difference
- summing all these squared differences
- dividing this sum by (`<number of numbers in the set>`)

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

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

Examples

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

Related Information

StdDevP [page 76]

1.6.1.3 Character functions

1.6.1.3.1 Asc

Description

Returns the ASCII value of a character

Function Group

Character

Syntax

```
int Asc(string)
```

Input

Parameter	Description	Type	Required
string	Any string	String	Yes

Notes

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

Examples

`Asc("A")` returns 65.

`Asc("ab")` returns 97.

`Asc([Country])` returns 85 when the value of [Country] is "US".

1.6.1.3.2 Char

Description

Returns the character associated with an ASCII code

Function Group

Character

Syntax

```
string Char(ascii_code)
```

Input

Parameter	Description	Type	Required
ascii_code	An ASCII code	Number	Yes

Notes

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

Example

s

`Char(123)` returns "{".

1.6.1.3.3 Concatenation

Description

Concatenates (joins) two character strings. With numbers, the function will sum up the values rather than concatenate them.

> **Note**
>
> If at least one of the input parameters is a string, then all other input parameters are converted into strings.

Function Group

Character

Syntax

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

Input

Parameter	Description	Type	Required
first_string	The first string	String or number	Yes
second_string	The string added to the first string	String or number	Yes

Notes

You can also use the '+' operator to concatenate strings.

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

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

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

Examples

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

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

If `[A]` is a number and `[A]` = `1`, `Concatenation([A];[A])` returns "2".

If `[A]` is a string and `[A]` = `1`, `Concatenation([A];[A])` returns "11".

If `[A]` is a string, `[B]` is a number, `[A]` = `1` and `[B]` = `2`, `Concatenation([A];[B])` returns "12".

1.6.1.3.4 Fill

Description

Builds a string by repeating a string n times

Function Group

Character

Syntax

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

Input

Parameter	Description	Type	Required
repeating_string	The repeating string	String	Yes
num_repeats	The number of repeats	Number	Yes

Examples

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

1.6.1.3.5 FormatDate

Description

Formats a date according to a specified format

Function Group

Character

Syntax

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

Input

Parameter	Description	Type	Required
date	The date to format	Date	Yes
format_string	The format to apply	String	Yes

Notes

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

Examples

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

Related Information

Custom formats [page 33]

1.6.1.3.5.1 Format_string examples for the FormatDate function

In the `FormatDate` syntax for `format_string`, you can use the examples in the following table.

> **Note**
>
> You can find these samples in the *Format Number* dialog box in the Rich Client; however what samples appear depend on your selected Product Locale in the BI launch pad preferences. For example, if you select *English*, then “September 21, 2004” will be an available sample.

Sample	Syntax
Tuesday, September 21, 2004	dddd', 'mmmm d', 'yyyy
September 21, 2004	mmmm d', 'yyyy
Sep 21, 2004	mmm d', 'yyyy

Sample	Syntax
9/21/04	M'/'d'/'yy
Sep 21, 2004 8:45:30 PM	mmm d', 'yyyy h':'mm':'ss a
9/21/04 8:45 PM	M'/'d'/'yy h':'mm a
9/21/2004	M'/'d'/'yyyy
09/21/2004	MM'/'d'/'yyyy
9/21/04 8:45:30 PM	M'/'d'/'yy h':'mm a
8:45:30 PM	h':'mm':'ss a
8:45 PM	h':'mm a
20:45:30	HH':'mm':'ss
20h45	HH'h'mm

→ Tip

We recommend that you represent actual text in the syntax surrounded by apostrophes so that the text is not confused as pattern symbols. For example, as in the last sample in the table above, 'h' in "HH'h'mm".

Related Information

FormatDate [page 85]
Custom formats [page 33]

1.6.1.3.6 FormatNumber

Description

Formats a number according to a specified format

Function Group

Character

Syntax

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

Input

Parameter	Description	Type	Required
number	The number to format	Number	Yes
format_string	The format to apply	String	Yes

Notes

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

Examples

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

Related Information

Custom formats [page 33]

1.6.1.3.7 HTMLEncode

Description

Applies HTML encoding rules to a string

Function Group

Character

Syntax

```
string HTMLEncode(html)
```

Input

Paramter	Description	Type	Required
html	An HTML string	String	Yes

Examples

`HTMLEncode("<Hello World!>")` returns "<Hello World!>", because the browser interprets the character. Internally, it returns "<Hello World!>".

1.6.1.3.8 InitCap

Description

Capitalizes the first letter of a string

Function Group

Character

Syntax

```
string InitCap(string)
```

Input

Parameter	Description	Type	Required
string	The string to capitalize	String	Yes

Examples

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

1.6.1.3.9 Left

Description

Returns the leftmost characters of a string.

> **Note**
>
> This function returns the first characters from the logical start of the string. Right-To-Left display/reading languages, such as Arabic or Hebrew for example, are supported.

Function Group

Character

Syntax

```
string Left(string;num_chars)
```

Input

Parameter	Description	Type	Required
string	The input string	string	Yes
num_chars	The number of characters to return from the start of the string	number	Yes

Example

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

1.6.1.3.10 LeftPad

Description

Pads a string to have a specified minimum length by adding a designated string to its left.

> **Note**
>
> This function pads the strings from the logical start. Right-To-Left display/reading languages, such as Arabic or Hebrew for example, are supported.

Function Group

Character

Syntax

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

Input

Parameter	Description	Type	Required
padded_string	The original string	String	Yes
length	The length of the output string	Number	Yes
left_string	The string to be added to the start of `padded_string`	String	Yes

Notes

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

Examples

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

`LeftPad("York";6;"New ")` returns "NeYork"

`LeftPad("York";11;"New ")` returns "New NewYork"

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

1.6.1.3.11 LeftTrim

Description

Trims the leading spaces and special characters in a string.

> **Note**
>
> This function removes the first characters from the logical start of the string. Right-To-Left display/reading languages, such as Arabic or Hebrew for example, are supported.

Function Group

Character

Syntax

```
string LeftTrim(trimmed_string[;char])
```

Input

Parameter	Description	Type	Required
trimmed_string	The string to be trimmed	String	Yes
string	The character to remove	String	No

Examples

- `LeftTrim([Product])` returns "Laptop" if [Product] is " Laptop".
- `LeftTrim([Product]; "=")` returns "Laptop" if [Product] is "==Laptop".

1.6.1.3.12 Length

Description

Returns the number of characters in a string

Function Group

Character

Syntax

```
int Length(string)
```

Input

Parameter	Description	Type	Required
string	The input string	String	Yes

Examples

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

1.6.1.3.13 Lower

Description

Converts a string to lower case

Function Group

Character

Syntax

```
string Lower(string)
```

Input

Parameter	Description	Type	Required
string	The string to be converted to lower case	String	Yes

Examples

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

1.6.1.3.14 Match

Description

Determines whether a string matches a pattern

Function Group

Character

Syntax

```
bool Match(test_string;pattern)
```

Input

Paramter	Description	Type	Required
test_string	The string to be tested against the text pattern	string	Yes
pattern	The text pattern	string	Yes

Notes

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

Examples

`Match([Country];"F*")` returns True if [Country] is "France".

`Match([Country];"?S?")` returns True if [Country] is "USA".

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

1.6.1.3.15 Pos

Description

Returns the starting position of a text pattern in a string.

Function Group

Character

Syntax

```
int Pos(test_string;pattern[;start][;end]])
```

Input

Parameter	Description	Type	Required
test_string	The string to be tested for the text pattern	string	Yes
pattern	The text pattern	string	Yes
start	The start position of the search of the string	integer	No
end	The end position of the search of the string	integer	No

Notes

- If the pattern occurs more than once, `Pos` returns the position of the first occurrence.
- The search is performed between the start and end position (included).

Examples

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

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

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

`Pos ("Hello World World"; "World"; 7)` returns 7.

`Pos ("Hello World World"; "World"; 8)` returns 13.

`Pos ("Hello World World"; "World"; 8; 13)` returns 13.

`Pos ("Hello World World"; "World"; 8; 10)` returns 0.

1.6.1.3.16 Replace

Description

Replaces part of a string with another string

Function Group

Character

Syntax

```
string Replace(replace_in;replaced_string;replace_with)
```

Input

Parameter	Description	Type	Required
replace_in	The string in which the text is replaced	string	Yes
replaced_string	The text to be replaced	string	Yes
replace_with	The text that replaces `replaced_string`	string	Yes

Examples

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

1.6.1.3.17 Reverse

Description

Returns the reverse of a string.

Function Group

Character

Syntax

```
string Reverse(string)
```

Input

Parameter	Description	Type	Required
string	The string to be reversed	String	Yes

Examples

`Reverse("abc123")` returns "`321cba`".

1.6.1.3.18 Right

Description

Returns the rightmost characters of a string (the characters at the end of the string).

> **Note**
>
> This function returns the first characters from the logical end of the string. Right-To-Left display/reading languages, such as Arabic or Hebrew for example, are supported.

Function Group

Character

Syntax

```
string Right(string;num_chars)
```

Input

Parameter	Description	Type	Required
string	Any string	string	Yes
num_chars	The number of characters to return from the right	number	Yes

Examples

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

1.6.1.3.19 RightPad

Description

Pads a string to have a specified minimum length by adding a designated string to its right end.

> **Note**
>
> This function pads the strings from the logical end. Right-To-Left display/reading languages, such as Arabic or Hebrew for example, are supported.

Function Group

Character

Syntax

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

Input

Parameter	Description	Type	Required
padded_string	The original string	String	Yes
length	The length of the output string	Number	Yes
right_string	The string to be added to the end of `padded_string`	String	Yes

Notes

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

Examples

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

1.6.1.3.20 RightTrim

Description

Trims the trailing spaces and special characters in a string.

> **Note**
>
> This function removes the last characters from the logical end of the string. Right-To-Left display/reading languages, such as Arabic or Hebrew for example, are supported.

Function Group

Character

Syntax

```
string RightTrim(trimmed_string[;char])
```

Input

Parameter	Description	Type	Required
trimmed_string	The string to be trimmed	String	Yes
string	The character to remove	String	No

Examples

- `RightTrim([Product])` returns "Laptop" if [Product] is "Laptop ".
- `RightTrim([Product]; "=")` returns "Laptop" if [Product] is "Laptop==".

1.6.1.3.21 RPos

Description

Returns the starting position of the last occurrence of a text pattern in a string.

Function Group

Character

Syntax

```
int RPos(test_string;pattern[;start][;end]])
```

Input

Parameter	Description	Type	Required
test_string	The string to be tested for the text pattern	string	Yes
pattern	The text pattern	string	Yes
start	The start position of the search of the string	integer	No
end	The end position of the search of the string	integer	No

Notes

The search is performed between the start and end position (included). The search is performed from the end of the string and is done backwards.

Examples

`RPos ("Hello World World"; "World")` returns 13.

`RPos ("Hello World World"; "World"; 7)` returns 13.

`RPos ("Hello World World"; "World"; 8)` returns 13.

`RPos ("Hello World World"; "World"; 8; 13)` returns 13.

`RPos ("Hello World World"; "World"; 1; 10)` returns 7.

`RPos ("Hello World World"; "World"; 1; 6)` returns 0.

1.6.1.3.22 Substr

Description

Returns part of a string

Function Group

Character

Syntax

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

Input

Parameter	Description	Type	Required
string	Any string	String	Yes
start	The start position of the extracted string	Number	Yes
length	The length of the extracted string	Number	Yes

Examples

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

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

1.6.1.3.23 Trim

Description

Removes leading spaces and trailing spaces from the string to scan. If another character is passed as a parameter then the character is removed.

Function Group

Character

Syntax

```
string Trim(trimmed_string[;char])
```

Input

Parameter	Description	Type	Required
string	The string to be trimmed	String	Yes
string	The character to remove	String	No

Examples

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

`Trim ( Trim ("--=-Hello= =--"; "-" ) ; "=")` returns "-Hello= ".

1.6.1.3.24 Upper

Description

Converts a string to upper case

Function Group

Character

Syntax

```
string Upper(string)
```

Input

Parameter	Description	Type	Required
string	The string to be converted	String	Yes

Examples

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

1.6.1.3.25 UrlEncode

Description

Applies URL encoding rules to a string

Function Group

Character

Syntax

```
string UrlEncode(html)
```

Input

Parameter	Description	Type	Required
html	The URL to be encoded	String	Yes

Examples

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

1.6.1.3.26 WordCap

Description

Capitalizes the first letter of all the words in a string

Function Group

Character

Syntax

```
string WordCap(string)
```

Input

Parameter	Description	Type	Required
string	The string to be capitalized	String	Yes

Examples

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

1.6.1.4 Date and Time functions

1.6.1.4.1 CurrentDate

Description

Returns the current date formatted according to the regional settings

Function Group

Date and Time

Syntax

```
date CurrentDate()
```

Examples

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

1.6.1.4.2 CurrentTime

Description

Returns the current time formatted according to the regional settings

Function Group

Date and Time

Syntax

```
time CurrentTime()
```

Examples

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

1.6.1.4.3 DatesBetween

Description

Returns the number of periods between two dates, irrespective of the time.

Function Group

Date and Time

Syntax

```
int DatesBetween(first_date;last_date;period)
```

Input

Parameter	Description	Type	Required
`first_date`	The first date of the time range	Date	Yes
`last_date`	The last date of the time range	Date	Yes
`period`	The type of period to be counted in the time range	Pre-defined	Yes

Note

- Possible values for the `period` parameter are: `DayPeriod`, `WeekPeriod`, `MonthPeriod`, `QuarterPeriod`, `SemesterPeriod`, `YearPeriod`.

  > ⚠ Caution
  >
  > When using the `WeekPeriod` parameter, the application doesn't consider a week to be seven days. A week difference can be anything from one to seven days depending on which day is used for the calculation. Also, Monday is defined as the first day of the week, as per the ISO 8601 standard, meaning that there is always a one week difference between a Monday and the Sunday before it.

- If the return value is out of range for int, an #OVERFLOW error is returned.

Examples

- `DatesBetween([Begin Date];[End Date];MonthPeriod)` returns 2 when `[Begin Date]` is 30 June 2016 and `[End Date]` is 3 August 2016.
- `DatesBetween([Begin Date];[End Date];DayPeriod)` returns -10 when `[Begin Date]` is 30 June 2016 and `[End Date]` is 20 June 2016.
- `DatesBetween([Begin Date];[End Date];QuarterPeriod)` returns 6 when `[Begin Date]` is 30 June 2016 and `[End Date]` is 17 November 2017.
- `DatesBetween([Begin Date];[End Date];MonthPeriod)` returns 1 when `[Begin Date]` is 31 December 2015 and `[End Date]` is 1 January 2016.
- `DatesBetween([Begin Date];[End Date];DayPeriod)` returns 1 when `[Begin Date]` is 31 December 2015 and `[End Date]` is 1 January 2016.
- `DatesBetween([Begin Date];[End Date];WeekPeriod)` returns 0 when `[Begin Date]` is 31 December 2015 and `[End Date]` is 1 January 2016, because both days belong to the same week.

1.6.1.4.4 DayName

Description

Returns the day name in a date

Function Group

Date and Time

Syntax

```
string DayName(date)
```

Input

Parameter	Description	Type	Required
date	The input date	Date	Yes

Examples

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

Note

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

1.6.1.4.5 DayNumberOfMonth

Description

Returns the day number in a month

Function Group

Date and Time

Syntax

```
int DayNumberOfMonth(date)
```

Input

Parameter	Description	Type	Required
date	The input date	Date	Yes

Examples

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

1.6.1.4.6 DayNumberOfWeek

Description

Returns the day number in a week

Function Group

Date and Time

Syntax

```
int DayNumberOfWeek(date)
```

Input

Parameter	Description	Type	Required
date	The input date	Date	Yes

Notes

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

Examples

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

1.6.1.4.7 DayNumberOfYear

Description

Returns the day number in a year

Function Group

Date and Time

Syntax

```
int DayNumberOfYear(date)
```

Input

Parameter	Description	Type	Required
date	The input date	Date	Yes

Examples

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

1.6.1.4.8 DaysBetween

Description

Returns the number of days between two dates

Function Group

Date and Time

Syntax

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

> ⓘ Note
>
> You must ensure that the dates given in the arguments are in the same timezone. This applies to all date operations: comparison and calculation.

Input

Parameter	Description	Type	Required
first_date	The first date	Date	Yes
last_date	The last date	Date	Yes

Examples

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

1.6.1.4.9 LastDayOfMonth

Description

Returns the date of the last day in a month

Function Group

Date and Time

Syntax

```
date LastDayOfMonth(date)
```

Input

Parameter	Description	Type	Required
date	Any date in the month	Date	Yes

Examples

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

1.6.1.4.10 LastDayOfWeek

Description

Returns the date of the last day in a week

Function Group

Date and Time

Syntax

```
date LastDayOfWeek(date)
```

Input

Parameter	Description	Type	Required
date	Any date in the week	Date	Yes

Notes

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

Examples

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

1.6.1.4.11 Month

Description

Returns the month name in a date

Function Group

Date and Time

Syntax

```
string Month(date)
```

Input

Parameter	Description	Type	Required
date	The input date	Date	Yes

Examples

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

1.6.1.4.12 MonthNumberOfYear

Description

Returns the month number in a date

Function Group

Date and Time

Syntax

```
int MonthNumberOfYear(date)
```

Input

Parameter	Description	Type	Required
date	Any date in the year	Date	Yes

Example

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

1.6.1.4.13 MonthsBetween

Description

Returns the number of months between two dates

Function Group

Date and Time

Syntax

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

Input

Parameter	Description	Type	Required
first_date	The first date	Date	Yes
last_date	The last date	Date	Yes

Examples

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

`MonthsBetween([Sale Date];[Invoice Date])` returns 1 if [Sale Date] is 31/03/2008 and [Invoice Date] is 30/04/2008.

`MonthsBetween([Sale Date];[Invoice Date])` returns 118 if [Sale Date] is 07/01/1993 and [Invoice Date] is 06/11/2002.

1.6.1.4.14 Quarter

Description

Returns the quarter number in a date

Function Group

Date and Time

Syntax

```
int Quarter(date)
```

Input

Parameter	Description	Type	Required
date	Any date in the quarter	Date	Yes

Examples

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

1.6.1.4.15 RelativeDate

Description

Returns a date relative to another date.

Function Group

Date and Time

Syntax

```
date RelativeDate(start_date;num;period)
```

Input

Parameter	Description	Type	Required
start_date	The start date	Date	Yes
num	The number of period units added to the start date	Number	Yes
period	The type of period added to the start date	Pre-defined	Optional

Notes

- The `num` parameter can be a constant, the numerical result of a function, a measure value or a numerical dimension value, and has to be an integer.
- The `num` parameter can be negative to return a date earlier than `start_date`.
- If omitted, the `period` parameter works with days `(DayPeriod)`.
- When adding or subtracting months (for `SemesterPeriod`, `QuarterPeriod` and `MonthPeriod`), if the day does not exist in the returned month, then the last day of the returned month must be used.
- Possible values for the period parameter are: `MillisecondPeriod`, `SecondPeriod`, `MinutePeriod`, `HourPeriod`, `DayPeriod`, `WeekPeriod`, `MonthPeriod`, `QuarterPeriod`, `SemesterPeriod`, `YearPeriod`.

Examples

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

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

`RelativeDate([Reservation Date];1;MonthPeriod)` returns 12 February 2007 when [Reservation date] is 12 January 2007.

1.6.1.4.16 TimeBetween

Description

Returns the number of periods between two dates, taking the time into account.

Function Group

Date and Time

Syntax

```
int TimeBetween(first_date;last_date;period)
```

> **Note**
>
> Make sure that the dates given in the arguments are in the same time zone, since no time zone offset is used in calculating the return value.

Input

Parameter	Description	Type	Required
`first_date`	The first date of the time range	Time	Yes
`last_date`	The last date of the time range	Time	Yes
`period`	The type of period to be counted in the time range	Pre-defined	Yes

Notes

- Possible values for the `period` parameter are: `DayPeriod`, `WeekPeriod`, `MonthPeriod`, `QuarterPeriod`, `SemesterPeriod`, `YearPeriod`, `HourPeriod`, `MinutePeriod`, `SecondPeriod`, `MillisecondPeriod`.
- If the return value is out of range for int, an #OVERFLOW error is returned.

