

The Macrobond formula language

The purpose of the Macrobond formula language is to make calculations on series of values. The series are typically time series. The result of a formula is a time series.

The formula language is available in Macrobond version 1.0.34 and later. This document refers to version 1.5 and if you use an older version of Macrobond, you might find that some functions are not available.

How calculation works

Series names

You can use names of series from the database in formulas.

A very simple example is the following:

sek

which evaluates to the series called *sek*.

Numerical constants

Examples of numerical constants are:

123

-123

123.456

123.456e-3

Formulas and series

An example of a simple formula is the following:

sek / nok

which calculates the cross rate between the Swedish and the Norwegian krona.

The time series *sek* and *nok* are treated as vectors of values. Each value can be a number or a *null* value.

Before any calculations are made, all series in the formulas are converted to the same frequency. They are also treated as series of the same length by adding *null* values to the beginning if the length differs.

The *division operator /* takes two series and produces a new series by dividing the corresponding values in the two series. For most calculations the, result is *null* when any of the values is *null*. With some fictional series values the result can look like this:

	<i>Observation</i>			
	<i>number</i>	<i>sek</i>	<i>nok</i>	<i>sek/nok</i>
1971-01-04	0		5.1643	
1971-01-05	1		5.1628	
1971-01-06	2	7.1367	5.1614	0.723219
1971-01-07	4	7.1386	5.1649	0.723517
1971-01-08	5	7.1382	5.1631	0.723306
1971-01-11	6	7.1390	5.1642	0.723379
1971-01-12	7	7.1411	5.1643	0.723180
1971-01-13	8	7.1390	5.1615	0.723000

In addition to operators, such as +, -, * and /, there are also functions. Functions can take zero or more parameters. Parameters are written within parentheses and separated with commas. An example of a formula using a function is the following:

Sum(sek)

This will calculate the sum of all values in the series. With the same fictional values, the result would look like this:

	<i>Observation</i>		
	<i>number</i>	<i>sek</i>	<i>Sum(sek)</i>
1971-01-06	0	7.1367	42.8326
1971-01-07	1	7.1386	42.8326
1971-01-08	2	7.1382	42.8326
1971-01-11	3	7.1390	42.8326
1971-01-12	4	7.1411	42.8326
1971-01-13	5	7.1390	42.8326

Another example is the following formula:

*sek*100*

The result is a new series where each value is multiplied by 100. There is no function multiplying a series with a constant. Instead, the formula interpreter will create a series where each value is 100 and then multiply these series:

	<i>Observation number</i>	<i>sek</i>	<i>constant</i>	<i>sek*100</i>
1971-01-06	0	7.1367	100	713.67
1971-01-07	1	7.1386	100	713.86
1971-01-08	2	7.1382	100	713.82
1971-01-11	3	7.1390	100	713.90
1971-01-12	4	7.1411	100	714.11
1971-01-13	5	7.1390	100	713.90

In most cases you do not need to know about the conversion of constants to series, but it can help you to understand what happens with a formula like this:

Sum(100)

If you have this formula together with the fictional series *sek*, the result would look like this:

	<i>Observation number</i>	<i>sek</i>	<i>Sum(100)</i>
1971-01-06	0	7.1367	600
1971-01-07	1	7.1386	600
1971-01-08	2	7.1382	600
1971-01-11	3	7,1390	600
1971-01-12	4	7.1411	600
1971-01-13	5	7.1390	600

You get the value 600 because the constant 100 is converted to a series with six values of 100 and then this series is summed.

Observation windows

Many functions are available in a form that takes a parameter that specifies the length of a *window*.

Sum(sek) This is the sum of all values in the series.

Sum(sek, 3) This is the sum of three observations including the current observation and the two previous ones.

The blue numbers in the table below show which numbers are included in the sum for one of the resulting values.

	<i>Observation number</i>	<i>sek</i>	<i>Sum(sek, 3)</i>
1971-01-06	0	7.1367	
1971-01-07	1	7.1386	
1971-01-08	2	7.1382	21.4135
1971-01-11	3	7,1390	21.4158
1971-01-12	4	7.1411	21.4183
1971-01-13	5	7.1390	21.4191

Another example of a function that takes a window length is *Mean(series, window)*. The result of this function is also known as the *moving average*.

Observation numbers

Each position in the series has an ordinal number that is called an *observation number*. You can get a series of the ordinal numbers with the function *Counter()*.