Examples

- `TimeBetween([Begin Date];[End Date];HourPeriod)` returns 2 when `[Begin Date]` is 30 June 2016, 8:45 and `[End Date]` is 30 June 2016, 10:05.
- `TimeBetween([Begin Date];[End Date];MinutePeriod)` returns -10 when `[Begin Date]` is 30 June 2016 8:45 and `[End Date]` is 30 June 2016 8:35.
- `TimeBetween([Begin Date];[End Date];SecondPeriod)` returns 120 when `[Begin Date]` is 30 June 2016 8:45 and `[End Date]` is 30 June 2016 8:47.
- TimeBetween([Begin Date];[End Date];MonthPeriod) returns 1 when `[Begin Date]` is 31 December 2015 11:45 and `[End Date]` is 1 January 2016 8:47.
- `TimeBetween([Begin Date];[End Date];DayPeriod)` returns 1 when `[Begin Date]` is 31 December 2015 11:45 and `[End Date]` is 1 January 2016 8:47.
- `TimeBetween([Begin Date];[End Date];WeekPeriod)` returns 0 when `[Begin Date]` is 31 December 2015 11:45 and `[End Date]` is 1 January 2016 8:47, because both days belong to the same week.

1.6.1.4.17 TimeDim

Description

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

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

> **Note**
>
> You cannot use the TimeDim function to filter on formulas (for example in a filter, input-control, element-link, filter/drill bar). Instead you should directly filter on the underlaying date dimension.

Function Group

Date and Time

Syntax

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

Input

Parameter	Description	Type	Required
Date Type	The date object for the report, for example, `InvoiceDate.`	Date	Yes
Period Type	The period for the results, from the following values: • `DayPeriod` • `MonthPeriod` • `QuarterPeriod` • `YearPeriod` When no value is selected, the `DayPeriod` is used by default. This object should be a data provider object, it must be available from report objects, and cannot be a variable.	Pre-defined	Optional

Use the above function in conjunction with the following functions:

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

Example

The first table below contains data that concerns only certain dates. The query examples that follow show how the results are interpreted.

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

The following formula `DayName(TimeDim([Invoice Date] ; QuarterPeriod)` returns daily values from the above table.

Invoice Date	Revenue
1/3/00	31,607
1/8/00	31,244
4/1/00	
7/3/00	38,154

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

Invoice Date	Revenue
Q1	62,851
Q2	
Q3	38,154

1.6.1.4.18 ToDate

Description

Turns a character string into a date. Give the date format as the parameter to indicate to Web Intelligence how to convert the string into a date. The date format you provide must match the format of the date in the original string. Refer to the link below for the possible date formats.

Function Group

Date and Time

Syntax

```
date ToDate(date_string;format[;cutoff_year])
```

or

```
date ToDate(date_string;"INPUT_DATE_TIME"[;cutoff_year])
```

> **Note**
>
> In scenarios where the *Preferred viewing locale* can be different depending on the user, a fixed format (for a particular locale) is not appropriate. In this case use INPUT_DATE_TIME as shown in the second example above.

Input

Parameter	Description	Type	Required
date_string	The string to be interpreted as a date.	String	Yes
format	The date format used by the string. Use "`INPUT_DATE_TIME`" to use the format of the Preferred viewing locale.	String	Yes*
cutoff_year	Optional parameter to indicate the year used for cutoff. By default, this year is 2029.	Integer	No

* See the note above. Use the format or `INPUT_DATE_TIME` depending on your needs.

Examples

`ToDate("12/15/2002";"MM/dd/yyyy")` interprets “12” as a month number, “15” as a day number and “2002” as a year.

`ToDate("Dec/02";"Mmm/yy")` interprets “Dec” as an abbreviated month name and “02” as the two last digits of a year.

`ToDate("15-December-02";"dd-Mmmm-yy")` interprets “15” as a day number, “December” as a month and “02” as the last two digits of a year.

`ToDate("12/15/02 11:00:00";"INPUT_DATE_TIME")` interprets “12/15/02 11:00:00” in the format used by the *Preferred viewing locale* on the user's machine.

> → Tip
>
> Use four digits for the year argument to prevent confusion or unwanted results. For example, "07" could mean "1907" or "2007".

> ⓘ Note
>
> - With `INPUT_DATE_TIME`, both the date and time must be specified in the `date_string` input string.
> - If `date_string` cannot be interpreted as a valid date with the specified format, the `ToDate()` formula returns `#ERROR`.
> - The way a date is displayed in a cell depends on the chosen date format in that cell. For instance, if the chosen date format is `"MM/dd/yyyy"`, then `ToDate(“Dec/15/02”;”MMM/dd/yy”)` will be displayed as 12/15/2002.
> - If the year in `date_string` is two digits and if a `cutoff_year` is provided, then:
> - The current century is the one of this `cutoff_year`.

- The last two digits of this `cutoff_year` define the threshold to use the current century of this `cutoff_year` or the previous one.
- If the `cutoff_year` is strictly lower than 100, it returns an error message.

Related Information

Custom formats [page 33]

1.6.1.4.19 Week

Description

Returns the week number in the year

Function Group

Date and Time

Syntax

```
int Week(date)
```

Input

Parameter	Description	Type	Required
date	The input date	Date	Yes

Examples

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

1.6.1.4.20 Year

Description

Returns the year in a date

Function Group

Date and Time

Syntax

```
int Year(date)
```

Input

Parameter	Description	Type	Required
date	The input date	Date	Yes

Examples

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

1.6.1.5 Data Provider functions

1.6.1.5.1 Connection

Description

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

Function Group

Data Provider

Syntax

```
string Connection(dp)
```

Input

Parameter	Description	Type	Required
dp	The data provider	Data provider	Yes

Notes

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

1.6.1.5.2 DataProvider

Description

Returns the name of the query as defined by the user in the *Query Properties* dialog.

Function Group

Data Provider

Syntax

```
string DataProvider(obj)
```

```
string DataProvider(dp)
```

Input

Parameter	Description	Type	Required
obj	A report object	Report object	Yes
dp	A data provider	Data provider	Yes

Examples

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

> **Note**
>
> DataProvider requires an object name to return its data provider name. If you use another function as a parameter of DataProvider, for example a dimension variable, that doesn't give an object name, the DataProvider function will return an error.

1.6.1.5.3 DataProviderKeyDate

Description

Returns the keydate of a data provider

Function Group

Data Provider

Syntax

```
date DataProviderKeyDate(dp)
```

Input

Parameter	Description	Type	Required
dp	The data provider	Data provider	Yes

Notes

- You must enclose the name of the data provider in square brackets.
- The returned keydate is formatted according to the document locale.
- This function is only supported on data providers based on BEx queries having SAP variables of KEYDATE type.
- This function is only supported for legacy OLAP UNV universes on BW. It is not supported for the BEx Direct Access in Web Intelligence or for BEx UNX universes. This function has been deprecated as UNV universes are deprecated in 4.3.

Examples

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

1.6.1.5.4 DataProviderKeyDateCaption

Description

Returns the keydate caption of a data provider

Function Group

Data Provider

Syntax

```
string DataProviderKeyDateCaption(dp)
```

Input

Parameter	Description	Type	Required
dp	The data provider	Data provider	Yes

Notes

- You must enclose the name of the data provider in square brackets.
- This function is only supported on data providers based on BEx queries having SAP variables of KEYDATE type.
- This function is only supported for legacy OLAP UNV universes on BW. It is not supported for the BEx Direct Access in Web Intelligence or for BEx UNX universes. This function has been deprecated as UNV universes are deprecated in 4.3.

Examples

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

1.6.1.5.5 DataProviderSQL

Description

Returns the SQL generated by a data provider

Function Group

Data Provider

Syntax

```
string DataProviderSQL(dp)
```

Input

Parameter	Description	Type	Required
dp	The data provider	Data provider	Yes

Notes

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

Examples

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

1.6.1.5.6 DataProviderType

Description

Returns the type of a data provider

Function Group

Data Provider

Syntax

```
string DataProviderType(dp)
```

Input

Parameter	Description	Type	Required
dp	The data provider	Data provider	Yes

Notes

- `DataProviderType` returns the type of data provider. Possible returned values are: `Universe`, `Web Intelligence`, `Excel`, `Text`, `Free-hand SQL`, `SAP HANA`, `SAP BW`, or `Web Service`.
- You must enclose the name of the data provider in square brackets.

Examples

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

1.6.1.5.7 DataSourceDescription

Description

Returns the description of a data provider's data source.

Function Group

Data Provider

Syntax

```
string DataSourceDescription ( dp )
```

```
string DataSourceDescription ( obj )
```

Input

Parameter	Description	Type	Required
obj	A report object	Report object	Yes
dp	A data provider	Data provider	Yes

Notes

The returned string is retrieved ans saved in the document when this data source is refreshed. If this data source has been never been refreshed, then this function returns an empty string

1.6.1.5.8 DataSourceLocationType

Description

Returns the location type of the data source.

Function Group

Data Provider

Syntax

```
string DataSourceLocationType ( dp )
```

```
string DataSourceLocationType ( obj )
```

Input

Parameter	Description	Type	Required
obj	A report object	Report object	Yes
dp	A data provider	Data provider	Yes

Notes

The returned string is retrieved ans saved in the document when this data source is refreshed. If this data source has been never been refreshed, then this function returns an empty string.

1.6.1.5.9 DataSourcePath

Description

Returns the full path folder of a data source.

Function Group

Data Provider

Syntax

```
string DataSourcePath ( dp )
```

```
string DataSourcePath ( obj )
```

Input

Parameter	Description	Type	Required
obj	A report object	Report object	Yes
dp	A data provider	Data provider	Yes

Notes

The returned string is retrieved ans saved in the document when this data source is refreshed. If this data source has been never been refreshed, then this function returns an empty string.

1.6.1.5.10 DataSourceParentFolder

Description

Returns the name of the parent folder containing the data provider's data source.

Function Group

Data Provider

Syntax

```
string DataSourceParentFolder ( dp )
```

```
string DataSourceParentFolder ( obj )
```

Input

Parameter	Description	Type	Required
obj	A report object	Report object	Yes
dp	A data provider	Data provider	Yes

Notes

The returned string is retrieved ans saved in the document when this data source is refreshed. If this data source has been never been refreshed, then this function returns an empty string.

1.6.1.5.11 DataSourceName

Description

Returns the name of the data source.

Function Group

Data Provider

Syntax

```
string DataSourceName ( dp )
```

```
string DataSourceName ( obj )
```

Input

Parameter	Description	Type	Required
obj	A report object	Report object	Yes
dp	A data provider	Data provider	Yes

Notes

The returned string is retrieved ans saved in the document when this data source is refreshed. If this data source has been never been refreshed, then this function returns an empty string.

1.6.1.5.12 IsPromptAnswered

Description

Determines whether a prompt has been answered

Function Group

Data Provider

Syntax

```
bool IsPromptAnswered([dp;]prompt_string)
```

Input

Parameter	Description	Type	Required
dp	The data provider containing the prompt	Data provider	No
prompt_string	The prompt text	String	Yes

Notes

- You must enclose the name of the data provider in square brackets.
- IsPromptAnswered returns a Boolean value that you can use with the If function.
- If you place IsPromptAnswered directly into a column, it returns an integer (1=true, 0=false). You can format this integer using a Boolean number format.

Examples

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

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

1.6.1.5.13 LastExecutionDate

Description

Returns the date on which a data provider was last refreshed

Function Group

Data Provider

Syntax

```
date LastExecutionDate(dp)
```

Input

Parameter	Description	Type	Required
dp	The data provider	Data provider	Yes

Notes

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

Examples

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

Related Information

DataProvider [page 127]

1.6.1.5.14 LastExecutionDuration

Description

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

Function Group

Data Provider

Syntax

```
num LastExecutionDuration(dp)
```

Input

Parameter	Description	Type	Required
dp	The data provider	Data provider	Yes

Notes

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

Examples

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

1.6.1.5.15 LastExecutionTime

Description

Returns the time at which a data provider was last refreshed

Function Group

Data Provider

Syntax

```
time LastExecutionTime(dp)
```

Input

Parameter	Description	Type	Required
dp	The data provider	Data provider	Yes

Notes

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

Examples

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

Related Information

DataProvider [page 127]

1.6.1.5.16 NumberOfColumns

Description

Returns the number of columns in a data provider

Function Group

Data Provider

Syntax

```
int NumberOfColumns(dp)
```

Input

Parameter	Description	Type	Required
dp	The data provider	Data provider	Yes

Example

`NumberOfColumns([Query 1])` returns 5 if the "Query 1" data provider has 5 rows.

1.6.1.5.17 NumberOfDataProviders

Description

Returns the number of data providers in a report

Function Group

Data Provider

Syntax

```
int NumberOfDataProviders()
```

Examples

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

1.6.1.5.18 NumberOfRows

Description

Returns the number of rows in a data provider

Function Group

Data Provider

Syntax

```
int NumberOfRows(dp)
```

Input

Parameter	Description	Type	Required
dp	The data provider	Data provider	Yes

Notes

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

Examples

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

Related Information

DataProvider [page 127]

1.6.1.5.19 QueryName

Description

Returns the name of the query of the data provider.

Function Group

Data Provider

Syntax

```
string QueryName ( dp )
```

```
string QueryName ( obj )
```

Input

Parameter	Description	Type	Required
obj	A report object	Report object	Yes
dp	A data provider	Data provider	Yes

1.6.1.5.20 RefValueDate

Description

Returns the date of the reference data used for data tracking

Function Group

Data Provider

Syntax

```
date RefValueDate()
```

Examples

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

1.6.1.5.21 RefValueUserReponse

Description

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

Function Group

Data Provider

Syntax

```
string RefValueUserResponse([dp;]prompt_string[;Index])
```

Input

Parameter	Description	Type	Required
dp	The data provider	Data provider	No

Parameter	Description	Type	Required
prompt_string	The prompt text	String	Yes
Index	Tells the function to return the database primary keys of the prompt values	Keyword	No

Notes

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

Examples

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

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

1.6.1.5.22 ServerValue

Description

Returns the database value of a measure

Function Group

Data Provider

Syntax

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

Input

Parameter	Description	Type	Required
measure	Any measure	Measure	Yes

Notes

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

Example

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

1.6.1.5.23 UniverseName

Description

Returns the name of the universe or file name on which a data provider is based for universes, text, Excel and FHSQL data sources.

Function Group

Data Provider

Syntax

```
string UniverseName(dp[,string])
```

```
string UniverseName(obj[,string])
```

Input

Parameter	Description	Type	Required
dp	The data provider	Data provider	Yes

Notes

- For SAP BW and SAP HANA data sources the functions returns the view name. The opt can be used:
  - `key`: to return the view's technical name
  - `caption`: to return the view's caption
  - `full`: to return the view's full path (package + key). For exemple, "`sales.sales::revenue`" for SAP HANA or "`xxx`" for SAP BW
- The value of `dp` in the formula is automatically updated if the name of the data provider changes. If the data provider is renamed to "Q1", the formula becomes `UniverseName([Q1])`.
- You must enclose the name of the data provider in square brackets.
- You can use the `DataProvider` function to provide a reference to a data provider.

Examples

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

Related Information

DataProvider [page 127]

1.6.1.5.24 UserResponse

Description

Returns the response to a prompt

Function Group

Data Provider

Syntax

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

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

```
string UserResponse ([dp;] prompt_string [;Index] [;multi_separator|
DefaultSeparator; between_separator])
```

Input

Parameter	Description	Type	Required
dp	The data provider	Data provider	No
prompt_string	The prompt text	String	Yes
Index	Tells the function to return the database primary keys of the prompt values	Keyword	No
multi_separator	The string used to separate different answered values. By default, it is ";".	String	No
DefaultSeparator	The default string (";") used to separate different answered values.	Keyword	No
between_separator	The string used to separate the two answered values of a range or between operator in a prompt.	String	No

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.
- The output of the function depends on the prompt type.

Examples

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

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

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

`UserResponse("Which country?")` returns "France" if it is a mono-value prompt.

`UserResponse("Which country?")` returns "France;Holland;USA" if it is a multi-value prompt.

`UserResponse("Which country?")` returns "France - USA" if it is an interval type prompt with a single entry.

`UserResponse("Which country?")` returns "France - Holland;Japan - USA" if it is an interval type prompt with multiple entries.

`UserResponse("Which country?")` returns "France" if it is a complex prompt used with the Equal To operator.

`UserResponse("Which country?")` returns ">France" if it is a complex prompt used with the Greater Than operator.

`UserResponse("Which country?")` returns ">=France" if it is a complex prompt used with the Greater Than or Equal To operator.

`UserResponse("Which country?")` returns "<France" if it is a complex prompt used with the Less Than operator.

`UserResponse("Which country?")` returns "<=France" if it is a complex prompt usedd with the Less Than or Equal To operator.

`UserResponse("Which country?")` returns "France - USA" if it is a complex prompt used with the Between operator.

`UserResponse("Which country?")` returns "!France" if it is a complex prompt used with the Not Equal To operator.

`UserResponse("Which country?")` returns "Austria;France - Holland;>Japan" if it is a complex prompt with multiple entries.

1.6.1.6 Document functions

1.6.1.6.1 DocumentAuthor

Description

Returns the InfoView logon of the document creator

Function Group

Document

Syntax

```
string DocumentAuthor()
```

Examples

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

1.6.1.6.2 DocumentCreationDate

Description

Returns the date on which a document was created

Function Group

Document

Syntax

```
date DocumentCreationDate()
```

Examples

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

1.6.1.6.3 DocumentCreationTime

Description

Returns the time when a document was created

Function Group

Document

Syntax

```
time DocumentCreationTime()
```

Examples

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

1.6.1.6.4 DocumentDate

Description

Returns the date on which a document was last saved

Function Group

Document

Syntax

```
date DocumentDate()
```

Examples

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

1.6.1.6.5 DocumentDescription

Description

Returns the document description in the user's preferred viewing locale.

Function Group

Document

Syntax

```
string DocumentDescription()
```

Example

`DocumentDescription()` "Sales figures analysis of 2019's four quarters" returns if the document description is "Sales figures analysis of 2019's four quarters".

1.6.1.6.6 DocumentName

Description

Returns the document name

Function Group

Document

Syntax

```
string DocumentName()
```

Examples

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

1.6.1.6.7 DocumentOwner

Description

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

Function Group

Document

Syntax

```
string DocumentOwner()
```

Examples

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

1.6.1.6.8 DocumentParentFolder

Description

Returns the name of the folder that contains the document.

Function Group

Document

Syntax

```
string DocumentParentFolder()
```

Notes

- This function returns the folder containing the current document where this function is used.
- If used in an instance, this function returns the folder containing the scheduled document.
- If the document is stored in a user's Inbox or Favorites, then it returns this user's name.

Examples

- `DocumentParentFolder()` returns "`Root Folder`" if it is used in a document located in the Public Folders.
- `DocumentParentFolder()` returns "`Web Intelligence Samples`" for a document stored in this folder.
- `DocumentParentFolder()` returns "`userA`" if the document is located in the userA's Favorites or Inbox folder.
- `DocumentParentFolder()` returns "`ZZ_Charting`" if it's an instance of this document.

1.6.1.6.9 DocumentPartiallyRefreshed

Description

Determines whether a document is partially refreshed

Function Group

Document

Syntax

```
bool DocumentPartiallyRefreshed()
```

Notes

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

Examples

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

1.6.1.6.10 DocumentPath

Description

Returns the document path. In a document's instance, returns the document's path and its name.

Function Group

Document

Syntax

```
string DocumentPath()
```

Notes

For a document, the path is suffixed with "/". This is not the case for a document's instance.

Examples

- `DocumentPath()` returns "Public Folders/Web Intelligence Samples/" for a document stored in this folder.
- `DocumentPath()` returns "Public Folders/Web Intelligence Samples/ZZ_Charting" if it's used in an instance of this scheduled document.

1.6.1.6.11 DocumentTime

Description

Returns the time when a document was last saved

Function Group

Document

Syntax

```
time DocumentTime()
```

Notes

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

Example

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

1.6.1.6.12 DrillFilters

Description

Returns the results of drill filters applied to a document or object in an declared report in drill mode. You can declare a different report within the document. If you do not declare a report, the current active report is used.

Function Group

Document

Syntax

```
string DrillFilters([obj|separator[;report]])
```

Input

Parameter	Description	Type	Required
obj	A report object	Report object	Either `obj` or `separator` required
separator	The drill filter separator	String	Either `obj` or `separator` required
report	Optional. The name of the report you want to use. It must be in the document. If	String	Either `obj` or `separator` required

Parameter	Description	Type	Required
	no report is declared, then the current report is used.		

Notes

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

Examples

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

`DrillFilters()` 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 filters restricting [Country] to "US" and [Year] to 1999.

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

1.6.1.6.13 ElementLinkingFilters

Description

Returns the values selected on the Block Name block for element linking.

Function Group

Document

Syntax

```
string ElementLinkingFilters(string Block Name[;string separator[;string
report]])
```

Input

Parameter	Description	Type	Required
string Block Name	The `Block Name` argument defines the block on which the element linking has been defined.	String	Yes
string separator	This is the string used to separate different selected values. By default it is "-".	String	No
string report	If a `report` argument is provided, it returns the element linking's selected values in the specified report. Otherwise, the values are returned in the report where the formula is used.	String	No

Notes

- In a document migrated from a previous version, `ElementLinkingFilters` works properly only after new values have been selected for the element linking and the document has been saved.

1.6.1.6.14 InputControlFilter

Description

Returns the values selected by the user through input control.

Function Group

Document

Syntax

```
string InputControlFilter(string InputControlName [;string separator][;string
report])
```

Input

Parameter	Description	Type	Required
string InputControlName	This is the input control name. Only the values selected by this input control are returned by the function.	String	Yes
string separator	This is the string used to separate different selected values or the values selected for different input controls. By default, it is ";".	String	No
string report	If a `report` argument is provided, it returns the input control's selected values in the specified report. Otherwise, the values are returned in the report where the formula is used.	String	No

Notes

- In a document migrated from a previous version, `InputControlFilter` works properly only after new values have been selected for the input control and the document has been saved.

1.6.1.6.15 PromptSummary

Description

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

Function Group

Document

Syntax

```
string PromptSummary([dp];[sorting_order];[show_definitive_prompts])
```

Input

Parameter	Type	Description	Required
dp	Data Provider	A data provider	No
sorting_order	Pre-Defined	Sorting order of the function output	No (default value = ascending)
show_definitive_prompts	Boolean	Show BW defintive prompts in the function output	No (default value = false)

Notes

- The optional use of `dp` allows to filter the output of PromptSummary() to the specified query.
- Possible values for the `sorting_order` parameter are: default, ascending or descending.
- If the `sorting_order` is not used in the function, then the data source order is used.
- If the `show_definitive_prompts` parameter is not used, the BW Definitive prompts are hidden in the function output.

Example

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

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

1.6.1.6.16 QuerySummary

Description

Returns information about the queries in a document. For each query, the method returns the data provider type, the data provider name, refresh information, the query properties and the query definition (Result Objects and Filters).

Function Group

Document

Syntax

```
string QuerySummary()
```

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

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

Input

Parameter	Description	Type	Required
dp	A data provider	Data provider	No
StatusOfData	BW Status of data	Boolean	No

Notes

- If you pass a data provider name as a parameter, then the function returns details only for this data provider. You must enclose this data provider name in square brackets.
- The name of the data provider is prefixed by its type, that can be: `Universe`, `Web Intelligence`, `Excel`, `Text`, `Free-hand SQL`, `SAP HANA`, `SAP BW` or `Web Service`.
- The BW status of data indicates the last refresh date of the BW info provider, and appears as the last line returned by the function.

Examples

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

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

Output example:

```
*** Query Name:Query 1 ***
    ** Query Properties:
       Universe:eFashion
       Last Refresh Date:4/1/20 5:15 PM
       Last Execution Duration: 2
       Number of rows: 586
       Refreshable: ON
       Retrieve Duplicate Rows: ON
       Retrieve Empty Rows: OFF
       Max Retrieval Time (s): /
       Max Rows Retrieved: /
       Query Stripping: OFF
    ** Query Definition:
```

```
        Result Objects: State, Year, Sales revenue, City, Quarter, Month
```

1.6.1.6.17 ReportFilter

Description

Returns the report filters applied to an object.

Function Group

Document

Syntax

```
string ReportFilter(obj[;separator])
```

Input

Parameter	Description	Type	Required
obj	A report object	Report object	Yes
separator	This string can be used to separate different values in the list. By default, the separator is ";".	String	No

Examples

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

1.6.1.6.18 ReportFilterSummary

Description

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

Function Group

Document

Syntax

```
string ReportFilterSummary(report_name)
```

Input

Parameter	Description	Type	Required
report_name	The name of the report	String	No

Notes

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

Examples

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

`ReportFilterSummary("Report1")` returns information about the report filters in the "Report1" report.

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

```
Filters on Report1:
                            (Sales Revenue Greater Than 1000000
                             Or (Sales Revenue Less Than 3000))
                        Filters on Section on City:
                             (City InList{"Los Angeles";"San Diego";})
```

```
                          Ranking Filter:
                               (Top 10 & Bottom 10 [Customer] Based on [Sales
Revenue] (Count))
```

1.6.1.7 Logical functions

1.6.1.7.1 Even

Description

Determines whether a number is even

Function Group

Logical

Syntax

```
bool Even(number)
```

Input

Parameter	Description	Type	Required
number	Any number	Number	Yes

Notes

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

Examples

`Even(4)` returns True.

`Even(3)` returns False.

`Even(23.2)` returns False.

`Even(-4)` returns True.

`Even(-2.2)` returns False.

1.6.1.7.2 IsDate

Description

Determines whether a value is a date

Function Group

Logical

Syntax

```
bool IsDate(obj)
```

Input

Parameter	Description	Type	Required
obj	Any report object	Report object	Yes

Notes

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

Examples

`IsDate([Reservation Date])` returns True if [Reservation Date] is a date.

Or one of the following to return "Date" if [Reservation Date] is a date:

- `If(IsDate([Reservation Date])) Then "Date" Else "Not a date"`
- `If IsDate([Reservation Date]) Then "Date" Else "Not a date"`

Related Information

If...Then...Else [page 238]

1.6.1.7.3 IsError

Description

Determines whether an object returns an error

Function Group

Logical

Syntax

```
bool IsError(obj)
```

Input

Parameter	Description	Type	Required
obj	Any report object	Report object	Yes

Notes

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

Examples

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

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

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

Related Information

If...Then...Else [page 238]

1.6.1.7.4 IsLogical

Description

Determines whether a value is boolean

Function Group

Logical

Syntax

```
bool IsLogical(obj)
```

Input

Parameter	Description	Type	Required
obj	Any report object	Report object	Yes

Notes

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

Examples

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

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

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

Related Information

If...Then...Else [page 238]

1.6.1.7.5 IsNull

Description

Determines whether a value is null

Function Group

Logical

Syntax

```
bool IsNull(obj)
```

Input

Parameter	Description	Type	Required
obj	Any report object	Report object	Yes

Notes

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

Examples

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

`IsNull([Average Guests])` returns True if the [Average Guests] variable is null.

Related Information

If...Then...Else [page 238]

1.6.1.7.6 IsNumber

Description

Determines whether a value is a number

Function Group

Logical

Syntax

```
bool IsNumber(obj)
```

Input

Parameter	Description	Type	Required
obj	Any report object	Report object	Yes

Notes

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

Examples

`IsNumber([Revenue])` returns True if the [Revenue] variable is a number.

`IsNumber([Customer Name])` returns False if the [Customer Name] variable is not a number.

`If IsNumber([Customer Name]) Then "Number" Else "Not a number"` returns "Not a number" if the [Customer Name] variable is not a number.

Related Information

If...Then...Else [page 238]

1.6.1.7.7 IsString

Description

Determines whether a value is a string

Function Group

Logical

Syntax

```
bool IsString(obj)
```

Input

Parameter	Description	Type	Required
obj	Any report object	Report object	Yes

Notes

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

Examples

`IsString([Revenue])` returns false if the [Revenue] variable is not a string.

`IsString([Customer Name])` returns true if the [Customer Name] variable is a string.

`If IsString([Customer Name]) Then "String" Else "Not a string"` returns "String" if the [Customer Name] variable is a string.

Related Information

If...Then...Else [page 238]

1.6.1.7.8 IsTime

Description

Determines whether a variable is a time variable

Function Group

Logical

Syntax

```
bool IsTime(obj)
```

Input

Parameter	Description	Type	Required
obj	Any report object	Report object	Yes

Notes

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

Examples

`IsTime([Reservation Time])` returns true if the [Reservation Time] variable is a time variable.

`IsTime([Average Guests])` returns false if the [Average Guests] variable is not a time variable.

`If IsTime([Average Guests]) Then "Time" Else "Not time"` returns "Not time" if the [Average Guests] variable is not a time variable.

Related Information

If...Then...Else [page 238]

1.6.1.7.9 Odd

Description

Determines whether a number is odd

Function Group

Logical

Syntax

```
bool Odd(number)
```

Input

Parameter	Description	Type	Required
number	Any number	Number	Yes

Notes

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

- `Odd` returns True for every decimal number, and False for every even integer.

Examples

`Odd(5)` returns True.

`Odd(4)` returns False.

`Odd(23.2)` returns True.

`Odd(24.2)` returns True.

`Odd(-23.2)` returns True.

`Odd(-24.2)` returns True.

Related Information

If...Then...Else [page 238]

1.6.1.8 Numeric functions

1.6.1.8.1 Abs

Description

Returns the absolute value of a number

Function Group

Numeric

Syntax

```
num Abs(number)
```

Input

Parameter	Description	Type	Required
number	Any number	Number	Yes

Examples

`Abs(25)` returns 25.

`Abs(-11)` returns 11.

1.6.1.8.2 Ceil

Description

Returns a number rounded up to the nearest integer

Function Group

Numeric

Syntax

```
num Ceil(number)
```

Input

Parameter	Description	Type	Required
number	Any number	Number	Yes

Examples

`Ceil(2.4)` returns 3.

`Ceil(3.1)` returns 4.

`Ceil(-3.1)` returns -3.

1.6.1.8.3 Cos

Description

Returns the cosine of an angle

Function Group

Numeric

Syntax

```
num Cos(angle)
```

Input

Parameter	Description	Type	Required
angle	An angle in radians	Number	Yes

Example

`Cos(180)` returns -0.6.

1.6.1.8.4 EuroConvertFrom

Description

Converts a Euro amount to another currency

Function Group

Numeric

Syntax

```
num EuroConvertFrom(euro_amount;curr_code;round_level)
```

Input

Parameter	Description	Type	Required
euro_amount	The amount in Euros	Number	Yes
curr_code	The ISO code of the target currency	String	Yes
round_level	The number of decimal places to which the result is rounded	Number	Yes

Notes

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

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	Portuguese escudo
FIM	Finnish mark

Examples

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

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

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

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

Related Information

Rounding and truncating numbers [page 283]

1.6.1.8.5 EuroConvertTo

Description

Converts an amount to Euros

Function Group

Numeric

Syntax

```
num EuroConvertTo(noneuro_amount;curr_code;round_level)
```

Input

Parameter	Description	Type	Required
euro_amount	The amount in the non-euro currency	Number	Yes
curr_code	The ISO code of the non-euro currency	String	Yes
round_level	The number of decimal places to which the result is rounded	Number	Yes

Example

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

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

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

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

Note

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

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	Portuguese escudo
FIM	Finnish mark

Related Information

Rounding and truncating numbers [page 283]

1.6.1.8.6 EuroFromRoundError

Description

Returns the rounding error in a conversion from Euros

Function Group

Numeric

Syntax

```
num EuroFromRoundError(euro_amount;curr_code;round_level)
```

Input

Parameter	Description	Type	Required
euro_amount	The amount in Euros	Number	Yes
curr_code	The ISO code of the target currency	String	Yes
round_level	The number of decimal places to which the result is rounded	Number	Yes

Output

The rounding error in the calculation

Examples

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

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

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

`EuroFromRoundError(1000;"DEM";1)` returns -0.01. (The unrounded conversion is 1955.83. The conversion rounded to 1 decimal place is 1995.80. The rounding error is -0.03.)

Note

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

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	Portuguese escudo
FIM	Finnish mark

Related Information

Rounding and truncating numbers [page 283]

1.6.1.8.7 EuroToRoundError

Description

Returns the rounding error in a conversion to Euros

Function Group

Numeric

Syntax

```
num EuroToRoundError(noneuro_amount;curr_code;round_level)
```

Input

Parameter	Description	Type	Required
euro_amount	The amount in the non-euro currency	Number	Yes
curr_code	The ISO code of the non-euro currency	String	Yes
round_level	The number of decimal places to which the result is rounded	Number	Yes

Examples

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

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

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

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

Note

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

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	Portuguese escudo
FIM	Finnish mark

Related Information

Rounding and truncating numbers [page 283]

1.6.1.8.8 Exp

Description

Returns an exponential (e raised to a power)

Function Group

Numeric

Syntax

```
num Exp(power)
```

Input

Parameter	Description	Type	Required
power	The power	Number	Yes

Notes

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

Examples

`Exp(2.2)` returns 9.03.

1.6.1.8.9 Fact

Description

Returns the factorial of a number

Function Group

Numeric

Syntax

```
int Fact(number)
```

Input

Parameter	Description	Type	Required
number	Any number	Number	Yes

Notes

The factorial of `number` is the product of all the integers from 1 to `number`.

Examples

`Fact(4)` returns 24.

`Fact(5.9)` returns 120.

1.6.1.8.10 Floor

Description

Returns a number rounded down to the nearest integer

Function Group

Numeric

Syntax

```
int Floor(number)
```

Input

Parameter	Description	Type	Required
number	Any number	Number	Yes

Example

`Floor(24.4)` returns 24.

1.6.1.8.11 Ln

Description

Returns the natural logarithm of a number

Function Group

Numeric

Syntax

```
num Ln(number)
```

Input

Parameter	Description	Type	Required
number	Any number	Number	Yes

Examples

`Ln(10)` returns 2.3

1.6.1.8.12 Log

Description

Returns the logarithm of a number in a specified base

Function Group

Numeric

Syntax

```
num Log(number;base)
```

Input

Parameter	Description	Type	Required
number	Any number	Number	Yes
base	The base of the logarithm	Number	Yes

Examples

`Log(125;5)` returns 3.

1.6.1.8.13 Log10

Description

Returns the base 10 logarithm of a number

Function Group

Numeric

Syntax

```
num Log10(number)
```

Input

Parameter	Description	Type	Required
number	Any number	Number	Yes

Examples

`Log10(100)` returns 2.

1.6.1.8.14 Mod

Description

Returns the remainder from the division of two numbers

Function Group

Numeric

Syntax

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

Input

Parameter	Description	Type	Required
dividend	The dividend	Number	Yes
divisor	The divisor	Number	Yes

Examples

`Mod(10;4)` returns 2.

`Mod (10.2;4.2)` returns 1.8.

1.6.1.8.15 Power

Description

Returns a number raised to a power

Function Group

Numeric

Syntax

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

Input

Parameter	Description	Type	Required
number	The number to raise to a power	Number	Yes
power	The power	Number	Yes

Example

`Power(10;2)` returns 100.

1.6.1.8.16 Rank

Description

Ranks a measure by dimensions

Function Group

Numeric

Syntax

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

Input

Parameter	Description	Type	Required
measure	The measure to be ranked	Measure	Yes
ranking_dims	The dimensions used to rank the measure	Dimension list	No
Top|Bottom	Sets the ranking order: • `Top` - descending • `Bottom` - ascending	Keyword	No (`Top` is default)
reset_dims	The dimensions that reset the ranking	Dimension list	No

Notes

- The function uses the default calculation context to calculate the ranking if you do not specify ranking dimensions.

- You must always place dimensions in parentheses even if there is only one dimension in the list of ranking or reset dimensions.
- When you specify a set of ranking or reset dimensions you must separate them with semi-colons.
- By default the ranking is reset over a section or block break.

Examples

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

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

Related Information

Bottom/Top operators [page 266]

1.6.1.8.17 Round

Description

Rounds a number

Function Group

Numeric

Syntax

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

Input

Parameter	Description	Type	Required
number	The number to be rounded	Number	Yes
round_level	The number of decimal places to which the number is rounded	Number	Yes

Examples

`Round(9.44;1)` returns 9.4.

`Round(9.45;1)` returns 9.5.

`Round(9.45;0)` returns 9.

`Round(9.45;-1)` returns 10.

`Round(4.45;-1)` returns 0.

Related Information

Rounding and truncating numbers [page 283]

1.6.1.8.18 Sign

Description

Returns the sign of a number

Function Group

Numeric

Syntax

```
int Sign(number)
```

Input

Parameter	Description	Type	Required
number	Any number	Number	Yes

Notes

`Sign` returns -1 if `number` is negative, 0 if `number` is zero and 1 if `number` is positive.

Examples

`Sign(3)` returns 1.

`Sign(-27.5)` returns -1.

1.6.1.8.19 Sin

Description

Returns the sine of an angle.

Function Group

Numeric

Syntax

```
num Sin(angle)
```

Input

Parameter	Description	Type	Required
angle	An angle in radians	Number	Yes

Example

`Sin(234542)` can return, depending on the decimal point setting, -0.116992 or -0.12.

1.6.1.8.20 Sqrt

Description

Returns the square root of a number

Function Group

Numeric

Syntax

```
num Sqrt(number)
```

Input

Parameter	Description	Type	Required
number	Any number	Number	Yes

Example

`Sqrt(25)` returns 5.

1.6.1.8.21 Tan

Description

Returns the tangent of an angle

Function Group

Numeric

Syntax

```
num Tan(angle)
```

Input

Parameter	Description	Type	Required
angle	An angle in radians	Number	Yes

Examples

`Tan(90)` returns -2.

1.6.1.8.22 ToDecimal

Description

Returns a decimal.

Function Group

Numeric

Syntax

```
num ToDecimal(number|string)
```

Input

Parameter	Description	Type	Required
number|string	A number or a string that can be interpreted as a number	Number or string	Yes

Notes

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

Examples

`ToDecimal("1234567890.1234567890")` returns `1234567890.1234567890`.

`ToDecimal("1234567890.12345")` returns `1234567890.12345`.

`ToDecimal("abcdefijkl")` returns `#ERROR`.

1.6.1.8.23 ToNumber

Description

Returns a string as a number

Function Group

Numeric

Syntax

```
num ToNumber(string)
```

or

Input

Parameter	Description	Type	Required
string	A number as a string	String	Yes

Notes

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

Examples

`ToNumber("45")` returns 45.

1.6.1.8.24 Truncate

Description

Truncates a number

Function Group

Numeric

Syntax

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

Input

Parameter	Description	Type	Required
number	The number to be rounded	Number	Yes
truncate_level	The number of decimal places to which the number is truncated	Number	Yes

Notes

Example

`Truncate(3.423;2)` returns 3.42.

Related Information

Rounding and truncating numbers [page 283]

1.6.1.9 Set functions

1.6.1.9.1 Ancestor

Description

Returns an ancestor member of a member

Function Group

Set

Syntax

```
member Ancestor(member;level|distance)
```

Input

Parameter	Description	Type	Required
member	Any member	member	Yes
level	The level of the ancestor	level	Either `level` or `distance` is required

Parameter	Description	Type	Required
distance	The distance of the ancestor level from the current level	int	Either `level` or `distance` is required

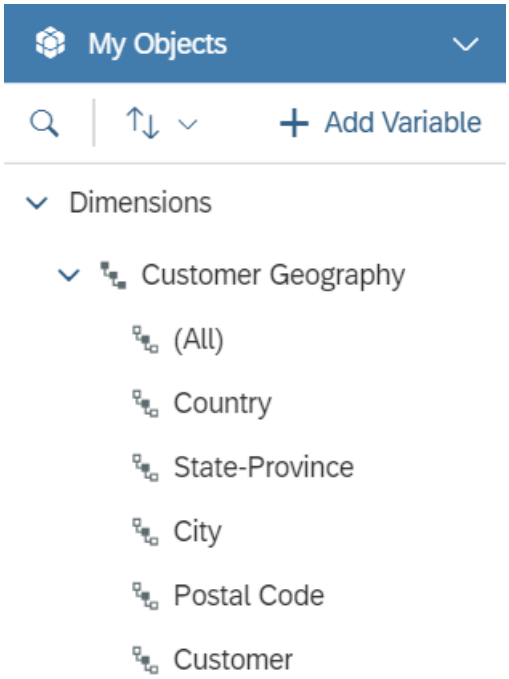
Notes

- `Ancestor` is not used as a standalone function. It is used in the input parameter in aggregate functions that specifies the member set for aggregation.
- `member` is the current member of a hierarchy. When the hierarchy is not in the context of the block, the formula returns an empty value.
- `distance` must be positive.