	<i>Observation number</i>	<i>sek</i>	<i>Counter()</i>
1971-01-06	0	7.1367	0
1971-01-07	1	7.1386	1
1971-01-08	2	7.1382	2
1971-01-11	3	7,1390	3
1971-01-12	4	7.1411	4
1971-01-13	5	7.1390	5

If it is a time series, each ordinal number is associated with a date. There are a number of functions related to dates and ordinal numbers. A useful function is *Date(year, month, day)* that returns the ordinal number for a specific date.

	<i>Observation number</i>	<i>sek</i>	<i>Date(1971,1,8)</i>
1971-01-06	0	7.1367	2
1971-01-07	1	7.1386	2
1971-01-08	2	7.1382	2
1971-01-11	3	7,1390	2
1971-01-12	4	7.1411	2
1971-01-13	5	7.1390	2

With the function $At(series, observation)$ you can get the value at a specific observation number.

	<i>Observation number</i>	<i>sek</i>	$At(sek, Date(1971,1,8))$
1971-01-06	0	7.1367	7.1382
1971-01-07	1	7.1386	7.1382
1971-01-08	2	7.1382	7.1382
1971-01-11	3	7.1390	7.1382
1971-01-12	4	7.1411	7.1382
1971-01-13	5	7.1390	7.1382

Please note that you should never make any assumption about what the ordinal numbers are. Thus, you should **not** use an expression like $At(sek, 0)$ to get the first value of a series. Instead you should use $FirstValid(sek)$.

Logical values and functions

The logical values True and False are represented by 1 and 0.

In the following example the operator $>$ (“greater than”) is used to compare a series with a lagged series. The result will be True (1) when the value has increased since the last observation.

	<i>Observation number</i>	<i>sek</i>	$Lag(sek, 1)$	$sek > Lag(sek, 1)$
1971-01-06	0	7.1367		
1971-01-07	1	7.1386	7.1367	1
1971-01-08	2	7.1382	7.1386	0
1971-01-11	3	7.1390	7.1382	1
1971-01-12	4	7.1411	7.1390	1
1971-01-13	5	7.1390	7.1411	0
1971-01-13	6		7.1390	

The function If is commonly used with logical expressions. The first parameter is interpreted as a logical expression that determines if the result should be the value of the second or the third parameter.

The following example replaces any null values (missing values) with the highest value of the last ten observations:

$If(IsNull(sek), Hi(sek, 10), sek)$

Operators

All operators work on constants and series. Here are the operators sorted in order of precedence.

Operator	Description	Precedence
-	Unary minus	7
*	Multiplication	6
/	Division	6
+	Plus	5
-	Minus	5
!	Logical “not”	4
>	Greater than	3
>=	Greater than or equal	3
<	Less than	3
<=	Less than or equal	3
=	Equal	3
<>	Not equal	3
&	Logical “and”	2
	Logical “or”	1

Parentheses can be used to further control the order of evaluation.

Comments

You can write comments in formulas by typing two slashes “//”. Everything after the slashes to the end of the line, will be regarded as a comment and not part of the formula expression.

Built in functions

In the function descriptions below, the following convention has been used when naming parameters:

value The parameter can be either a number or a series. The type of the result is typically the same as this parameter. For instance, $\text{Log}(20)$ returns a number, but $\text{Log}(\text{sek})$ returns a series.

series The parameter can only be a series. (See discussion earlier in this document for how the automatic conversion of numbers to series works)

other Other parameter names, such as number, window, observation, length, must be numbers unless other information is provided. For instance, you can write Lag(sek, 10), but not Lag(sek, nok).

Mathematical functions

Abs(value)

Returns the absolute value.

Acos(value)

Returns the angle whose cosine is the specified value.

The angle is expressed in radians.

AggregateProduct(series)

Returns the aggregated product of all previous numbers. Null numbers are treated as one.

AggregateSum(series)

Returns the aggregated sum of all previous numbers. Null numbers are treated as zero.

Asin(value)

Returns the angle whose sine is the specified value.

The angle is expressed in radians.

Atan(value)

Returns the angle whose tangent is the specified value.

The angle is expressed in radians.

Atan2(x, y)

Returns the angle whose tangent is the quotient of two specified values.

The angle is expressed in radians.

Beta(series1, series2)

Returns the beta number after regression correlation of series1 and series2 with series1 as the dependent.