Examples

The following examples are all taken from an English language data source.

Based on the following geography hierarchy, you want to know the Internet Sales Amount impact of each customer independant of the customer's city.



Firstly, for each City, you want the Internet Sales Amount value for its Country:

```
=Sum([Query 2].[Internet Sales].[Internet Sales Amount];{Ancestor([Customer
Geography];[Customer Geography].[City])})
```

Customer Geography	Sales Amount per State/Province	Internet Sales Amount
All Customers		29,358,677.22
Australia		9,061,000.58
New South Wales	3,934,485.73	3,934,485.73
Coffs Harbour	3,934,485.73	235,454.97
2450	3,934,485.73	235,454.97
Adriana Smith	3,934,485.73	5,333.25
Aimee Guo	3,934,485.73	77.27
Allison R. Young	3,934,485.73	39.98
Ann A. Sara	3,934,485.73	39.98
Antonio G. Patterson	3,934,485.73	8,068.03
Ariana Stewart	3,934,485.73	6,070.59
Arthur Kapoor	3,934,485.73	23.97
Barbara W. Lal	3,934,485.73	2,795.01
Bobby D. Saunders	3,934,485.73	120.48
Brianna J. Johnson	3,934,485.73	38.98

Then you calculate the contribution of each City in the country's global Internet Sales amount :

```
=[Query 2].[Internet Sales].[Internet Sales Amount] / Sum([Query 2].[Internet
Sales].[Internet Sales Amount];{Ancestor([Customer Geography];[Customer
Geography].[City])})
```

Customer Geography	City Contribution	Internet Sales Amount
All Customers		29,358,677.22
Australia		9,061,000.58
New South Wales	100.00%	3,934,485.73
Coffs Harbour	5.98%	235,454.97
2450	5.98%	235,454.97
Adriana Smith	0.14%	5,333.25
Aimee Guo	0.00%	77.27
Allison R. Young	0.00%	39.98
Ann A. Sara	0.00%	39.98
Antonio G. Patterson	0.21%	8,068.03
Ariana Stewart	0.15%	6,070.59
Arthur Kapoor	0.00%	23.97
Barbara W. Lal	0.07%	2,795.01
Bobby D. Saunders	0.00%	120.48
Brianna J. Johnson	0.00%	38.98

Note

When using BICS connections to SAPBW providers, you need to specify an offset level instead of naming the level:

```
=[Query 2].[Internet Sales].[Internet Sales Amount] / Sum([Query 2].[Internet
Sales].[Internet Sales Amount];{Ancestor([Customer Geography];2)})
```

In this case you will have results also for State Province and Country.

Related Information

Aggregate [page 41]
Average [page 48]
Count [page 49]
Max [page 55]
Min [page 57]
Sum [page 77]

1.6.1.9.2 Children

Description

Returns the child members of a hierarchy member within an aggregate function.

Function Group

Set

Syntax

```
member_set member.Children
```

Input

Parameter	Description	Type	Required
member	Any member	member	Yes

Notes

- `Children` is not used as a standalone function. It is used in the input parameter in aggregate functions that specifies the member set for aggregation.
- `member` is the current member of a hierarchy. When the hierarchy is not in the context of the block, the formula returns an empty value.

Examples

`[Geography].[US].[California].Children` returns [Los Angeles], [San Francisco], [San Diego].

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

Related Information

Aggregate [page 41]
Average [page 48]
Count [page 49]
Max [page 55]
Min [page 57]
Sum [page 77]

1.6.1.9.3 Depth

Description

Returns the depth of a member in a hierarchy

Function Group

Set

Syntax

```
int member.Depth
```

Input

Parameter	Description	Type	Required
member	Any member	member	Yes

Notes

- The depth is the distance of the member from top level of the hierarchy.
- The top level of the hierarchy is level 0.
- `member` is the current member of a hierarchy. When the hierarchy is not in the context of the block, the formula returns an empty value.

Examples

You want to know the depth of hierarchy members:

```
=[Calendar].[Date.Calendar].Depth
```

Date.Calendar	'=[Calendar].[Date.Calendar].Depth
All Periods	0
CY 2001	1
H2 CY 2001	2
Q3 CY 2001	3
July 2001	4
July 1, 2001	5
July 2, 2001	5
July 3, 2001	5
July 4, 2001	5
July 5, 2001	5
July 6, 2001	5
July 7, 2001	5

Now combine with the Children functions to check if you have all days listed every month:

```
=If [Calendar].[Date.Calendar].Depth = 4 Then Count([Internet Sales].[Internet
Sales Amount];{[Calendar].[Date.Calendar].Children()})
```

Date.Calendar	Internet Sales Amount	[Date.Calendar].Depth	[Date.Calendar].Children()
All Periods	29,358,677.22	0	
CY 2001	3,266,373.66	1	
H2 CY 2001	3,266,373.66	2	
Q3 CY 2001	1,453,522.89	3	
July 2001	473,388.16	4	31
August 2001	506,191.69	4	30
September 2001	473,943.03	4	29
Q4 CY 2001	1,812,850.77	3	
October 2001	513,329.47	4	30
November 2001	543,993.41	4	30
December 2001	755,527.89	4	31

1.6.1.9.4 Descendants

Description

Returns descendants of a hierarchy member within an aggregation function.

Function Group

Set

Syntax

```
member_set Descendants(member[;level|distance][;desc_flag])
```

Input

Parameter	Description	Type	Required
member	Any member	member	Yes

Parameter	Description	Type	Required
level	The level of the descendants	level	No (the level of `member` is the default)
distance	The distance of the descendant level from the current level	int	No (the level of `member` is the default)
desc_flag	Determines which descendant members are returned	keyword	No (default is `Self`)

Notes

- `Descendants` is not used as a standalone function. It is used in the input parameter in aggregate functions that specifies the member set for aggregation.
- `member` is the current member of a hierarchy. When the hierarchy is not in the context of the block, the formula returns an empty value.
- `Self` in `desc_flag` refers to the level specified by the `level|distance` parameter.
- `Before` in `desc_flag` refers to all levels above the level specified by the `level|distance` parameter.
- `After` in `desc_flag` refers to all levels below the level specified by the `level|distance` parameter.
- The values of `desc_flag` are as follows:

Self	Returns the descendants at the level specified by the `level|distance` parameter, including the current member if it is at this level.
Before	Returns the current member and all descendants above the level specified by the `level|distance` parameter.
After	Returns the descendants below the level specified by the `level|distance` parameter.
Self_Before	Returns the current member and all descendants above and including the level specified by the `level|distance` parameter.
Self_After	Returns the current member and all descendants at and below the level specified by the `level|distance` parameter.
Before_After	Returns the current member and all descendants except those at the level specified by the `level|distance` parameter.
Self_Before_After	Returns the current member and all descendants.
Leaves	Returns all members between the current member and the level specified by the `level|distance` parameter that do not have child members.

- `distance` must be positive.

Example

You have a financial hierarchy, some of the nodes are not always cumulative ones, but you want to sum their descendants. In this example, you will get the sum of descendants of each Balance Sheet member, only 1 level below:

```
=Sum([Query 3 (1)].[Financial Reporting].[Amount];
{Descendants([Accounts]&[Balance Sheet];1)})
```

Accounts		
Balance Sheet	0	
Assets	13,740,731	
Liabilities and Owners Equity	13,740,731	
Net Income	12,609,503	

27,481,462

```
=Sum([Query 3 (1)].[Financial Reporting].[Amount];
{Descendants([Accounts]&[Balance Sheet].[Assets].[Current Assets];1;Leaves)})
```

Balance Sheet	0	
Assets	13,740,731	
Current Assets	12,445,628	
Cash	3,236,799	
Receivables	3,475,923	
Trade Receivables	3,371,580	
Other Receivables	104,343	
Allowance for Bad Debt	67,429	
Inventory	4,143,398	
Raw Materials	2,007,586	
Work in Process	1,393,582	
Finished Goods	742,230	
Deferred Taxes	505,424	
Prepaid Expenses	341,992	
Intercompany Receivable	674,663	

12,445,628

Now you want to sum all members below Current Assets :

```
=Sum([Query 3 (1)].[Financial Reporting].[Amount];
{Descendants([Accounts]&[Balance Sheet].[Assets].[Current Assets];0;After)})
```

Balance Sheet	0	20,064,949
Assets	13,740,731	
Current Assets	12,445,628	
Cash	3,236,799	
Receivables	3,475,923	
Trade Receivables	3,371,580	
Other Receivables	104,343	
Allowance for Bad Debt	67,429	
Inventory	4,143,398	
Raw Materials	2,007,586	
Work in Process	1,393,582	
Finished Goods	742,230	
Deferred Taxes	505,424	
Prepaid Expenses	341,992	
Intercompany Receivable	674,663	

Now add Current Assets itself :

```
=Sum([Query 3 (1)].[Financial Reporting].[Amount];
{Descendants([Accounts]&[Balance Sheet].[Assets].[Current Assets];0;Self_After)})
```

Balance Sheet	0	32,510,577
Assets	13,740,731	
Current Assets	12,445,628	
Cash	3,236,799	
Receivables	3,475,923	
Trade Receivables	3,371,580	
Other Receivables	104,343	
Allowance for Bad Debt	67,429	
Inventory	4,143,398	
Raw Materials	2,007,586	
Work in Process	1,393,582	
Finished Goods	742,230	
Deferred Taxes	505,424	
Prepaid Expenses	341,992	
Intercompany Receivable	674,663	

Related Information

Aggregate [page 41]
Average [page 48]
Count [page 49]
Max [page 55]
Min [page 57]
Sum [page 77]

1.6.1.9.5 IsLeaf

Description

Determines whether a member is a leaf member

Function Group

Misc

Syntax

```
bool member.IsLeaf
```

Input

Parameter	Description	Type	Required
member	Any member	member	Yes

Notes

- A leaf member is a member that does not have any child members.
- `member` is the current member of a hierarchy. When the hierarchy is not in the context of the block, the formula returns an empty value.

Examples

You want to know if the line is a day:

```
=[Calendar].[Date.Calendar].IsLeaf()
```

Date.Calendar	'=[Query 1].[Calendar].[Date.Calendar].IsLeaf
All Periods	0
CY 2001	0
H2 CY 2001	0
Q3 CY 2001	0
July 2001	0
July 1, 2001	1
July 2, 2001	1
July 3, 2001	1
July 4, 2001	1
July 5, 2001	1
July 6, 2001	1
July 7, 2001	1
July 8, 2001	1

1.6.1.9.6 Key

Description

Returns the key of a member

Syntax

```
string member.Key
```

Function Group

Set

Input

Parameter	Description	Type	Required
member	Any member	member	Yes

Notes

- The key is the internal identifier of a member.
- `member` is the current member of a hierarchy. When the hierarchy is not in the context of the block, the formula returns an empty value.

Example

`[Geography].[US].Key` returns "XYZ" if the key of the [US] member is "XYZ".

1.6.1.9.7 Lag

Description

Returns a member at the same level as the current member and a given distance after it, within an aggregate function.

Syntax

```
member member.Lag(distance)
```

Function Group

Set

Input

Parameter	Description	Type	Required
member	Any member	member	Yes
distance	The distance of the member from the current member	int	Yes

Notes

- `Lag` is not used as a standalone function. It is used in the input parameter in aggregate functions that specifies the member set for aggregation.
- If `distance` is positive, `Lag` returns the member `distance` places after `member`. If `distance` is negative, `Lag` returns the member `distance` places before `member`.
- `member` is the current member of a hierarchy. When the hierarchy is not in the context of the block, the formula returns an empty value.
- `Lag` uses the member order in the hierarchy and query to return the related member.

Examples

You want to get the differences in internet sales from week to week.

```
=Max([Internet Sales].[Internet Sales Amount];{[Calendar].
[Date.Calendar].Lag(7)})
```

Date.Calendar	Internet Sales Amount	'=Max([Query 1].[Internet Sales].[Internet Sales Amount];{[Query 1].[Calendar].[Date.Calendar].Lag(7)})
All Periods	29,358,677.22	
CY 2001	3,266,373.66	
H2 CY 2001	3,266,373.66	
Q3 CY 2001	1,453,522.89	1,623,971.06
July 2001	473,388.16	550,816.69
July 1, 2001	14,477.34	7,855.64
July 2, 2001	13,931.52	20,909.78
July 3, 2001	15,012.18	10,556.53
July 4, 2001	7,156.54	14,313.08
July 5, 2001	15,012.18	14,134.8
July 6, 2001	14,313.08	7,156.54
July 7, 2001	7,855.64	25,047.89
July 8, 2001	7,855.64	11,230.63
July 9, 2001	20,909.78	14,313.08
July 10, 2001	10,556.53	14,134.8

Or you want to compare a specific year to another year two years previously:

Date.Calendar	Internet Sales Amount	
All Periods	29,358,677.22	
CY 2001	3,266,373.66	11.13%
H2 CY 2001	3,266,373.66	100.00%
CY 2002	6,530,343.53	22.24%
H1 CY 2002	3,805,710.59	58.28%
H2 CY 2002	2,724,632.94	41.72%
CY 2003	9,791,060.3	33.35%
H1 CY 2003	3,037,501.36	31.02%
H2 CY 2003	6,753,558.94	68.98%
CY 2004	9,770,899.74	33.28%

CY 2002	CY 2002.Lag(2)	CY 2002 - CY 2002.Lag(2)
6,530,343.53	9,770,899.74	-3,240,556.21

Now you want to combine Lag and IsLeaf to know over a one week period the difference for the amount sold. The formula set in the last column will be :

```
=If [Calendar].[Date.Calendar].IsLeaf() Then [Internet Sales].[Internet
Sales Amount] - Max([Internet Sales].[Internet Sales Amount];{[Calendar].
[Date.Calendar].Lag(7)})
```

Date.Calendar	Internet Sales Amount	'[Calendar].[Date.Calendar].Lag(7))	Difference week to week
All Periods	29,358,677.22		
CY 2001	3,266,373.66		
H2 CY 2001	3,266,373.66		
Q3 CY 2001	1,453,522.89	1,623,971.06	
July 2001	473,388.16	550,816.69	
July 1, 2001	14,477.34	7,855.64	6,621.7
July 2, 2001	13,931.52	20,909.78	-6,978.26
July 3, 2001	15,012.18	10,556.53	4,455.65
July 4, 2001	7,156.54	14,313.08	-7,156.54
July 5, 2001	15,012.18	14,134.8	877.38
July 6, 2001	14,313.08	7,156.54	7,156.54
July 7, 2001	7,855.64	25,047.89	-17,192.25
July 8, 2001	7,855.64	11,230.63	-3,374.99
July 9, 2001	20,909.78	14,313.08	6,596.7
July 10, 2001	10,556.53	14,134.8	-3,578.27
July 11, 2001	14,313.08	6,953.26	7,359.82

Related Information

Aggregate [page 41]
Average [page 48]
Count [page 49]
Max [page 55]
Min [page 57]
Sum [page 77]

1.6.1.9.8 MemberAtDepth

Description

Returns the members of a hierarchy, at a chosen depth.

Function Group

Set

Syntax

```
string MemberAtDepth(hierarchy;depth)
```

Input

Parameter	Description	Type	Required
`hierarchy`	Hierarchical object	dimension	Yes
`depth`	The level of the member set in the chosen hierarchy	int	Yes

Notes

- `depth` must be positive or zero, else the function will return an error
- When there is no member at the given depth, then `MemberAtDepth()` returns `Null`
- When the input object is not a hierarchy, then `MemberAtDepth()` returns `Null` for any depth greater than zero

Examples

The following `[Country]` hierarchy has been filtered to keep only two children of the EUROPE node.

Country	Quantity
WORLD	262,461
EUROPE	31,009
France	6,965
Germany	6,331
NORTH_AMERICA	219,944
Canada	17,754
USA	202,190
ASIA_PAC	9,065

`MemberAtDepth([Country];0)` returns:

	Order Quantity
WORLD	755,719

“WORLD” is the root and the only value at level 0.

If the *Avoid duplicate row aggregation* option is disabled, then the function aggregates all the members' values since they all stem from the "WORLD" root. If you want to check all the aggregated values, enable the *Avoid duplicate row aggregation* option. The hierarchy would then look like this:

	Order Quantity
WORLD	262,461
WORLD	31,009
WORLD	6,965
WORLD	6,331
WORLD	219,944
WORLD	17,754
WORLD	202,190
WORLD	9,065
Sum:	**755,719**

`MemberAtDepth([Country];1)` returns:

	Order Quantity
	262,461
ASIA_PAC	9,065
EUROPE	44,305
NORTH_AME	439,888

The first row contains a NULL value because the root has no values at level 1.

On other rows, the function aggregates all the members' values of the `[Country]` hierarchy at level 1: “ASIA_PAC”, “EUROPE” and “NORTH_AMERICA”. If you want to check all the aggregated values, enable the *Avoid duplicate row aggregation* option. The hierarchy would then look like this:

	Order Quantity
	262,461
ASIA_PAC	9,065
EUROPE	31,009
EUROPE	6,965
EUROPE	6,331
NORTH_AME	219,944
NORTH_AME	17,754
NORTH_AME	202,190
Sum:	**755,719**

`MemberAtDepth([Country];2)` returns:

	Order Quantity
	522,479
Canada	17,754
France	6,965
Germany	6,331
USA	202,190

Again, the first row aggregates all the members that have no values at level 2, that is, the root and each node. On other rows, all the members' values of the `[Country]` hierarchy at level 2 are aggregated.

`MemberAtDepth([Country];3)` returns:

	Order Quantity
	755,719

There is only one row left, with a NULL value, since the hierarchy has no third level. Therefore, all nodes and leaves' values of the hierarchy are aggregated.

If you want to visualize the whole hierarchy, add columns containing the levels of the hierarchy in the existing table, then use the function and give it different level values. It would then look like this:

Level 0	Level 1	Level 2	Order Quantity
WORLD			262,461
WORLD	ASIA_PAC		9,065
WORLD	EUROPE		31,009
WORLD	EUROPE	France	6,965
WORLD	EUROPE	Germany	6,331
WORLD	NORTH_AMERICA		219,944
WORLD	NORTH_AMERICA	Canada	17,754
WORLD	NORTH_AMERICA	USA	202,190

You can use the `IsLeaf` formula to filter the hierarchy leaves in the above table: For more information on the `IsLeaf` formula, see IsLeaf [page 211]

Level 0	Level 1	Level 2	Order Quantity	IsLeaf
WORLD			262,461	false
WORLD	ASIA_PAC		9,065	true
WORLD	EUROPE		31,009	false
WORLD	EUROPE	France	6,965	true
WORLD	EUROPE	Germany	6,331	true
WORLD	NORTH_AMERICA		219,944	false
WORLD	NORTH_AMERICA	Canada	17,754	true
WORLD	NORTH_AMERICA	USA	202,190	true

Once it's done, you can hide the `IsLeaf` column to get the equivalent of a flattened hierarchy table:

Level 0	Level 1	Level 2	Order Quantity
WORLD	ASIA_PAC		9,065
WORLD	EUROPE	France	6,965
WORLD	EUROPE	Germany	6,331
WORLD	NORTH_AMERICA	Canada	17,754
WORLD	NORTH_AMERICA	USA	202,190

1.6.1.9.9 Parent

Description

Returns the parent member of a hierarchy member within an aggregate function.

Function Group

Set

Syntax

```
member member.Parent
```

Input

Parameter	Description	Type	Required
member	Any member	member	Yes

Notes

- `Parent` is not used as a standalone function. It is used in the input parameter in aggregate functions that specifies the member set for aggregation.
- `member` is the current member of a hierarchy. When the hierarchy is not in the context of the block, the formula returns an empty value.

Examples

The second column contains the formula that allow yous to obtain the Parent of each hierarchy member:

```
=Max([Customer Geography];{[Customer Geography].Parent})
```

All Customers	
Australia	All Customer:
New South Wales	Australia
Alexandria	New South W
Coffs Harbour	New South W
2450	Coffs Harbou
Adriana Smith	2450
Aimee Guo	2450
Allison R. Young	2450
Ann A. Sara	2450

Related Information

Aggregate [page 41]
Average [page 48]
Count [page 49]
Max [page 55]
Min [page 57]
Sum [page 77]

1.6.1.9.10 Siblings

Description

Returns the member and sibling members of the hierarchy member within an aggregate function.

Function Group

Set

Syntax

```
member_set member.Siblings
```

Input

Parameter	Description	Type	Required
member	Any member	member	Yes

Notes

- `Siblings` is not used as a standalone function. It is used in the input parameter in aggregate functions that specifies the member set for aggregation.
- `member` is the current member of a hierarchy. When the hierarchy is not in the context of the block, the formula returns an empty value.
- Sibling members are members from the same level and with the same parent as `member`.

Examples

You have a time hierarchy and want to know the percentage of each Quarter within a year or the percentage of each year within the period.

```
=[Query 1].[Internet Sales].[Internet Sales Amount] / Sum([Query 1].[Internet
Sales].[Internet Sales Amount];{[Query 1].[Calendar].[Date.Calendar].Siblings()})
```

Date.Calendar	Internet Sales Amount	
⊟ All Periods	29,358,677.22	
⊟ CY 2001	3,266,373.66	11.13%
⊞ H2 CY 2001	3,266,373.66	100.00%
⊟ CY 2002	6,530,343.53	22.24%
⊞ H1 CY 2002	3,805,710.59	58.28%
⊞ H2 CY 2002	2,724,632.94	41.72%
⊟ CY 2003	9,791,060.3	33.35%
⊞ H1 CY 2003	3,037,501.36	31.02%
⊞ H2 CY 2003	6,753,558.94	68.98%
⊞ CY 2004	9,770,899.74	33.28%

In a free form cell you want to know the contribution of Year 2004 in the overall period :

```
=Sum([Query 1].[Internet Sales].[Internet Sales Amount];{[Query 1].[Calendar].
[Date.Calendar]&[All Periods].[CY 2004]}) / Sum([Query 1].[Internet Sales].
[Internet Sales Amount];{[Query 1].[Calendar].[Date.Calendar]&[All Periods].[CY
2004].Siblings()})
```

{CY 2001;CY 2002}	2004 percentage in 2001 to 2004 perdiod
9,796,717.18	33.28%

Date.Calendar	Internet Sales Amount	
⊟ All Periods	29,358,677.22	
⊟ CY 2001	3,266,373.66	11.13%
⊞ H2 CY 2001	3,266,373.66	100.00%
⊟ CY 2002	6,530,343.53	22.24%
⊞ H1 CY 2002	3,805,710.59	58.28%
⊞ H2 CY 2002	2,724,632.94	41.72%
⊟ CY 2003	9,791,060.3	33.35%
⊞ H1 CY 2003	3,037,501.36	31.02%
⊞ H2 CY 2003	6,753,558.94	68.98%
⊞ CY 2004	9,770,899.74	33.28%

Related Information

Aggregate [page 41]
Average [page 48]
Count [page 49]
Max [page 55]
Min [page 57]
Sum [page 77]

1.6.1.10 Misc functions

1.6.1.10.1 BlockName

Description

Returns the block name

Function Group

Misc

Syntax

```
string BlockName()
```

Examples

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

1.6.1.10.2 ClosingPeriod

Description

Returns the measure at the last date of the period in the current context and in the time range defined in the time dimension.

Function Group

Misc

Syntax

```
iinput_type ClosingPeriod(measure;timeperiod)
```

Input

Parameter	Description	Type	Required
measure	Any measure or a variable.	Measure	Yes
timeperiod	The time period that provides the calculation context in the block.	Time period	Yes

> **Note**
>
> - The time object must be a time period available in the block. If there is not the time period in the block, the function returns the #COMPUTATION error.
> - The report filters on the time dimension can impact the function results. You can combine the function with `NoFilter` function to ignore report filters in the function evaluation..

Examples

The ClosingPeriod column in the table below contains the following formula:

```
ClosingPeriod([Revenue]; [Time Dimension].[Year])
```

Year	Revenue	ClosingPeriod
2016	1000	2000
2017	2000	2000

```
ClosingPeriod([Revenue]; [Time Dimension].[Semester])
```

Year	Semester	Revenue	ClosingPeriod
2016	H1 2016	400	1500
2016	H2 2016	600	1500
2017	H1 2017	500	1500
2017	H2 2017	1500	1500

Related Information

#COMPUTATION [page 298]

1.6.1.10.3 ColumnNumber

Description

Returns the column number

Function Group

Misc

Syntax

```
int ColumnNumber()
```

Examples

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

1.6.1.10.4 Comment

Description

Returns the comment of a cell

Function Group

Misc

Syntax

```
string Comment()
```

Note

The comment returned by the function is either the first or last comment entered in the cell, depending on how you have set the parameter in the *Document Properties*.

Example

`Comment()` returns "Increase the gross margin in Q3" if the comment in the cell is "Increase the gross margin in Q3".

1.6.1.10.5 CurrentUser

Description

Returns the BI launch pad login of the current user

Function Group

Misc

Syntax

```
string CurrentUser()
```

Examples

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

1.6.1.10.6 CustomProperties

Description

Returns the list of defined custom properties.

Function Group

Misc

Syntax

```
string CustomProperties ()
```

> ⓘ Note
> - If no custom property exists in document, then the function returns the empty string.
> - If several custom properties exist in document, then they are separated by semicolons in the string returned by the function.

Examples

If the document contains two custom properties named Lines and Category, then,

`CustomProperties()` returns `"Lines;Category"`

1.6.1.10.7 CustomPropertyValue

Description

Returns the value of a defined custom property.

Function Group

Misc

Syntax

```
string CustomPropertyValue ( custom_property )
```

Input

Parameter	Description	Type	Required
`custom_property`	A custom property's name	String	Yes

> **Note**
> - If the custom property does not exist, then the function returns the empty string.

Examples

If the document contains two custom properties Lines whose value is Dresses and Category whose value is Skirts, then:

- `CustomPropertyValue("Lines")` return `"Dresses"`
- `CustomPropertyValue("Category")` return `"Skirts"`
- `CustomPropertyValue("Color")` return `""`

1.6.1.10.8 DescriptionOf

Description

Returns the description of an object.

Function Group

Misc

Syntax

```
string DescriptionOf(obj)
```

Input

Parameter	Description	Type	Required
obj	A report object	Report object	Yes

Notes

- If no description has been set, an empty string is returned.
- The description is returned in your preferred viewing locale. If the description has not been defined in your preferred viewing locale, the description may be returned in the fallback locale.

1.6.1.10.9 ForceMerge

Description

Includes synchronized dimensions in measure calculations when the dimensions are not in the measure's calculation context

Function Group

Misc

Syntax

```
num ForceMerge(measure)
```

Input

Parameter	Description	Type	Required
measure	Any measure	Measure	Yes

Output

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

Notes

- `ForceMerge` returns #MULTIVALUE if applied to a smart measure because the grouping set necessary to calculate the smart measure does not exist.
- `ForceMerge` is the equivalent of the BusinessObjects/Desktop Intelligence `Multicube` function.

Examples

`ForceMerge([Revenue])` returns the value of [Revenue], taking into account any synchronized dimensions that do not appear in the same block as the [Revenue] measure.

1.6.1.10.10 FormulaOf

Description

Returns the formula defining a variable if the object is a variable. If the object is not a variable, then it returns an empty string.

Function Group

Misc

Syntax

```
string FormulaOf(obj)
```

Input

Parameter	Description	Type	Required
obj	A report object	Report object	Yes

Notes

- The formula is returned in your preferred viewing locale.

1.6.1.10.11 GetContentLocale

Description

Returns the locale of the data contained in the document (the Document Locale)

Function Group

Misc

Syntax

```
string GetContentLocale()
```

Notes

The Document Locale is used to format the data in a document.

Examples

`GetContentLocale()` returns "fr_FR" if the Document Locale is "French (France)".

1.6.1.10.12 GetDominantPreferredViewingLocale

Description

Returns the dominant locale in the user's Preferred Viewing Locale group

Function Group

Misc

Syntax

```
string GetDominantPreferredViewingLocale()
```

Notes

- Each group of related locales has a dominant locale, used as a base for all the other locales in the group. For example, US English ("en_US") is the dominant locale in the English locales group. New Zealand English ("en_NZ") is also a member of this group.
- The *Translation Manager Guide* lists all the Dominant Preferred Viewing Locales.

Examples

`GetDominantPreferredViewingLocale` returns "en_US" when the Preferred Viewing Locale is "English (New Zealand)".

Related Information

GetPreferredViewingLocale [page 237]

1.6.1.10.13 GetLocale

Description

Returns the user's locale used to format the user interface (the Product Locale)

Function Group

Misc

Syntax

```
string GetLocale()
```

Notes

The Product Locale is the locale of the user interface (for example, menu items and button text).

Examples

`GetLocale()` returns "en_US" if the user's Product Locale is "English (US)".

1.6.1.10.14 GetLocalized

Description

Returns a string localized according to the user's Preferred Viewing Locale

Syntax

```
string GetLocalized(string[;comment])
```

Input

Parameter	Description	Type	Required
string	The string to be translated	string	Yes
comment	A comment to aid translators	string	No

Notes

- The `string` parameter can be a string in any formula (for example, in a cell, an alerter message or a variable definition).
- When designing a report, you can use the `comment` parameter to provide further information to help translators translate the string. The comment appears with the string in the Translation Manager tool which translators use to translate reports.
- Each `string` + `comment` pair generates a separate string to be translated in the Translation Manager tool. As a result, `GetLocalized("Product Total";"Max 20 characters")` and `GetLocalized("Product Total";"Use no more than 20 characters")` might return different translations.

Examples

`GetLocalized("Total for all products")` returns the French translation of "Total for all products" if the Preferred Viewing Locale is "fr_FR".

`GetLocalized("Total for all products";"Try not to use more than 20 characters")` returns the German translation of "Total for all products" if the Preferred Viewing Locale is "de_DE". The function also tells the translator of the report not to use more than 20 characters if possible when translating the string.

Related Information

GetPreferredViewingLocale [page 237]

1.6.1.10.15 GetPreferredViewingLocale

Description

Returns the user's preferred locale for viewing document data (the Preferred Viewing Locale)

Function Group

Misc

Syntax

```
string GetPreferredViewingLocale()
```

Examples

`GetPreferredViewingLocale` returns "en_US" if the Preferred Viewing Locale is "English (US)".

Related Information

GetLocalized [page 236]
GetDominantPreferredViewingLocale [page 235]

1.6.1.10.16 If...Then...Else

Description

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

Function Group

Misc

Syntax

```
If bool_value Then true_value [Else false_value]
```

Input

Parameter	Description	Type	Required
bool_value	A boolean value	Boolean	Yes
true_value	The value to return if `bool_value` is true	Any	Yes
false_value	The value to return if `bool_value` is false	Any	Yes if `Else` is included

Notes

- `true_value` and `false_value` can mix datatypes.
- You can use the boolean operators `And`, `Between`, `InList`, `Or` and `Not` with `If`.
- You can nest `If` conditions by replacing any `Else` clause with an `ElseIf` clause. The following syntax describes one level of nesting:

```
If bool_value Then true_value [ElseIf bool_value Then true_value Else
false_value…]
```

- The original syntax of the `If` function, `If(bool_value;true_value;false_value)`, is also supported.

Examples

`If [Sales Revenue] > 1000000 Then "High Revenue"` returns "High Revenue" for all rows whose revenue is larger than 1,000,000 and nothing for all other rows.

`If [Sales Revenue] > 1000000 Then "High Revenue" Else [Revenue]` returns "High Revenue" for all rows whose revenue is larger than 1,000,000 and the revenue value for all other rows.

`If [Sales Revenue] > 1000000 Then "High Revenue" Else "Low Revenue"` returns "High Revenue" for all rows whose revenue is larger than 1,000,000 and "Low Revenue" for all rows whose revenue is less than 1,000,000.

`If [Sales Revenue] > 1000000 Then "High Revenue" ElseIf [Sales Revenue] > 800000 Then "Medium Revenue" Else "Low Revenue"` returns "High Revenue" for all rows whose revenue is larger than 1000000, "Medium Revenue" for all rows whose revenue is between 800000 and 1000000, and "Low Revenue" for all other rows.

Related Information

If [page 240]
And operator [page 262]

Between operator [page 263]
InList operator [page 264]
Or operator [page 262]
Not operator [page 263]

1.6.1.10.17 If

Description

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

Function Group

Misc

Syntax

```
If(bool_value;true_value;false_value)
```

Input

Parameter	Description	Type	Required
bool_value	A boolean value	Boolean	Yes
true_value	The value to return if `bool_value` is true	Any	Yes
false_value	The value to return if `bool_value` is false	Any	Yes

Notes

- `true_value` and `false_value` can mix datatypes.
- You can nest `If` conditions by replacing `false_value` with additional `If` conditions. This syntax shows one level of nesting:

  ```
  If(bool_value;true_value;If(bool_value;true_value;false_value);false_value)
  ```

- The `If...Then...Else` syntax is also supported.

Examples

`If([Sales Revenue]>1000000;"High Revenue";"Low Revenue")` returns "High Revenue" for all rows whose revenue is larger than 1,000,000 and "Low Revenue" for all rows whose revenue is less than 1,000,000.

`If([Sales Revenue]>1000000;"High Revenue";[Revenue])` returns "High Revenue" for all rows whose revenue is larger than 1,000,000 and the revenue value for all other rows.

Related Information

If...Then...Else [page 238]

1.6.1.10.18 LineNumber

Description

Returns the line number in a table

Function Group

Misc

Syntax

```
int LineNumber()
```

Notes

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

Examples

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

1.6.1.10.19 Next

Description

Returns the next value of an object.

Function Group

Misc

Syntax

```
input_type Next(dimension|measure [;Row|Col][;reset_dims][;offset][;NoNull])
```

Input

Parameter	Description	Type	Required
dimension/measure	This is the dimension or measure whose next value the function returns.	Dimension or measure	Yes
Row/col	Sets the calculation direction. It is used in a cross-table to define if the next returned value is the one in row or column.	Keyword	No
reset_dims	This is the list of dimensions used to reset the calculation.	Dimension list	No
offset	Returns the next value that is `offset` rows after the current row.	Integer	No (default is 1)

Parameter	Description	Type	Required
NoNull	Tells the function to return the first non-null value starting from the offset.	Keyword	No

Notes

- `Next` isn't compatible with display dependent functions like `ColumnNumber`, `LineNumber`, `PageNumber`, `Page`, and `PageInSection`. Using a combination of this functions generates a `#RECURSIVE` error. As a workaround, replace the `ColumnNumber` and `LineNumber` functions with a running sum. There's no workaround for `PageNumber`, `Page`, and `PageInSection`.
- The default value of `offset` is 1. `Next([Revenue];1)` and `Next([Revenue])` are functionally the same.
- When you include the `NoNull` argument, the function returns the first non-null value of the object beginning from the cell `offset` rows before the current row and counting backwards.
- You can use extended syntax context operators with `Next`.
- 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.
- `Next` is applied after all report, section and block filters, and all sorts, are applied.
- You can't apply sorts or filters on formulas that use `Next`.
- If `Next` is applied on a measure and the measure returns an undefined value, Next returns an undefined value even if the previous line returned a value.
- `Next` ignores breaks when placed outside a break header or footer.
- `Next` returns the value in the previous instance of the footer when placed in a break footer.
- `Next` is reset in each report section.
- When used in a crosstab, `Next` does not treat the last value in a row as the previous value of the first value of the next row.

1.6.1.10.20 NoFilter

Description

Ignores filters when calculating a value. NoFilter is used with measure objects. It does not apply to dimensions.

Function Group

Misc

Syntax

```
input_type NoFilter(obj[;All|Drill])
```

Input

Parameter	Description	Type	Required
obj	Any report object	Report object	Yes
All|Drill	• No keyword specified - ignore report and block filters • `All` - ignore all filters • `Drill` - ignore report and drill filters	Keyword	No

Notes

- `NoFilter(obj;Drill)` does not work in query drill mode because the drill filters are added to the query rather than applied to the report data.
- If you end drill mode with drill filters applied, the drill filters become report filters and can change the value of any objects to which `NoFilter(obj;Drill)` is applied.

Examples

When placed in a block footer, `NoFilter(Sum([Sales Revenue]))` returns the total sales revenue of all possible rows in the block, even when rows are filtered out of the block.

`NoFilter(Sum([Sales Revenue]);All)` returns the sum of the sales revenue for all countries including France, even though there is a filter that excludes France from the report.

`NoFilter(Sum([Sales Revenue]);Drill)` returns the sum of the sales revenue for all countries, even when there is a drill filter on the [Country] dimension.

1.6.1.10.21 NumberOfPages

Description

Returns the number of pages in a report

Function Group

Misc

Syntax

```
integer NumberOfPages()
```

Notes

If you place the `NumberOfPages` function in a cell whose Autofit Height or Autofit Width properties are set, the cell returns #RECURSIVE because the placing of this formula in an Autofit cell creates a circular dependency. This function needs the exact size of the report to return a value, but the size of the cell, which affects the size of the report, is determined by the cell content.

Examples

`NumberOfDataPages()` returns 2 if the report has two pages.

1.6.1.10.22 OpeningPeriod

Description

Returns the measure at the first date of the period in the current context and in the time range defined in the time dimension.

Function Group

Misc

Syntax

```
input_type OpeningPeriod(measure;timeperiod)
```

Input

Parameter	Description	Type	Required
measure	Any measure or a variable.	Measure	Yes
timeperiod	The time period that provides the calculation context in the block.	Time period	Yes

> **Note**
>
> - The time object must be a time period available in the block. If there is not the time period in the block, the function returns the #COMPUTATION error.
> - The report filters on the time dimension can impact the function results. You can combine the function with `NoFilter` function to ignore report filters in the function evaluation.

Examples

The OpeningPeriod column in the table below contains the following formula:

```
OpeningPeriod([Revenue]; [Time Dimension].[Year])
```

Year	Revenue	OpeningPeriod
2016	1000	1000
2017	2000	1000

```
OpeningPeriod([Revenue]; [Time Dimension].[Semester])
```

Year	Semester	Revenue	OpeningPeriod
2016	H1 2016	400	400
2016	H2 2016	600	400
2017	H1 2017	500	400
2017	H2 2017	1500	400

Related Information

#COMPUTATION [page 298]

1.6.1.10.23 Page

Description

Returns the current page number in a report

Function Group

Misc

Syntax

```
integer Page()
```

Notes

If you place the `Page` function in a cell whose Autofit Height or Autofit Width properties are set, the cell returns #RECURSIVE because the placing of this formula in an Autofit cell creates a circular dependency. This function needs the exact size of the report to return a value, but the size of the cell, which affects the size of the report, is determined by the cell content.

Example

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

1.6.1.10.24 PageInSection

Description

Returns the page number within the current section instance in a specified section

Function Group

Misc

Syntax

```
integer PageInSection([section_level)]
```

Input

Parameter	Description	Type	Required
section_level	Hierarchical level of the section	integer	No

Notes

- If you place the `PageInSection` function in a cell whose Autofit Height or Autofit Width properties are set, the cell returns #RECURSIVE because the placing of this formula in an Autofit cell creates a circular dependency. This function needs the exact size of the report to return a value, but the size of the cell, which affects the size of the report, is determined by the cell content.
- `PageInSection()` must be within a section instance. Otherwise, it returns 0.
- The section hierarchical levels start at 1 (top level).
- If `section_level` is not specified, the function returns the number in the current section level.
- If you specifiy a section level that does not exist, the function returns 0.
- Variables created with the function must be measures.

Examples

In a document with a section on Year (= 2010, 2011, 2012) and a sub-section on State (= California, Florida, Texas):

- `PageInSection(1)` repeated in the Year section returns 2 on the second page of 2010, 2011 and 2012.
- `PageInSection(2)` repeated in the State sub-section returns 1 on the first page of California, Florida and Texas.