Beta(series1, series2, window)

Returns the beta after regression correlation of series1 and series2 with series1 within a window of the specified length

Ceiling(value)

Returns the smallest integer greater than or equal to the specified value.

Cos(value)

Returns the cosine of the specified angle.

The angle should be expressed in radians and must be in the range -9223372036854775295 to 9223372036854775295.

Cot(value)

Returns the cotangent of the specified angle.

The angle should be expressed in radians.

CountValid(series)

Returns the number of values that are not null.

CountValid(series, window)

Returns the number of values that are not null in a window of the specified length.

CoVariance(series1, series2)

Returns the covariance of series1 and series2 within a window of the specified length.

CoVariance(series1, series2, window)

Returns the covariance of series1 and series2 within a window of the specified length.

EMean(series, factor)

Returns the exponential moving average with the specified factor.

The factor must be a number between 0 and 1.

Exp(value)

Returns e raised to the power of the specified value.

First(series)

Returns the first number of a series.

FirstValid(series)

Returns the first number of a series that is not null.

Floor(value)

Returns the largest integer less than or equal to the specified value.

High(series)

Returns the highest number.

High(series, window)

Returns the highest number in a window of the specified length.

HPFilter(series, lambda)

Returns the Hodrick-Prescott smoothing with the specified lambda factor, which must be a number greater than zero.

A typical value for lambda when used for quarterly data is 1600.

Intercept(series1, series2)

Returns the intercept after regression correlation of series1 and series2 with series1 as the dependent.

Intercept(series1, series2, window)

Returns the intercept after regression correlation of series1 and series2 with series1 within a window of the specified length.

Last(series)

Returns the last value of a series.

Linear(series)

Returns a line fitted using the least square method.

Linear(series, observationStart, observationEnd)

Returns a line fitted using the least square method by using data from observationStart to, but not including, observationEnd.

LastValid(series)

Returns the last value of a series that is not null.

LinIntercept(series)

Returns the intercept of a line fitted with the least square method.

LinIntercept(series, window)

Returns the intercept of a line fitted with the least square method within a window of the specified length.

LinSlope(series)

Returns the slope of a line fitted with the least square method.

LinSlope(series, window)

Returns the slope of a line fitted with the least square method within a window of the specified length.

Ln(value)

Returns the natural logarithm.

Log(value)

Returns the logarithm with base 10.

Low(series)

Returns the lowest number of the series.

Low(series, window)

Returns the lowest number in a window of the specified length.

Max(value1, value2)

Returns the largest value of a pair of values.

Mean(series)

Returns the mean value as a number.

Mean(series, window)

Returns the moving average of the specified length.

Median(series)

Returns the median as a number.

Median(series, window)

Returns the median value in a window of the specified length.

Min(value1, value2)

Returns the smallest value of a pair of values.

Mod(value1, value2)

Returns the remainder of an integer division.

The calculation uses a floored division and is equal to: $\text{value1} - \text{value2} * \text{Floor}(\text{value1}/\text{value2})$

Momentum(series, length)

Returns the difference between the series and a lagged series.

Null()

Returns the null number.

Null0(value)

Returns 0 if the value is null and otherwise the value.

Null1(value)

Returns 0 if the value is null and otherwise the value.

PI()

The number π .

E()

The number e .

Pow(value, power)

Returns a value raised to a power.

Product(series)

Returns the product of all numbers in the series.

Product(series, window)

Returns the product of all values in a window of the specified length.

Sign(value)

Returns -1 for negative values, 0 for 0 and 1 for positive values.

Sin(value)

Returns the sine of the specified angle.

The angle should be expressed in radians and must be in the range -9223372036854775295 to 9223372036854775295.

Sqrt(value)

Returns the square root.

Sum(series)

Returns the sum of all numbers in the series.

Sum(series, window)

Returns the sum of all numbers in a window of the specified length.

Tan(value)

Returns the tangent of the specified angle.

The angle should be expressed in radians.

Statistical functions

ChiDist(value, degFree)

Returns the probability for a value based on the chi distribution with the specified degrees of freedom.

The parameters can be either a number or series.

ChiDistInv(probability, degFree)

Returns the value for a probability based on the chi distribution with the specified degrees of freedom.

The parameters can be either a number or series.