1.6.1.10.25 ParallelPeriod

Description

Returns the measure at the date of a period parallel to the dates present in the block in the current context. Selected dates are shifted a number of intervals either forward in time or back in time.

Function Group

Misc

Syntax

```
input_type ParallelPeriod(measure;timeperiod;offset)
```

Input

Parameter	Description	Type	Required
measure	Any measure or a variable.	Measure	Yes
timeperiod	The time period that provides the calculation context in the block.	Time period	Yes
offset	Specifies the number of intervals either forward in time or back in time	Integer	Yes

ⓘ Note

- The time object must be a time period available in the block. If there is not the time period in the block, the function returns the #COMPUTATION error.
- The function is not dependent on the time dimension sort order in the block.
- The report filters on the time dimension can impact the function results. You can combine the function with `NoFilter` function to ignore report filters in the function evaluation.

Examples

The ParallelPeriod column in the table below contains the following formula:

```
ParallelPeriod([Revenue]; [Time Dimension].[Year];-1
```

Year	Revenue	ClosingPeriod
2015	600	-
2016	1000	600
2017	2000	2000

```
ParallelPeriod ([Revenue]; [Time Dimension].[Semester];-1)
```

Year	Semester	Revenue	ParallelPeriod
2015	H1 2015	200	
2015	H2 2015	400	200
2016	H1 2016	400	400
2016	H2 2016	600	400
2017	H1 2017	500	600
2017	H2 2017	1500	500

Related Information

#COMPUTATION [page 298]

1.6.1.10.26 PeriodToDate

Description

Returns the running total of the values of the measure over the time period to date, in the current context. For example, user might select Year to see the year-to-date (YTD) values for each month next to the monthly values.

Function Group

Misc

Syntax

```
input_type PeriodToDate(measure;timeperiod;[Sum|Max|Min|Count|Average|Product])
```

Input

Parameter	Description	Type	Required
measure	Any measure or a variable.	Measure	Yes
timeperiod	The time period that provides the calculation context in the block.	Time period	Yes
runningfunct	Running function to apply. Possible values: • Sum (default) • Max • Min • Count • Average • Product	Enum	No

> **Note**
>
> - The time object must be a time period available in the block. If there is not the time period in the block, the function returns the #COMPUTATION error.
> - If the block contains other dimensions, these last ones will be implicitly used as reset dimensions for the running function.
> - If the block contains other time periods that are different to those defined in the function, the running total will be applied on the time period having the lowest time granularity.
> - The function is not dependent on the time dimension sort order in the block.
> - The report filters on the time dimension can impact the function results. You can combine the function with `NoFilter` function to ignore report filters in the function evaluation.

Examples

The PeriodToDate column in the table below contains the following formula

```
PeriodToDate([Revenue]; [Time Dimension].[Year])
```

Year	Semester	Revenue	ClosingPeriod
2015	H1 2015	200	200
2015	H2 2015	400	600
2016	H1 2016	400	400
2016	H2 2016	600	1000
2017	H1 2017	500	500
2017	H2 2017	1500	2000

Year	Semester	Product Family	Revenue	PeriodToDate
2015	H1 2015	Foods	50	50
2015	H1 2015	Electronics	150	150
2015	H2 2015	Foods	100	150
2015	H2 2015	Electronics	300	450
2016	H1 2016	Foods	150	150
2016	H1 2016	Electronics	250	250
2016	H2 2016	Foods	200	350
2016	H2 2016	Electronics	400	650
2017	H1 2017	Foods	200	200
2017	H1 2017	Electronics	300	300
2017	H2 2017	Foods	500	700
2017	H2 2017	Electronics	1000	1300

Related Information

#COMPUTATION [page 298]

1.6.1.10.27 Previous

Description

Returns a previous value of an object

Function Group

Misc

Syntax

```
input_type Previous(dimension|measure|Self [;Row|col][;(reset_dims)][;offset]
[;NoNull])
```

Input

Parameter	Description	Type	Required
dimension|measure|Self	The dimension or measure whose previous value the function returns, or the Self keyword	Dimension, measure or keyword	Yes
Row/Col	Sets the calculation direction	Keyword	No
reset_dims	The list of dimensions used to reset the calculation	Dimension list	No
offset	Specifies the value of `dimension` or `measure` that is `offset` rows previous to the current row	Integer	No (default is 1)
NoNull	Tells the function to return the first non-null value starting from the offset	Keyword	No

Notes

- `Previous` isn't compatible with display dependent functions like `ColumnNumber`, `LineNumber`, `PageNumber`, `Page` and `PageInSection`. Using a combination of this functions generates a `#RECURSIVE` error. As a workaround, replace the `ColumnNumber` and `LineNumber` functions with a running sum. There's no workaround for `PageNumber`, `Page` and `PageInSection`.
- The default value of `offset` is 1. `Previous([Revenue];1)` and `Previous([Revenue])` are functionally the same.
- When you include the `NoNull` argument, the function returns the first non-null value of the object beginning from the cell `offset` rows before the current row and counting backwards.
- You can use extended syntax context operators with `Previous`.
- The `Self` operator allows you to refer to the previous value of a cell when it contains content other than one report object.
- You must always place dimensions in parentheses even if there is only one dimension in the list of reset dimensions.
- When you specify a set of reset dimensions you must separate them with semi-colons.
- `Previous` is applied after all report, section and block filters, and all sorts, are applied.
- You can't apply sorts or filters on formulas that use `Previous`.

- If `Previous` is applied on a measure and the measure returns an undefined value, `Previous` returns an undefined value even if the previous line returned a value.
- `Previous` ignores breaks when placed outside a break header or footer.
- `Previous` returns the value in the previous instance of the footer when placed in a break footer.
- `Previous` is reset in each report section.
- When used in a crosstab, `Previous` does not treat the last value in a row as the previous value of the first value of the next row.

Examples

`Previous([Country];1)` returns the following values in the following table:

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

`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		2,500,000	2,000,000
France	3,000,000		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

`Previous([Revenue]);2;NoNull)` returns the following values in the following table:

Year	Quarter	Revenue	Previous
2008	Q1	500	
2008	Q2		
2008	Q3	400	500
2008	Q4	700	500
2008	Q1	300	400
2008	Q2		700
2008	Q3		300
2008	Q4	200	300

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

Related Information

Comparing values using the Previous function [page 304]
Self operator [page 273]

1.6.1.10.28 RefValue

Description

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

Function Group

Misc

Syntax

```
input_type RefValue(obj)
```

Examples

`RefValue([Top Performing Region])` returns "South West" if the value of the [Top Performing Region] variable is "South West" in the reference data.

`RefValue([Revenue])` returns 1000 if the value of the [Revenue] measure is 1000 in the reference data.

Notes

- The RefValue() function can be used with either a measure or a dimension object. However, when used in a variable qualified as a dimension or a detail, the RefValue() function will return the current values of that object, rather than its reference values. In order to get the reference values, the variable must be qualified as a measure.
- When created directly in a section, table, form, or chart, a formula will always be qualified as a measure, so if the formula uses the RefValue() function, it will return the expected reference values.

Example of RefValue function with a variable

We have the list of values for the [State] dimension: California, Florida, Texas and New York. After a data refresh, this list becomes: Arizona, California, Florida, Texas and New York. A variable such as Variable=RefValue([State]) will either return:

Variable is qualified as	List of values returned is
Dimension or detail	Arizona, California, Florida, Texas and New York
Measure	(null value), California, Florida, Texas and New York

1.6.1.10.29 RelativeValue

Description

Returns previous or subsequent values of an object

Function Group

Misc

Syntax

```
input_type RelativeValue(measure|detail;slicing_dims;offset)
```

Input

Parameter	Description	Type	Required
measure|detail	Any measure or a detail of a dimension in the block	Measure or detail	Yes
slicing_dims	The dimensions that provide the calculation context	Dimension list	Yes
offset	Specifies the value of `measure` or `detail` that is `offset` rows removed from the current row	Integer	Yes

Notes

- The object must be a measure or a detail of a dimension available in the block.
- The sort order of the list of values of the slicing dimensions is used to determine the output of the function. The sort order is determined by two factors: sorts applied to the slicing dimensions, and the order in which the slicing dimensions are listed in the function.
- A dimension used as a section lead can be specified as a slicing dimension.
- All the slicing dimensions must be present in the block or section cell of the block in which the function is placed. If a slicing dimension is later removed from the block, the function returns the #COMPUTATION error.

- If the offset exceeds the number of rows in the list of values of the slicing dimension, the function returns null.
- `RelativeValue` cannot be used recursively.
- You must always place dimensions in parentheses even if there is only one dimension in the list of slicing dimensions.

Examples

The RelativeValue column in the table below contains the following formula:

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

Year	Quarter	Sales Person	Revenue	RelativeValue
2007	Q1	Smith	1000	
2007	Q2	Jones	2000	
2007	Q3	Wilson	1500	
2007	Q4	Harris	3000	
2008	Q1	Smith	4000	1000
2008	Q2	Jones	3400	2000
2008	Q3	Wilson	2000	1500
2008	Q4	Harris	1700	3000

Related Information

#COMPUTATION [page 298]
Comparing values using the RelativeValue function [page 305]

1.6.1.10.30 ReportName

Description

Returns the name of a report

Function Group

Misc

Syntax

```
string ReportName()
```

Examples

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

1.6.1.10.31 RowIndex

Description

Returns the number of a row

Function Group

Misc

Syntax

```
integer RowIndex()
```

Notes

- Row numbering starts at 0.
- `RowIndex` returns #MULTIVALUE when placed in a table header or footer.

Examples

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

1.6.1.10.32 UniqueNameOf

Description

Returns the unique name of an object

Function Group

Misc

Syntax

```
string UniqueNameOf(obj)
```

Input

Parameter	Description	Type	Required
obj	Any report object	Report object	Yes

Examples

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

1.6.2 Function and formula operators

Operators link the various components in a formula.

Formulas can contain mathematical, conditional, logical, function-specific or extended syntax operators.

1.6.2.1 Mathematical operators

Mathematical operators are familiar from everyday arithmetic.

There are addition (+), subtraction (-), multiplication (*), division (/) operators that allow you to perform mathematical operations in a formula. The formula `[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".

1.6.2.2 Conditional operators

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

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 Then "High" Else "Low"
```

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

1.6.2.3 Logical operators

The logical operators are `And`, `Or`, `Not`, `Between` and `InList`.

Logical operators are used in boolean expressions, which return `True` or `False`.

1.6.2.3.1 And operator

The And operator links boolean values.

Description

If all the boolean values linked by And return true, the combination of all the values also returns true.

Syntax

```
bool_value And bool_value [And bool_value...]
```

Examples

`If [Resort] = "Bahamas Beach" And [Revenue]>100000 Then "High Bahamas Revenue"` returns "High Bahamas Revenue" if [Resort] = "Bahamas Beach" And [Revenue]>100000.

1.6.2.3.2 Or operator

The `Or` operator links boolean values.

Description

If any one boolean value linked by `Or` returns true, the combination of all the values also returns true.

Syntax

```
bool_value Or bool_value [Or bool_value...]
```

Examples

`If [Resort] = "Bahamas Beach" Or [Resort]="Hawaiian Club" Then "US" Else "France"` returns "US" if [Resort]="Bahamas Beach" or "Hawaiian Club", or "France" otherwise.

1.6.2.3.3 Not operator

Description

The `Not` operator returns the opposite of a boolean value.

Syntax

```
bool Not(bool_value)
```

Examples

`If Not([Country] = "US") Then "Not US"` returns "Not US" if [Country] has any value other than "US".

1.6.2.3.4 Between operator

Description

The `Between` operator determines if a variable is between two values.

Syntax

```
bool Between(first_value;second_value)
```

Notes

- You use `Between` with the `If` function and the `Where` operator.
- Changing the document locale can impact the result returned by the Between operator.

Examples

`If [Sales revenue] Between(800000;900000) Then "Medium revenue"` returns "Medium revenue" if [Sales revenue] is between 800000 and 900000.

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

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

Related Information

If...Then...Else [page 238]
Where operator [page 274]

1.6.2.3.5 InList operator

Description

The `InList` operator determines if a value is in a list of values.

Syntax

```
bool test_value InList(value_list)
```

Notes

It is the combination of `test_value` + `InList` that returns a boolean value, not `InList` alone.

Examples

`If Not ([Country] InList("England";"Scotland";"Wales")) Then "Not Britain" Else "Britain"` returns "Not Britain" if [Country] is not equal to "England", "Scotland" or "Wales", or "Britain" otherwise.

`If [Resort] InList("Bahamas Beach";"Hawaiian Club") Then "US Resort"` returns "US Resort" if [Resort] is equal to "Bahamas Beach" or "Hawaiian Club".

Related Information

If...Then...Else [page 238]
Where operator [page 274]

1.6.2.4 Function-specific operators

Some functions can take specific operators as arguments.

For example, the `Previous` function can take the `Self` operator.

All functions use `)` and `(` to enclose function arguments. Functions that accept multiple parameters use `;` to separate the parameters.

1.6.2.4.1 All operator

The `All` operator tells the `NoFilter` function to ignore all filters.

The `All` operator can also tell the `Count` function to count all values, including duplicates.

Related Information

Count [page 49]
Distinct/All operators [page 268]
NoFilter [page 243]
All/Drill operators [page 266]

1.6.2.4.2 All/Drill operators

The `All/Drill` operators work with the `NoFilter` function.

Description

The `All/Drill` operators determine which filters the `NoFilter` function ignores.

- Not specified - `NoFilter` ignores report and block filters
- All - `NoFilter` ignores all filters
- Drill - `NoFilter` ignores report filters and drill filters

1.6.2.4.3 Ascending

The `Ascending` operator is an argument of the `PromptSummary` function.

Description

When set, the `PromptSummary` function sorts the prompts in the ascending order.

Related Information

PromptSummary [page 160]

1.6.2.4.4 Bottom/Top operators

The `Bottom/Top` operators work with the `Rank` function.

Description

The `Bottom/Top` operators tell the `Rank` function to rank in descending or ascending order.

- `Top` - ranks in descending order
- `Bottom` - ranks in ascending order

Examples

`Rank([Revenue];([Country]);Top)` ranks countries by revenue from highest to lowest.

Related Information

Rank [page 191]

1.6.2.4.5 Break operator

The `Break` operator works with the `Percentage` function.

Description

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

Examples

The formula `Percentage([Revenue])` gives the following result in the following table (percentages are calculated on the total revenue in the block):

Year	Quarter	Revenue	Percentage
2005	Q1	10000	10%
2005	Q2	20000	20%
2006	Q1	30000	30%
2006	Q2	40000	40%

The formula `Percentage([Revenue];Break)` gives the following result in the following table (percentages are calculated on the total revenue in each part of the block):

Year	Quarter	Revenue	Percentage
2005	Q1	10000	33.3%
2005	Q2	20000	66.6%

2006	Q1	30000	42.9%
2006	Q2	40000	57.1%

Related Information

Percentage [page 60]

1.6.2.4.6 Descending

The `Descending` operator is an argument of the `PromptSummary` function.

Description

When set, the `PromptSummary` function sorts the prompts in the descending order.

Related Information

PromptSummary [page 160]

1.6.2.4.7 Distinct/All operators

The `Distinct/All` operators work with the `Count` function.

The `Distinct/All` operators tell the `Count` function to count distinct values only, or all values.

Examples

`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 Information

Count [page 49]

1.6.2.4.8 IncludeEmpty operator

The `IncludeEmpty` operator works with aggregate functions.

Description

The `IncludeEmpty` operator tells some aggregate functions (Average, Count, RunningAverage, RunningCount) to include empty values in calculations.

Examples

`Average([Revenue];IncludeEmpty)` returns 3 if [Revenue] has the values (5;3;<empty>;4).

Related Information

Average [page 48]
Count [page 49]
RunningAverage [page 63]
RunningCount [page 65]

1.6.2.4.9 Index operator

The `Index` operator works with the `UserResponse` and `RefValueUserResponse` functions.

Description

The `Index` operator tells the `UserResponse` and `RefValueUserResponse` functions to return the database primary key of the prompt response.

Related Information

UserResponse [page 147]
RefValueUserReponse [page 144]

1.6.2.4.10 Linear operator

The `Linear` operator works with the `Interpolation` function.

Description

The `Linear` operator tells the `Interpolation` function to use linear regression with least squares interpolation to supply missing measure values.

Linear regression with least squares interpolation calculates missing values by calculating a line equation in the form f(x) = ax + b that passes as closely as possible through all the available values of the measure.

Related Information

Interpolation [page 52]

1.6.2.4.11 NoNull operator

The `NoNull` operator works with the `Previous` function.

Description

The `NoNull` operator tells the `Previous` function to ignore null values.

When used with `NoNull`, `Previous` returns the first non-null value of the object, beginning from the cell `offset` rows before the current row and counting backwards.

Related Information

Previous [page 252]

1.6.2.4.12 NotOnBreak operator

The `NotOnBreak` operator works with the `Interpolation` function.

Description

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

Related Information

Interpolation [page 52]

1.6.2.4.13 PointToPoint operator

The `PointToPoint` operator tells the `Interpolation` function to use point-to-point interpolation to supply missing measure values.

Description

Point-to point interpolation calculates missing values by calculating a line equation in the form f(x) = ax + b that passes through the two adjacent values of the missing value.

Related Information

Interpolation [page 52]

1.6.2.4.14 Row/Col operators

The `Row` operator calculates each value in the row as a percentage of the total value of all the rows in the embedding context. The `Col` operator calculates each value in the column as a percentage of the total value of all the columns in the embedding context.

Description

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

Notes

In a crosstab, the value in each cell is calculated by default as a percentage of the total value in the crosstab. The `Row` operator calculates the values in the rows as percentages of the total value for the row. The `Col` operator calculates the values in the columns as percentages of the total value in the column.

Examples

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

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%

The Row operator calculates the running aggregate by row. The Col operator calculates the running aggregate by column.

In a crosstab, `RunningSum([Measure])` or `RunningSum([Measure];Row)` gives the following result:

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

In a crosstab, `RunningSum([Measure];Col)` gives the following result:

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

Related Information

Percentage [page 60]
RunningAverage [page 63]
RunningCount [page 65]
RunningMax [page 68]
RunningMin [page 70]
RunningProduct [page 72]
RunningSum [page 73]

1.6.2.4.15 Self operator

The `Self` operator works with the `Previous` function.

Description

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

Examples

`5 + Previous(Self)` returns the sequence 5, 10, 15, 20, 25, 30...

`1 + 0.5 * Previous(Self)` returns the sequence 1, 1.5, 1.75, 1.88...

Related Information

Previous [page 252]

1.6.2.4.16 Where operator

Description

The `Where` operator restricts the data used to calculate a measure.

Examples

The formula `Average ([Sales Revenue]) Where ([Country] = "US")` calculates the average sales where the country is "US".

The formula `Average ([Sales Revenue]) Where ([Country] = "US" Or [Country] = "France")` calculates the average sales where the country is "US" or "France".

The formula `[Revenue] Where (Not ([Country] Inlist ("US"; "France")))` calculates the revenue for the countries other than US and France.

The variable [High Revenue] has the formula `[Revenue] Where [Revenue > 500000]`. When placed in a block, [High Revenue] displays either the revenue when its value is greater than 500000, or nothing. When placed in a footer at the bottom of the [High Revenue] column, the formula `Average ([High Revenue])` returns the average of all the revenues greater than 500000.

Related Information

And operator [page 262]
Between operator [page 263]
InList operator [page 264]
Or operator [page 262]
Not operator [page 263]

1.6.2.5 Extended syntax 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.

1.6.2.5.1 In context operator

The `In context` operator specifies dimensions explicitly in a context.

Example: Using In to specify the dimensions in a context

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

Year	Sales revenue	Max Quarterly Revenue
2001	$8,096,123.60	$2,660,699.50
2002	$13,232,246.00	$4,186,120.00
2003	$15,059,142.80	$4,006,717.50

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
2001	Q1	$2,660,699.50
2001	Q2	$2,279,003.00
2001	Q3	$1,367,841.00
2001	Q4	$1,788,580.00
	Max:	$2,660,699.50

Year	Quarter	Sales revenue
	Q1	$3,326,172.00
	Q2	$2,840,651.00
	Q3	$2,879,303.00
	Q4	$4,186,120.00

Year	Quarter	Sales revenue
	Max:	$4,186,120.00

Year	Quarter	Sales revenue
	Q1	$3,742,989.00
	Q2	$4,006,717.50
	Q3	$3,953,395.00
	Q4	$3,356,041.00
	Max:	$4,006,717.50

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

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

1.6.2.5.2 ForEach context operator

The `ForEach` operator adds dimensions to a context.

Example: Using ForEach to add dimensions to a context

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

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]).

1.6.2.5.3 ForAll context operator

The `ForAll` context operator removes dimensions from a context.

Example: Using ForAll to remove dimensions from a context

You have a report showing Year, Quarter and Sales revenue and you want to add a column that shows the total revenue in each year, as shown in the following block:

Year	Quarter	Sales revenue	Yearly Revenue
2004	Q1	$2,660,700	$8,096,124
2004	Q2	$2,279,003	$8,096,124
2004	Q3	$1,367,841	$8,096,124
2004	Q4	$1,788,580	$8,096,124
2005	Q1	$3,326,172	$13,232,246
2005	Q2	$2,840,651	$13,232,246
2005	Q3	$2,879,303	$13,232,246
2005	Q4	$4,186,120	$13,232,246
2006	Q1	$3,742,989	$15,059,143
2006	Q2	$4,006,718	$15,059,143
2006	Q3	$3,953,395	$15,059,143
2006	Q4	$3,356,041	$15,059,143

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.

1.6.2.6 Set operators

Set operators work on members in hierarchical data.

1.6.2.6.1 Range operator

Description

The range operator (:) returns a set of members between and including two members at the same level

Syntax

```
first_member:last_member
```

Examples

`[Geography]&[US].[California].[Los Angeles]:[Geography]&[US].[California].[San Francisco]` returns [Los Angeles], [San Diego], [San Francisco] if the members at the level are in the order ...[Los Angeles], [San Diego], San Francisco]...

`Sum([Revenue];{[Geography]&[US].[California].[Los Angeles]:[Geography]&[US].[California].[San Francisco]})` returns the total revenue for Los Angeles, San Diego and San Francisco.

1.6.3 Extended syntax keywords

Extended syntax keywords are a form of shorthand that allows you to refer to dimensions in extended syntax without specifying those dimensions explicitly.

These keywords help 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.

1.6.3.1 The Block keyword

This topic 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 uses the following formula:

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

The First Half Average column uses the following formula:

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

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

1.6.3.2 The Body keyword

This topic describes the dimensions referenced by the keyword in a block Body, depending on where it is placed in a report.

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

1.6.3.3 The Break keyword

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

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	$8,096,124
	Q2	$2,279,003	$8,096,124
	Q3	$1,367,841	$8,096,124
	Q4	$1,788,580	$8,096,124

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]).

1.6.3.4 The Report keyword

This topic 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 as follows:

```
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]).

1.6.3.5 The Section keyword

This topic describes the data referenced by the keyword for a Section, 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]).

1.6.4 Rounding and truncating numbers

Several functions contain a parameter that determines to what level the function rounds or truncates the value it returns.

This parameter accepts an integer that is either greater than 0, 0, or less than 0. The following table explains how numbers are rounded and truncated in these cases:

Parameter	Description
> 0	The function rounds/truncates to `<parameter>` decimal places. Examples: `Round(3.13;1)` returns 3.1 `Round(3.157;2)` returns 3.16
0	The function rounds/truncates to the nearest integer. Examples: `Truncate(3.7;0)` returns 3 `Truncate(4.164;0)` returns 4
< 0	The function rounds/truncates to the nearest 10 (parameter = -1), 100 (parameter = -2), 1000 (parameter = -3) and so on. Examples:

Parameter	Description
	Round(123.76;-1) returns 120 Round(459.9;-2) returns 500 Truncate(1600;-3) returns 1000

> **Note**
>
> Numbers are represented internally using double-precision floating-point formats and have 15 to 17 digits of precision.

Related Information

Round [page 193]
Truncate [page 199]
EuroConvertTo [page 179]
EuroConvertFrom [page 178]
EuroFromRoundError [page 181]
EuroToRoundError [page 183]

1.6.5 Referring to members and member sets in hierarchies

You refer to members and member sets in functions using the syntax `[hierarchy]&path.function`.

The `path` and `function` parts are optional. In `path`, you refer to each member in square brackets, with members separated by full stops. The names of members and levels are case-sensitive.

> **Note**
>
> You use member sets to override the default calculation context for a hierarchy. In functions that accept member sets, you enclose the member set in {}.

You refer to ranges of members using a colon (:) between the start and end member, and with the full path specified for each member. A range includes all members at the same level as the specified members.

An example of range syntax is: `[Sales Hierarchy]&[Customer_Type].[ENTERPRISE];[Large].[Nancy Davolio]: [Sales Hierarchy]&[Customer_Type].[ENTERPRISE];[Large].[Andrew Smith]`.

Example: Referring to members and member sets

You have the following hierarchy:

Sales Hierarchy				Order Amount
Customer_Type				277,290,434
	ENTERPRISE			180,063,361
		Large		113.905,997
			Nancy Davolio	44,855,689
			Janet Leverling	44,050,308
			Andrew Smith	30,000,000
	GLOBAL			91,157,363

- `[Sales Hierarchy]&[Customer_Type].[ENTERPRISE].[Large].Children` refers to the [Nancy Davolio], [Janet Leverling] and [Andrew Smith] members.
- `Sum([Order Amount];{[Sales Hierarchy]&[Customer_Type].[ENTERPRISE].[Large].children})` returns 113,905,997 (the sum of the measure for the three child members).
- `[Sales Hierarchy]&[Customer_Type].[ENTERPRISE].[Large].[Janet Leverling]` refers to the `[Janet Leverling]` member.
- `Sum([Order Amount];{[Sales Hierarchy]&[Customer_Type].[ENTERPRISE].[Large].[Janet Leverling];[Sales Hierarchy]&[Customer_Type].[ENTERPRISE].[Large].[Nancy Davolio]})` returns 88,905,997 (the sum of the measure for the two members).
- `[Sales Hierarchy]&[Customer_Type].[ENTERPRISE].[Large].[Nancy Davolio]:[Sales Hierarchy]&[Customer_Type].[ENTERPRISE].[Large].[Andrew Smith]` refers to the [Nancy Davolio], [Janet Leverling] and [Andrew Smith] members.
- `Sum([Order Amount];{[Sales Hierarchy]&[Customer_Type].[ENTERPRISE].[Large].[Nancy Davolio]:[Sales Hierarchy]&[Customer_Type].[ENTERPRISE].[Large].[Andrew Smith]})` returns 113,905,997 (the sum of the measure for the three members in the range).
- `[Sales Hierarchy].children` refers to all members in the [Sales Hierarchy] hierarchy.
- `Sum([Order Amount];{Sales Hierarchy].children})` returns 277,290.434.

1.7 Building custom functions

1.7.1 Overview of external functions

Calculation extensions are custom Web Intelligence reporting calculations that enhance the list of existing Web Intelligence functions.

To use the Calculation Extension Library, create a C++ external library following a specific API.

1.7.1.1 External functions

External functions are visible and usable like the other Web Intelligence standard functions. You can build a formula with functions that implement your own logic.

> **Note**
>
> You can define as many functions as you need. Only functions that use single value parameters are supported. You can have a maximum of five single value parameters.

To define a function:

1. Declare in an XML file the description of the external function using a given XML structure.
2. Implement the function in a C++ library using a given API.
3. Copy the XML file and library to the appropriate folder in your Business Objects Enterprise installation directory folder for the server and the desktop client.
4. Restart the system to automatically add the external function to the list of the functions available for creating formulas.

The external function is based on a unique identifier so that when it is used in a report, it cannot be misinterpreted in case of using a different external library.

If the system cannot load a library or is missing information for an external function, has an inconsistent XML declaration, missing library, or duplicated function, an error message appears. The system also writes errors in the trace log.

Related Information

#EXTERNAL error message [page 296]

1.7.1.2 Deploying the custom functions

Deployment of custom functions requires a few manual steps. The BusinessObjects administrator must place the XML file and related library DLL file in the library folder for the server, as well as on every machine where desktop rich-client is installed.

> **Caution**
>
> Replacing or adding a library in the custom library folder can represent a threat to the system. Since the library is automatically loaded, an external library can access internal critical data or processes, putting the system in danger.
>
> Make sure that the site administrator implements the appropriate security access to the related folder, so that only authorized people access the custom library folder.

1.7.1.3 The library declaration

The library file extensions are different depending on the operating system:

- DLL for Windows

- SO for Linux or UNIX

The file types are:

Type	Description
XML catalogs declaration	There is only one file of this type, and it should be named `externalcatalogs.xml`. This file contains the list of all XML function definition files.
XML functions declaration	This file defines a list of functions and their associated library and is listed in the XML catalogs declaration file. **Note** The catalogs file can either contain or reference the function declaration libraries.
library file	This file contains the code in C++ for the user functions. The library file contains the user function implementation as defined in the XML function declaration.

1.7.1.4 Using the Web Intelligence sample files

Make sure to have the following applications installed:

- Visual Studio C++ VS2015 or higher
- Web Intelligence 4.1 or higher

The examples in this document use the sample files in the `Samples.zip` file located in `[Install directory]\userlibs\WebI\Samples\`.

1. Unzip `Samples.zip`.
2. To open the samples, launch `OpenSolution.bat`.

   The `OpenSolution.bat` sets the temporary `<WEBICALCPLUGINAPI>` variable environment that is used by the solution to find Web Intelligence specific headers files.

   Caution

   If the required version of Web Intelligence is not installed, you cannot use the `OpenSolution.bat`. If that is the case, manually set the `<WEBICALCPLUGINAPI>` variable environment to the path of the folder that contains the Web Intelligence header files and open Samples\WebICalcPlugIn.

Related Information

Examples [page 294]

1.7.2 Defining a custom calculation

To customize a function within Web Intelligence:

1. Define the XML function declaration
2. Define the XML catalog declaration.
3. Implement the library in C++ using the specific API for external function.
4. Compile the source file.
5. Copy the XML definition and the library into the dedicated `WebiCalcPlugin` folder (server side and any rich client).
6. Restart the Web Intelligence server.

> **Note**
>
> The chapter's examples use the sample files delivered with Web Intelligence.

The system automatically adds the function to the function list in the formula editor and formula bar contextual help.

If a formula is using a function for which no external library is available, the `#EXTERNAL` error message appears.

> **Note**
>
> Only functions that use single value parameters are supported. Table parameters for instance aren't supported.

1.7.2.1 XML function objects

The XML definition contains objects which define the custom function. XML custom functions extend the function list of the formula language so that a formula using this function can be parsed according its XML signature and turn into a tokenized form. You assign the external function a global unique ID (GUID) so that it cannot be reused or confused with other custom libraries.

The XML definition contains the following objects:

Tag	XML attribute	XML definition object
`<CATALOG>`		The XML root
`<LIBRARY>`	file	The name of the library file that contains the C++ implementation code The library file can contain several functions. The library extension should not be specified.

Tag	XML attribute	XML definition object
`<FUNCTION>`	guid	The unique function GUID **→ Tip** Define all GUIDs in advance and make sure that all GUIDs are unique from a global point of view. For Windows you can use the GUID tool provided with Visual Studio or download it from the Microsoft website. For Linux, the tool `usr/bin/uuidgen` can be found in the `libuuid1` (Debian) package.
	name	The function name that appears in the formula editor The function name must: • be a simple, unique name for the function • start with a letter • use lower and upper case letters, number characters, or the _ character • not already exist in the Web Intelligence library **ⓘ Note** The name will not be translated to another language.
`<ARGLIST>`		The list of parameters The number of parameters should be lower than or equal to five.
`<ARG>`	type	The parameter types The possible parameter types are as follows: • Numeric • Boolean • Date • String
	name	The name of each parameter as it should appear in the Formula Editor The name shows the prototype of the method to the user. Use only alphanumeric characters.

Tag	XML attribute	XML definition object
`<RETURN>`	type	The return values type Return values can be: • Numeric • Boolean • Date • String
`<CATEGORY>`	type	The category in which the function will appear in the Formula Editor Be consistent; place strings in the Character category and Numbers in the Numeric category. The available categories are: • Character • Date • Document • DP • Misc • Logical • Num
`<HINT>`	value	A hint to appear in the Formula Editor The hint explains the use of the function.

1.7.2.2 Defining the XML function declaration

The XML for the signature uses the following structure:

```
Function_list
```

The XML for the signature uses the following structure:

```
Function_list := [Function*]
Function := [name, GUID, data_type = Numeric|Boolean|Date|
String, category = character|Date|Document|DP|Misc|Logical|Num,
parameter_list, (online_help_signature?),
(online_help_description?),library_name)]
parameter_list := [parameter*]
parameter := [name, data_type =Numeric|Boolean|Date|String]
```

1. Set the XML root tag to `CATALOG`.
2. To the `CATALOG` add `LIBRARY` tags.
3. To the `LIBRARY` add the name of the library file without the DLL or SO file extension. This is the file attribute.
4. To the `LIBRARY` add `FUNCTION` tags.

A `FUNCTION` tag should have a unique GUID and an additional, unique attribute name which defines the name of the function.

The `FUNCTION` tag should contain:

- a `ARGLIST` tag with `ARG` tags. The `ARG` tags should have a first attribute type that defines the type of this parameter, and a second attribute that defines the name of this parameter.
  The `ARG` type can be Boolean, Numeric, Date, or String. The `ARG` name contains only alphanumeric characters.

  > **Note**
  >
  > You are limited to five parameters.

- a `RETURN` tag which defines a type attribute.
  The `RETURN` type can be Boolean, Numeric, Date, or String.
- a `CATEGORY` tag which defines a type attribute.
  The `CATEGORY` type can be Character, Date, Document, DP, Misc, Logical, or Num.
- a `HINT` tag which defines a value attribute.

5. Place the XML definition into the dedicated folder (server side and any rich client).

Example: SampleMath.xml

```
<CATALOG>
 <LIBRARY file="SampleMath">
  <FUNCTION guid="CC3E9742-67A7-4844-9DBF-2CCD4F6ECABE" name="MySquareFct">
    <ARGLIST>
     <ARG type="Numeric" name="input_number"/>
    </ARGLIST>
    <RETURN type="Numeric"/>
    <CATEGORY type="Num"/>
    <HINT value="My square function."/>
  </FUNCTION>
 </LIBRARY>
</CATALOG>
```

Related Information

Using the Web Intelligence sample files [page 287]

1.7.2.3 Defining the XML catalog declaration

You can create the XML catalog declaration or add it to an existing catalogs declaration.

`<CATALOG>` references an XML function declaration file or directly define the `<CATALOG>` as is shown in the section which defines an XML functions declaration format.

To create a catalog declaration:

1. Name the declaration `externalcatalogs.xml`.
2. Set the XML root tag to `CATALOGS`.
3. To the `CATALOGS` add `CATALOG` tags.

   This action defines the file name value of the XML functions declarations.
4. Place the XML library into the dedicated folder (server side and any rich client).

Example: externalcatalogs.xml

```
<CATALOGS>
    <CATALOG file="SampleMath.xml"/>
</CATALOGS>
```

Related Information

Using the Web Intelligence sample files [page 287]

1.7.2.4 Implementing the C++ file

1. In the file, add the `ibovariant.h` header.
2. For each method, start the declaration with the `BO_DECLARE_USER_FCT` macro.

   The macro includes:

   - the function name as it appears in the XML functions declaration file.
   - the return value object name
   - the parameter object name

   > **Note**
   >
   > The function returns a `BONOERROR` if everything is okay, otherwise the `#EXTERNAL` error message appears into the report.

Example: Square.cpp

```
// Headers file include of the WebI hearders
#include <ibovariant.h>
// To not repeat BOExtFunct::
using namespace BOExtFunct;
BO_DECLARE_USER_FCT (// Name of function as it was defined in the XML.
                        MySquareFct,
                     // Name of the return value object.
```

```
                    retVal,
                    // Name of the parameters object.
                    parameters
                    )
{
        try // Always used a try{}catch(…) to be sure no
            // exception was thrown outside this Web
            // Intelligence user function.
        {
           // Get the first parameter.
          const iBOValue&param0 = parameters[0];
           // Transform the parameter to the correct type.
          double valPar0(param0);
           // Assign value to the return value.
          retVal = valPar0 * valPar0;
        }
        catch(...)
        {
          return BOERROR; // Unkonwn exception so notify WebI
        }
        return BONOERROR; // It's OK
}
```

Related Information

Using the Web Intelligence sample files [page 287]

1.7.2.5 Compiling the source file in Microsoft Visual Studio 2015

1. To create a project, go to *File* > *New* > *Project*.
2. In *Project types*, select *Visual C++* > *General*.
3. In *Templates*, select *Empty Project*.
4. Specify the name of the project.
5. Specify the destination folder for the project.
6. Click *OK*.
7. Right-click the project and select *Properties*.
8. In *Configuration*, select *All configurations*.
9. In *Configuration Properties* > *General* set *Configuration Type* to *Dynamic Library (.dll)*.
10. Click *OK*.
11. Right-click the project and select *Add* > *New Item*.
12. In *Category*, select *Code*.
13. In *Template*, select *C++ File (.CPP)* .
14. Specify the name of the CPP file.
15. Click *Add*.
16. Right-click the project and select *Properties*.

17. In *Configuration*, select *All configurations*.
18. In *Configuration Properties* > *C/C++* > *Additional Include Directories*, add the folder which contains the Business Objects file headers.
19. Click *Apply*.
20. In *Configuration*, select *Debug*.
21. In *Configuration Properties* > *C/C++* > *Code generation*, set *Runtime Library* to *Multi-threaded DLL (/MD)*.

    > **Note**
    >
    > If you are running a machine where Microsoft Visual Studio is installed, you can use *Multi-threaded Debug DLL (/MDd)* instead of *Multi-threaded DLL (/MD)* to benefit from its debug environment.

22. Click *Apply*.
23. In *Configuration*, select *Release*.
24. In *Configuration Properties* > *C/C++* > *Code generation* set *Runtime Library* to *Multi-threaded DLL (/MD)* .

    > **Note**
    >
    > If you are running a machine where Microsoft Visual Studio is installed, you can use *Multi-threaded Debug DLL (/MDd)* instead of *Multi-threaded DLL (/MD)* to benefit from its debug environment.

25. Click *OK*.
26. Add the code to the CPP file.
27. Compile.

1.7.2.6 Copying files into WebiCalcPlugin

Copy the XML functions declaration, the XML catalogs declaration, and the DLL/SO file into the `WebiCalcPlugIn` folder.

The folder is available in:

```
[installation directory]\[BusinessObjects Version]\[OS]_[PLATFORM]\WebiCalcPlugIn
```

Where: [BusinessObjects Version] is the version of the product, for example BusinessObjects Enterprise XI 4.0, and [OS] is the operating system, for example win32 for Windows Operating System or linux for Linux Operating System, and [PLATEFORM] is the platform, for example x86 on an Intel 32-bit CPU.

1.7.3 Examples

The examples use the sample files in the `Samples.zip` file, which is located in `[Install directory]\userlibs\WebI\Samples\`.