FDist(value, degFreeNum, degFreeDenom)

Returns the probability for a value based on the F distribution with the specified degrees of freedom for the numerator and denominator.

The parameters can be either a number or series.

FDistInv(probability, degFreeNum, degFreeDenom)

Returns the value for a probability based on the F distribution with the specified degrees of freedom for the numerator and denominator.

The parameters can be either a number or series.

LogNormDist(value, mean, stdDev)

Returns the probability for a value based on the log-normal distribution with the specified mean and standard deviation.

The parameters can be either a number or series.

LogNormDistInv(probability, mean, stdDev)

Returns the value for a probability based on the log-normal distribution with the specified mean and standard deviation.

The parameters can be either a number or series.

NormDist(value)

Returns the probability for a value based on the standard normal distribution.

The parameters can be either a number or series.

NormDist(value, mean, stdDev)

Returns the probability for a value based on the normal distribution with the specified mean and standard deviation.

The parameters can be either a number or series.

NormDistInv(probability)

Returns the value for a probability based on the standard normal distribution.

The parameters can be either a number or series.

NormDistInv(probability, mean, stdDev)

Returns the value for a probability based on the normal distribution with the specified mean and standard deviation.

The parameters can be either a number or series.

Percentile(series, p)

Returns the number of the p:th percentile from series, where p is in the range 0-100.

Percentile(series, p, window)

Returns the value of the p:th percentile within a window of the specified length, where p is in the range 0-100.

Standardize(series)

Returns a normalised series where the mean is 0 and the standard deviation is 1.

StdDev(series)

Returns the standard deviation of the numbers in a series relative its mean value.

StdDev(series, window)

Returns the standard deviation compared to the mean value within a window of the specified length.

TDist(value, degFree)

Returns the probability for a value based on the Student-T distribution with the specified degrees of freedom.

The parameters can be either a number or series.

TDistInv(probability, degFree)

Returns the value for a probability based on the Student-T distribution with the specified degrees of freedom.

The parameters can be either a number or series.

TwoPieceNormDist(value, mode, stdDev1, stdDev2)

Returns the probability for a value based on the 2-piece normal distribution with the specified mode, mean and standard deviations.

The parameters can be either a number or series.

TwoPieceNormDistInv(probability, mode, stdDev1, stdDev2)

Returns the value for a probability based on the 2-piece normal distribution with the specified mode, mean and standard deviation.

The parameters can be either a number or series.

Variance(series)

Returns the variance of the series.

Variance(series, window)

Returns the variance for a series within a window of the specified length.

Date and observation number functions

AddMonths(months)

Returns a series of all observation numbers offset by a number of months.

AddMonths(observation, months)

Returns the observation number for the specified observation number plus a number of months.

AddYears(years)

Returns a series of all observation numbers offset by a number of years.

AddYears(observation, years)

Returns the observation number for the specified observation number plus a number of years.

At(series, observationValue)

Returns the value in the series at the specified observation number.

Counter()

Returns a series of all observation numbers.

Counter(series)

Returns a series of all observation number for the specified series.

Date(year, month, day)

Returns the observation number of the specified date, where month is 1-12 and day is 1-31.

If there is no observation at the specified date, the observation number of the closest previous observation is returned.

You should generally not use this function to find the start of a year or month. Use the StartOfYear(year) and StartOfMonth(year, month) for this purpose.

Day()

Returns a series of the day (1-31) for each observation.

Day(observation)

Returns the day (1-31) of the specified observation number.

DayOfWeek()

Returns a series of day of the week for each observation.

0 = Sunday, 1 = Monday, 2 = Tuesday, 3 = Wednesday, 4 = Thursday, 5 = Friday, 6 = Saturday

DayOfWeek(observation)

Returns the day of the week of the specified observation number.

0 = Sunday, 1 = Monday, 2 = Tuesday, 3 = Wednesday, 4 = Thursday, 5 = Friday, 6 = Saturday

Days(observation1, observation2)

Returns the number of calendar days passed from the first observation number to the second one.

Days30E(observation1, observation2)

Returns the number of days passed from the first observation number to the second using the 30E convention.

End()

Returns the last observation number for calculations.

End(series)

Returns the last observation number of the series.

EndOfMonth()

Returns a series with the last observation number during each month.

EndOfMonthAhead(monthsAhead)

Returns a series with the last observation number during each month.

EndOfQuarter()

Returns a series with the last observation number during each quarter.

EndOfQuarterAhead(monthsAhead)

Returns a series with the last observation number during each quarter.

EndOfYear()

Returns a series with the last observation number during each year.

EndOfYearAhead(monthsAhead)

Returns a series with the last observation number during each year.

EndValid(series)

Returns the observation number of the last valid value of the series.

FlagForecast(value)

Returns a value as a forecast value.

FlagForecast(value, condition)

Returns a value as a forecast value if the condition is True and as a non-forecast value if it is False.

IsForecast(value)

Returns True if a value is a forecast and otherwise False.

Month()

Returns a series of the month (1-12) for each observation.

Month(observation)

Returns the month (1-12) of the specified observation number.

MonthOffset()

Returns a series with the offset of each observation within the month.

MonthOffset(observation)

Returns the offset within the month of the specified observation number.

NextValidObservationNumber(series)

Returns a series that contains the observation number of the current observation if it is valid and otherwise the number of the next observation that is valid.

NextValidObservationNumber(series, observation)

Returns the observation number of the specified observation if it is valid and otherwise the number of the next observation that is valid.

Now()

Returns the observation number of today's date.

ObservationCountPerYear()

Returns the number of observations per year based on the frequency.

PreviousValidObservationNumber(series)

Returns a series that contains the observation number of the current observation if it is valid (not null) and otherwise the number of the previous observation that is valid.

PreviousValidObservationNumber(series, observation)

Returns the observation number of the specified observation if it is valid and otherwise the number of the previous observation that is valid.

Quarter()

Returns a series of the quarter (1-4) for each observation.

Quarter(observation)

Returns the quarter (1-4) of the specified observation number.

QuarterOffset()

Returns a series with the offset of each observation within the quarter.

QuarterOffset(observation)

Returns the offset within the quarter of the specified observation number.

Start()

Returns the first observation number for calculations.

Start(series)

Returns the first observation number of the series.

StartValid(series)

Returns the observation number of the first valid value of the series.

StartOfYear()

Returns a series with the first observation number during each year.

StartOfYear(year)

Returns the observation number of the first observation of the specified year.

StartOfQuarter()

Returns a series with the first observation number during each quarter.

StartOfMonth()

Returns a series with the first observation number during each month.

StartOfMonth(year, month)

Returns the observation number of the first observation of the specified month, where the month is in the range 1-12.

Year()

Returns a series of the year for each observation.

Year(observation)

Returns the year of the specified observation number.

YearOffset()

Returns a series with the offset of each observation within the year.

YearOffset(observation)

Returns the offset within the year of the specified observation number.

Years(observation1, observation2)

Returns the number of years between the first observation number to the second. The calculation is made using the Actual/Actual method defined by ISDA.

Series operations

Cut(series, observationStart, observationEnd)

Returns a series excluding observations before the specified start observation number and all observations starting with the specified end observation number.

CutEnd(series, observation)

Returns a series of all observations before the specified observation number.

CutStart(series, observation)

Returns a series without any observations before the specified observation number.

Extend(series, observation, number)

Returns a series with the specified number at the specified observation number if it is past the end.

ExtendLastAsForecast(series)

Extends the series with the last valid value until the end of the calendar.

ExtendLinear(series, observation, number)

Returns a series extended with the specified number using a linear interpolation from the end until the specified observation number.

Join(series1, series2)

Returns a series that joins two series at the end of series1.

Join(series1, series2, observation)

Returns a series that joins two series at the specified observation number.

JoinScaled(series1, series2)

Returns a series that joins two series at the end of the series1 and scales the first series.

JoinScaled(series1, series2, observation)

Returns a series that joins two series at the specified observation number and scales the first series.

Lag(series, length)

Returns the series lagged by the specified length.

The length is rounded to an integer.

Trim(series)

Returns a series where any null values at the start or end have been excluded.

Logical functions

Count(series)

Returns the number of values that are True.

Count(series, window)

Returns the number of values that are True in a window of the specified length.

If(condition, value1, value2)

Returns the first value if condition is True and the second value if it is False.

IsForecast(value)

Returns True if a value is a forecast and otherwise False.

IsNull(value)

Returns True if a value is null and otherwise False.

Financial

BollingerLower(series, nStdDev, window)

Returns the lower Bollinger band using a window of the specified length and number of standard deviations.

BollingerUpper(series, nStdDev, window)

Returns the upper Bollinger band using a window of the specified length and number of standard deviations.