Example: XML catalog declaration for the externalcatalogs.xml

```
<CATALOGS>
  <CATALOG file="SampleString.xml"/>
</CATALOGS>
```

Example: XML function declaration in SampleString.xml

```
<CATALOG>
 <LIBRARY file="SampleString">
  <FUNCTION guid="A91BD526-B8EB-4b09-90F2-FFCD350776A8" name="MyHelloWorld">
    <RETURN type="String"/>
    <CATEGORY type="Num"/>
    <HINT value="My simple hello world function."/>
  </FUNCTION>
 </LIBRARY>
</CATALOG>
```

Example: C++ file declaration in HelloWorld.cpp

```
// Headers file include of the Web Intelligence hearders
#include <ibovariant.h>
// To not repeat BOExtFunct::
using namespace BOExtFunct;
BO_DECLARE_USER_FCT(
                   // Name of function as it was defined in the XML.
                   MyHelloWorld,
                   // Name of the return value object.
                   retVal
                   // Don't use parameter.
                   /*parameters*/
)
{
     try  // Always used a try{}catch(…) to be sure no
          // exception was thrown outside this
          // Web Intelligence user function.
     {
          // Create an std::wstring with wide char Hello world.
          std::wstring helloWorldStr = L"Hello world!!!";
          // Initialyse the return value.
          retVal = helloWorldStr;
     }
     catch(...)
     {
          // Unkonwn exception so notify Web Intelligence
          return BOERROR;
     }
    return BONOERROR;   // It’s OK
}
```

Related Information

Using the Web Intelligence sample files [page 287]

1.7.4 #EXTERNAL error message

The #EXTERNAL error message is caused by the following problems:

- A formula refers to an external function that is not in the external library folder.
- A document contains an external method and the system cannot load it. The library file is not found, or there is an inconsistent declaration.
- An external method does not initialize the return value.
- An external method initialized the return type with bad type. For example, a double was set to a string.
- An external method returns an error code.

Ask the BusinessObjects administrator to deploy the correct library that implements this function.

1.7.5 Trace log message errors

If an error appears during XML parsing/validation, a message appears to the user and errors are created in the trace logs.

Log type	Error messages
XML logs	File cannot be read or is missing. Bad XML structure due to: • Parent/Children relation invalid. • Missing field (ID function, name function). • Invalid field value.
DLL logs	File is missing. DLL cannot be loaded. Function is not found in the DLL.

Log type	Error messages
Function logs	Function name is already in use.
	Function ID is already used.
	Function name is missing.
	Return type is invalid.
	ID is invalid.
	Number of parameters is invalid.
Parameters logs	Parameter name is missing.
	Parameter type is invalid.
Runtime logs	The user function does not initialize the return value.
	The user function initializes the return value with a bad type.
	The user function returns the BOERROR error code.

1.8 Troubleshooting formulas

1.8.1 Automatic rewrite formula mechanism

The succession of corrective maintenance releases for Web Intelligence can sometimes lead to calculation result differences between versions.

Since version 4.1 SP3, Web Intelligence provides an Automatic Formula Rewrite mechanism that automatically modifies a selection of formulas (see list below) in a document migrated from a previous version. These formulas follow a certain pattern. After modification, the formulas return the same result than before the calculation change. Therefore, it is recommended to save the document so that the modifications are stored in the document, thus completing the formula rewrite mechanism.

The Automatic Formula Rewrite mechanism is available by default for documents migrated to BI 4.1 SP3 and later, for the following formula patterns:

1. Where() operator with a dimension as parameter in a condition,
2. Running calculations with reset in sections,
3. Running calculations with reset in cross-tables.

This list of rules could be extended in future releases with more formula patterns.

Rule(1)

In previous versions, the data was calculated in a specific way if you had a Where() operator with a dimension as a parameter in a condition. Indeed, the dimension was added to the measure context. Rule(1) reproduces the former behavior.

This rules applies to every document migrated from XI 3.1 FP3.6, XI 3.1 FP4.1, XI 3.1 FP5.1 and. 4.0 SP5.

Rule(2)

In previous versions, running calculations in sections was not properly executed, as calculations would reset at each section instance. Rule(2) reproduces the former behavior.

This rules applies to every document migrated from XI R2 SP4.

Rule(3)

In previous versions, running calculations with reset cross-tables meant that calculations were executed in an "N" pattern (column after column) instead of a "Z" pattern (row after row).

Rule(3) introduced a FORCE_COL keyword that forces Web Intelligence to run calculations in a "N" pattern.

For example, with Rule(3), the RunningSum([Sales revenue];([State])) formula will be forced to execute column after column when modified as RunningSum([Sales revenue];FORCE_COL;([State])).

This rule applies to every document migrated from every version of XI 3.x, 4.0 Patch 2.20, 4.0 SP5, 4.0 SP6, 4.0 SP7, 4.1 and 4.1 SP1.

1.8.2 Formula error and information messages

You can format report data that returns error messages using conditional formatting.

In some cases a formula cannot return a value and returns an error or information message beginning with '#". The message appears in the cell in which the formula is placed.

1.8.2.1 #COMPUTATION

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

#COMPUTATION also occurs when a merged object containing a hierarchy is included in a report.

#COMPUTATION is also related to the misuse of context operators in a formula.

Related Information

RelativeValue [page 257]

1.8.2.2 #CONTEXT

#CONTEXT appears in a measure when the measure has a non-existent calculation context.

#CONTEXT is related to the #INCOMPATIBLE and #DATASYNC error messages, which appear in dimensions when a block contains a non-existent calculation context.

In the case of #INCOMPATIBLE the context is non-existent because the dimensions are incompatible; in the case of #DATASYNC the context is non-existent because the dimensions are from multiple unsynchronized data providers.

Example: Non-existent calculation context in a query

If a block based on the Island Resorts Marketing universe contains the Reservation Year and Revenue objects, the #CONTEXT error message appears because it is not possible to aggregate revenue by reservation year. (Reservations have not yet generated any revenue.)

1.8.2.3 #DATASYNC

#DATASYNC occurs when you place a dimension from a different data provider in a block containing dimensions from another data provider, and the two data providers are not synchronized through a merged dimension.

#DATASYNC appears in all dimensions in the block and #CONTEXT in the measures.

Example: Dimensions from different data providers in a block

If a report based on the Island Resorts Marketing universe contains data providers with the objects (Year, Revenue) and (Quarter), a block containing Year, Quarter and Revenue displays #DATASYNC in the Year and Quarter columns because the two data providers are not synchronized through a merged dimension.

1.8.2.4 #DIV/0

#DIV/0 occurs when a formula tries to divide a number by zero, which is mathematically impossible.

Zero can never appear as a divisor.

Example: Determining revenue per item

You have a report showing sales revenues, numbers of items sold and the revenue per item (which is calculated by dividing the sales revenue by the number of items sold).

You had a very bad quarter in which you didn't create any revenue; the Revenue per Item column returns #DIV/0 for this quarter, because the formula is attempting to divide by zero; that is, divide the revenue by zero number of items sold.

1.8.2.5 #ERROR

#ERROR is the default error message that covers all errors not covered by other error messages.

1.8.2.6 #EXTERNAL

#EXTERNAL occurs when a formula references an external function that is not available to use in Web Intelligence.

1.8.2.7 #INCOMPATIBLE

#INCOMPATIBLE occurs when a block contains incompatible objects.

Example: Incompatible objects in a query

If a block based on the Island Resorts Marketing universe contains the Year and Reservation Year dimensions, the columns containing these dimensions show #INCOMPATIBLE because these objects are incompatible.

1.8.2.8 #MIX

#MIX occurs when an aggregated measure has different units.

For example, a cell shows #MIX if it aggregates currency values denominated in different currencies.

1.8.2.9 #MULTIVALUE

#MULTIVALUE occurs when you place a formula that returns more than one value in a cell that outputs one value only.

Example: Multivalue in a cell

You have a report showing Country, Resort and Revenue and you add a cell to the report containing the formula [Revenue] ForEach ([Country]). This cell returns #MULTIVALUE because Country has two values in the report: 'US' and 'France'.

One cell cannot display the revenues for both the US and France. Placed outside the table, a cell containing revenue can only aggregate the revenues in the table in some way (for example by summing or averaging them).

If the report is broken into sections on Country, the formula is correct when placed in a section because there is only one value of Country per section. Outside a section, however, the formula still returns #MULTIVALUE

1.8.2.10 #N/A

When there is a value for a cell in report that is based on a value from a report that is not available on the underlying data base (for example, a BW error in a BEx Cell), the cell displays #N/A (not available), meaning that the cell is empty because the data cannot be retrieved.

1.8.2.11 #OVERFLOW

#OVERFLOW occurs when a calculation returns a value that is too large for the software to handle.

This value, in exponential form, is 1.7E308 (1.7 followed by 307 zeros).

1.8.2.12 #PARTIALRESULT

`#PARTIALRESULT` occurs when all rows associated with a report object were not retrieved.

If `#PARTIALRESULT` occurs often in your reports and you have the appropriate security rights, modify the `MaxRowsRetrieved` query property to allow the retrieval of more data. If you do not have the right to modify the query, contact the BI administrator.

If your report contains smart measures it is more likely to display `#PARTIALRESULT` because smart measures require the retrieval of larger amounts of data than classic measures.

1.8.2.13 #RANK

#RANK occurs when you try to rank data based on an object that depends on the order of values.

Objects that use the `Previous` function or any running aggregate function depend on the order of values.

Ranking causes these objects to recalculate their values, which then changes the ranking, resulting in a circular dependency. Such a dependency can occur either when you use the Rank dialog box to create a ranking, or when you use the `Rank` function.

Example: Ranking on running average or previous values

If you attempt to rank a block on a column that contains the `Previous` function or any running aggregate function, the entire block returns #RANK.

1.8.2.14 #RECURSIVE

#RECURSIVE occurs when it is not possible to perform a calculation due to a circular dependency.

Example: Using the NumberOfPages(), Page() and PageInSection() functions

If you place the `NumberOfPages`, `Page` and `PageInSection` functions in a cell whose Autofit Height or Autofit Width properties are set, the cell returns #RECURSIVE because the placing of these formulas in an Autofit cell creates a circular dependency. These functions need the exact size of the report to return a value, but the size of the cell, which affects the size of the report, is determined by the cell content.

1.8.2.15 #REFRESH

#REFRESH appears in report cells whose values are derived from objects that were stripped from a query and then re-added to the query.

Objects are stripped from a query when the *Enable query stripping* query property is selected and the objects do not contribute to any reports based on the query.

The cells are re-populated with values from the objects when the query is refreshed.

1.8.2.16 #SECURITY

#SECURITY occurs when you attempt to use a function for which you do not have security rights.

Example: Using the DataProviderSQL() function

If a user who does not have the right to view data provider SQL places the DataProviderSQL() function in a cell, the #SECURITY message appears in the cell.

1.8.2.17 #SYNTAX

#SYNTAX occurs when a formula references an object that no longer exists in the report.

Example: Referencing a non-existent object

You have a report that originally showed Year, Quarter and Sales revenue, with an additional column showing difference between the revenue and the average yearly revenue. This figure is given by the variable Difference from Yearly Average.

If the Difference from Yearly Average variable is deleted from the report, the column containing it returns #SYNTAX.

1.8.2.18 #TOREFRESH

#TOREFRESH appears in cells based on smart measures when the value returned by the smart measure is not available.

This situation occurs when the grouping set containing the value is not available in the data provider.

You remove the #TOREFRESH error by refreshing the data.

Some of the measures are "delegated" (for BW, this refers to a measure which is not aggregating with SUM); when you define a table or calculation on a measure, this measure is queried in specific context of aggregation (the measure is given for a set of dimensions). If this set of dimensions is a subset of the query dimension set, the measure has to be aggregated along the given dimension set (or grouping set that is referring to a group by clause in SQL).

For normal measures the system is carrying out the aggregation, for delegated measures this aggregation is delegated to the underlying database. For this the system needs to query again this database. Since this is not automatic, it displays #TOREFRESH and waits for the user to proceed with a refresh. Once the user refreshes, the system will run the additional query to get the requested aggregation and then replace #TOREFRESH by the appropriate value.

1.8.2.19 #UNAVAILABLE

#UNAVAILABLE appears when it is not possible to calculate the value of a smart measure.

This occurs when it is not possible to display the values in a filtered smart measure without applying a filter to the query. Because this carries a risk of impacting other reports based on the same query, no filter is applied.

1.9 Comparing values using functions

1.9.1 Comparing values using the Previous function

The `Previous` function returns a comparative previous value of an expression.

The value returned depends on the layout of the report.

For more powerful comparison capabilities, use the `RelativeValue` function. `RelativeValue` returns a previous or subsequent comparative value of an expression. The value returned does not depend on the layout of the report.

Related Information

Previous [page 252]
RelativeValue [page 257]
Comparing values using the RelativeValue function [page 305]

1.9.2 Comparing values using the RelativeValue function

The `RelativeValue` function returns comparative values of an expression. The function returns these values independently of the layout of a report.

When using `RelativeValue`, you specify the following:

- The expression whose comparative value you want to find (the expression must be a measure or a detail of a dimension available in the block)
- The list of slicing dimensions
- The offset.

The function uses the slicing dimensions, the offset, and the sub-axis dimensions (which are implied by the slicing dimensions) to return a comparative value. The sub-axis dimensions are all the other dimensions in the calculation context apart from the slicing dimensions.

Expressed in general terms, `RelativeValue` returns the value of the expression in the row which, in the list of values of the slicing dimensions, is `offset` rows removed from the current row, and where the values of the sub-axis dimensions are the same as in the current row.

> **Note**
>
> All slicing dimensions must always be in the calculation context of the block in which the function is placed. If a slicing dimension is subsequently removed, the function returns #COMPUTATION.

Example

In this example, the RelativeValue column contains the following formula:

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

- The expression is [Revenue];
- The slicing dimension is [Year];
- The offset is -1 (the function returns the immediately previous value in the list).

Year	Quarter	Sales Person	Revenue	RelativeValue
2007	Q1	Smith	1000	
2007	Q2	Jones	2000	
2007	Q3	Wilson	1500	
2007	Q4	Harris	3000	
2008	Q1	Smith	4000	1000
2008	Q2	Jones	3400	2000
2008	Q3	Wilson	2000	1500
2008	Q4	Harris	1700	3000

Expressed as a business question, the formula returns the revenue generated by the same sales person in the same quarter in the previous year.

Expressed as a calculation in words, the formula returns the value of [Revenue] (the expression) in the row where the value of [Year] (the slicing dimension) is the previous value from the list of values of the [Year] object, and where the values of [Quarter] and [Sales Person] (the sub-axis dimensions) are the same as in the current row.

Related Information

RelativeValue [page 257]

1.9.2.1 Slicing dimensions and the RelativeValue function

The `RelativeValue` function uses the list of values of the slicing dimensions to find the comparative row.

The function returns the comparative value of the expression specified in the function that is `offset` number of rows away in the list of slicing dimensions.

As a result, the sort order of the slicing dimensions is crucial in determining the function output.

Example: Multiple slicing dimensions

In the table below, the RelativeValue column has the following formula:

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

- The expression is [Revenue];
- The slicing dimensions are ([Year];[Quarter]);
- The offset is -1 (the function returns the immediately previous value in the list).

Year	Quarter	Sales Person	Revenue	RelativeValue
2007	Q1	Smith	1000	
2007	Q2	Smith	2000	
2007	Q3	Smith	1500	
2007	Q4	Smith	3000*	
2007	Q1	Jones	4000	
2007	Q2	Jones	3400	
2007	Q3	Jones	2000	
2007	Q4	Jones	1700	
2008	Q1	Smith	5000**	3000*
2008	Q2	Smith	3000***	5000**

Year	Quarter	Sales Person	Revenue	RelativeValue
2008	Q3	Smith	2700****	3000***
2008	Q4	Smith	6800	2700****

Expressed as a business question, the formula returns the revenue generated by the same sales person in the previous quarter.

Expressed as a calculation in words, the formula returns the value of [Revenue] in the row where the values of [Year] and [Quarter] represent the previous value in the ([Year];[Quarter]) list of values, and where the value of [Sales Person] is the same as in the current row.

The function uses the list of values of the slicing dimensions to find the comparative revenue:

Year	Quarter	
2007	Q1	
2007	Q2	
2007	Q3	
2007	Q4	*
2008	Q1	**
2008	Q2	***
2008	Q3	****
2008	Q4	

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

Related Information

RelativeValue [page 257]

1.9.2.2 Slicing dimensions and sections

A slicing dimension can be in the section leading cell of a report.

Example: Slicing dimension in a section cell

In the table below, the RelativeValue column has the following formula:

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

2007

Quarter	Sales Person	Revenue	RelativeValue
Q1	Smith	1000	
Q2	Smith	2000	
Q3	Smith	1500	
Q4	Smith	3000*	
Q1	Jones	4000	
Q2	Jones	3400	
Q3	Jones	2000	
Q4	Jones	1700	

2008

Quarter	Sales Person	Revenue	RelativeValue
Q1	Smith	5000**	3000*
Q2	Smith	3000***	5000**
Q3	Smith	2700 ****	3000***
Q4	Smith	6800	2700****

The function uses the list of values of the slicing dimensions to find the comparative revenue:

Year	Quarter	
2007	Q1	
2007	Q2	
2007	Q3	
2007	Q4	*
2008	Q1	**
2008	Q2	***
2008	Q3	****
2008	Q4	

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

Related Information

RelativeValue [page 257]

1.9.2.3 Order of slicing dimensions

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

Example: Order of slicing dimensions

In the table below, the RelativeValue column has the following formula:

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

Year	Quarter	Sales Person	Revenue	RelativeValue
2007	Q1	Smith	1000	
2007	Q2	Smith	2000	
2007	Q3	Smith	1500	
2007	Q4	Smith	3000*	
2007	Q1	Jones	4000	
2007	Q2	Jones	3400	
2007	Q3	Jones	2000	
2007	Q4	Jones	1700	
2008	Q1	Smith	5000**	3000*
2008	Q2	Smith	3000***	5000**
2008	Q3	Smith	2700****	3000***
2008	Q4	Smith	6800	2700****

Expressed as a business question, the formula returns the revenue generated by the same sales person in the previous quarter.

The sort order of the slicing dimensions is as follows:

Year	Quarter	
2007	Q1	
2007	Q2	
2007	Q3	
2007	Q4	*
2008	Q1	**
2008	Q2	***
2008	Q3	****
2008	Q4	

The function is changed to:

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

The sort order of the slicing dimensions becomes:

Quarter	Year	
Q1	2007	*
Q1	2008	**
Q2	2007	***
Q2	2008	****
Q3	2007	*****
Q3	2008	******
Q4	2007	*******
Q4	2008	********

The sort order has the following impact on the function result:

Year	Quarter	Sales Person	Revenue	RelativeValue
2007	Q1	Smith	1000*	
2007	Q2	Smith	2000***	
2007	Q3	Smith	1500*****	
2007	Q4	Smith	3000*******	
2007	Q1	Jones	4000	
2007	Q2	Jones	3400	
2007	Q3	Jones	2000	
2007	Q4	Jones	1700	
2008	Q1	Smith	5000**	1000*
2008	Q2	Smith	3000****	2000***
2008	Q3	Smith	2700******	1500*****
2008	Q4	Smith	6800********	3000*******

Expressed as a business question, the formula now returns the revenue generated by the same sales person in the same quarter of the previous year.

The change in the sort order of the slicing dimension changes the meaning of the formula. The * in the tables indicate the sort order.

Related Information

RelativeValue [page 257]

1.9.2.4 Slicing dimensions and sorts

Because the sort order of the list of values of the slicing dimensions determines the function output, a sort applied to any dimension in the slicing dimensions impacts the function output.

Example: A custom sort applied to a slicing dimension

In the table below, the RelativeValue column has the following formula:

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

A custom sort (Q1, Q2, Q4, Q3) is applied to [Quarter], giving the following result for the function:

Year	Quarter	Sales Person	Revenue	RelativeValue
2007	Q1	Smith	1000	
2007	Q2	Smith	2000	
2007	Q4	Smith	3000	
2007	Q3	Smith	1500*	
2007	Q1	Jones	4000	
2007	Q2	Jones	3400	
2007	Q4	Jones	1700	
2007	Q3	Jones	2000	
2008	Q1	Smith	5000**	1500*
2008	Q2	Smith	3000***	5000**
2008	Q4	Smith	6800****	3000***
2008	Q3	Smith	2700	6800****

The sorted list of slicing dimensions is as follows:

Year	Quarter	
2007	Q1	
2007	Q2	
2007	Q4	
2007	Q3	*
2008	Q1	**
2008	Q2	***
2008	Q4	****
2008	Q3	

The * in the tables show the sort order.

Related Information

RelativeValue [page 257]

1.9.2.5 Using RelativeValue in crosstabs

The `RelativeValue` function works in crosstabs in exactly the same way as in vertical tables.

The layout of the data in a crosstab has no impact on the function output.

Related Information

RelativeValue [page 257]

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering an SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

© 2024 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see https://www.sap.com/about/legal/trademark.html for additional trademark information and notices.

THE BEST RUN SAP