COP(series, length)

Returns the percentage change over the number of specified observations.

FMACD(series, short, long)

Returns the fast moving average convergence divergence.

Rsi(series, window)

Returns the relative strength index of the series using a window of the specified length.

SMACD(series, short, long, signal)

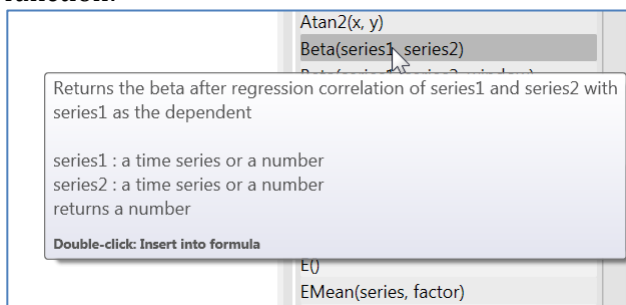
Returns the exponential average of the fast moving average convergence divergence.

The formula editor

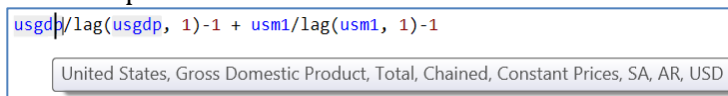
The Formula editor helps you write formulas. In the Series List, you can bring up the Formula editor by clicking on the fx button to the right of an expression, or by pressing Ctrl+E on the keyboard.

Some of the useful features of the editor are:

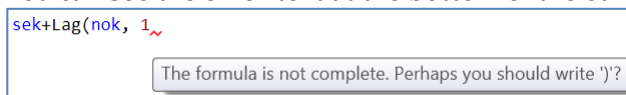
- You can double click on a function in the list to the right in order to insert a function.
- You can filter the list of functions by typing a text in the box above the list of functions. Only functions that contain the text in the function name or description will be included in the list.
- The tooltip of a function in the list or in an expression will show details about the function:



- The tooltip of a series will show the title of the series:



- Numerical constants are dark red. Series, parameters and variables are blue. Comments are green.
- When the caret is just before a left parenthesis or directly after a right parenthesis, a blue shade on the background will show the area between the matching parentheses:
$$(usgdp/lag(usgdp, 1)-1) - (usm1/lag(usm1, 1)-1)$$
- When the caret is within or next to an identifier, all instances of that identifier in the expression are shaded:
$$segdp*nok/sek + sem1*nok/sek$$
- The editor will check the syntax as you write and show a red squiggle if there is an error. You can see the error text at the bottom of the editor or as a tooltip:



User defined functions

User defined functions are used as a way to structure formula expressions and to reuse commonly used expressions. The names of user defined functions and variables must start with a period "." followed by a letter. The rest of the name can contain letters, numbers and underscore "_".

The names of parameters follow the same rule, except that they should not begin with a period.

In-line functions and variables

Sometimes you want to use the same sub expression several times in a formula. In this example, the sub expression *nok/sek* is used twice:

```
segdp*nok/sek + sem1*nok/sek
```

You can make the expression shorter and easier to read by using an inline variable. In-line variables and functions are declared by using the let-in-end construct. The expression above can be rewritten like this:

```
let  
  .fx = nok/sek  
in  
  segdp*.fx + sem1*.fx  
end
```

In the new expression, a variable called *.fx* is declared and assigned the result of *nok/sek* and then used in the expression.

You could write this as one line if you like, but in general it is easier to read if you use line breaks.

In addition to variables, you can declare in-line functions. Let us take this example:

```
usgdp/lag(usgdp, 1)-1 + usm1/lag(usm1, 1)-1
```

Since the same calculation is done for two different series, this can be written like:

```
let  
  .ret(series) = series/lag(series, 1)-1  
in  
  .ret(usgdp) + .ret(usm1)  
end
```

In this example, a local function called `.ret` is declared: `.ret(series) = series/lag(series, 1)-1`
It is declared to take one parameter

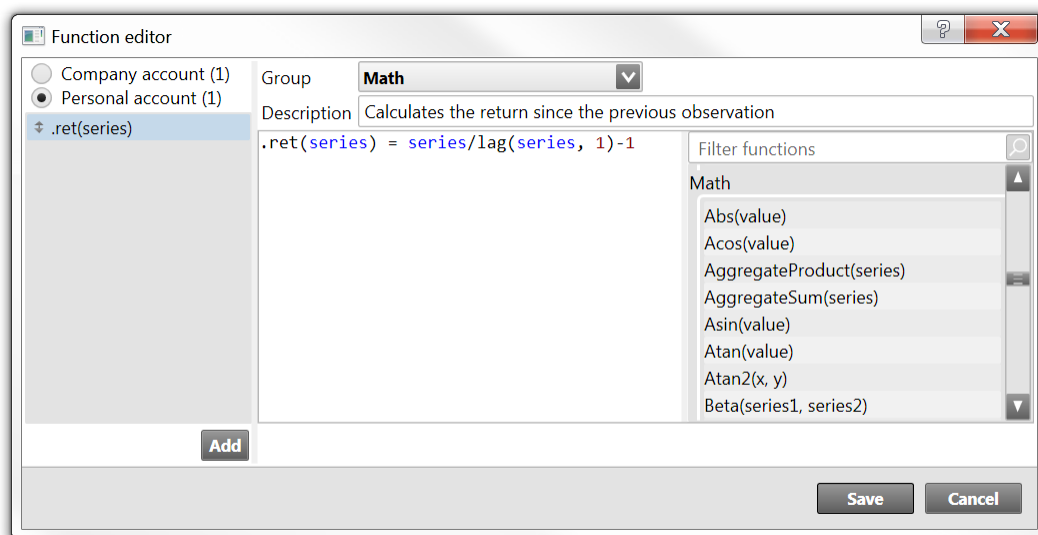
You can define several variables and functions in one expression:

```
let
  .ret(series) = series/lag(series, 1)-1
  .prop = 0.6
in
  .ret(usgdp)*.prop + .ret(usm1)*(1-.prop)
end
```

Reusable functions

The Function editor

If you have a function definition that you want to use in several places, you can add them to the list of functions in the *Function editor*. You bring up the editor by selecting “User defined functions...” on the Edit menu when the Analytics activity is active.



You can store your functions in one of the different stores. Functions stored in the Company account will be available for all users in the company. Functions in the Personal account store are only available to you. You can also store functions that should only be available in the current document. For some users there is a store called Library. This store is available to all users in the company, but only some users, with special access rights, can write to it.

You add a new function to a store by pressing the Add button. The new function can then be edited in the editor. You can also select what group it belongs to and you can type a description of the function. This description will be available as a tooltip in the Formula editor.

You can delete a function by selecting it in the list to the left and then press the Del key or select Delete from the context menu. Copy, Cut and Paste can be used in the list to move or duplicate functions.

Function order

Two functions with the same name, but with a different number of parameters, are regarded as two separate functions.

A function with the same name and same number of parameters can be defined several times. When you use a user defined function in a formula in the Series List, the formula evaluator will try to find the definition of the function by looking in the following order and pick the first instance:

- 1) In-line function definitions
- 2) In the Document store
- 3) In the Private account store
- 4) In the Company account store
- 5) In the Library store (if available)

The stores in the Function editor are presented with the store it will look in first at the bottom and the last store at the top of the list.

If the function is defined several times within a store, the last definition will be used. The last definition is the one closest to the bottom of the list in the Function editor.

When user defined functions are used in other user defined functions, the evaluator will look for definitions earlier in the same store and then in the earlier stores, if there are any.

Sharing documents that uses user defined functions

In order to make it possible to share documents with other users that do not have access to the same function stores, a hidden copy of all user defined functions used in the document, is stored in the document.

When a user opens a document that uses user defined functions, and the user does not have access to the original function stores, the document is still functional. You can change analysis, charts and tables in the document, but you cannot edit the expressions in the Series List.

As an example, if one user has a private function called *.ret(series)* and uses this in a document in a formula like *.ret(sek)* and then sends this document to another user, then the other user will see this information when looking at the Series List:

This document is using functions that could not be found. Click to unlock by copying the missing functions to the document.	
Expression	Description
.ret(sek)	Sweden, FX Spot Rates, Macrobond, .ret(SEK per USD)

It is important to note that even if the other user has also defined a function with the same name in his private store, it will not be regarded as the same function since it is not stored in the same store. You will still get the yellow bar and the original function will be used.

You can unlock the document by clicking on the yellow bar. The relevant functions will then be copied to the Document function store and you are free to edit the expressions.