# Formula Language Specification for the EcoSpold2 Format

## Objectives

The EcoSpold2 format introduces the new field „mathematicalRelation" for exchanges, properties and parameters. The allowed content for this field is described in this document. Its goal is to specify a formula language with a manageable, well-defined set of functions that cover the needs of data suppliers and can be implemented by LCA software vendors with little effort.

## OpenFormula as Specification Basis

The OpenDocument format (ODF) is a widely accepted standard. The OpenDocument v1.0 specification was approved as an OASIS Standard on 1 May 2005. It is available in PDF and in OpenOffice.org XML formats.

The OpenDocument v1.1 specification was approved as OASIS Standard on 2 February 2007. It is available in OpenDocument, PDF and XHTML (zipped) formats.

Currently the OpenDocument v1.2 Specification is under development. OpenDocument v1.2 consists of three parts. Part one is in Committee Draft state. Part two and three are in draft state.

Part 1 defines an XML schema for office applications and its semantics. The schema is suitable for office documents, including text documents, spreadsheets, charts and graphical documents like drawings or presentations, but it is not restricted to this kind of documents.

Part 2 (used as a base for this document) defines a formula language to be used in OpenDocument documents.

Part 3 defines a package format to be used for OpenDocument documents.

Participants in the subcommittee working on the OpenFormula specification are: IBM, KDE, Microsoft Corporation, Novell, Sun Microsystems

## Starting Document

This specification is based on this document:

Open Document Format for Office Applications (OpenDocument)v1.2
Part 2: Recalculated Formula (OpenFormula) Format – Annotated Version
Pre-Draft12, 8 May 2009

*OpenDocument-formula-20090508.odt*

From the following web page:
http://www.oasis-open.org/committees/documents.php?wg_abbrev=office-formula

## Important ODF Chapters

Please check the ODF specification document for these important introductions to the formula language, including a description of the supported standard operators:

- "2.1.1 Small Group" defines a minimal subset of the standard, but the EcoSpold02 subset is even smaller.
- "3.1 Expression Calculation" defines the calculation logic.
- "3.6 Basic Limits" defines limitation like minimal formula length.
- "6.3 Standard Operators" defines a clear syntax and describes how its display in user interfaces can differ from the storage format of formulas.

## Irrelevant ODF Chapters

A large part of Open Document Format is not relevant for the ecoSpold02 format. This includes all data types besides numbers and the functions defined for their manipulation, financial and statistical functions and definitions related to spreadsheet software. The below chapters are considered to be irrelevant to the ecoSpold02 format.

### Types

- 4.1 Text (String)
- 4.3 Complex Numbers
- 4.5 Error
- 4.7 Reference List
- 4.8 Array

### Standard Operators and Functions

- 6.5 Bit operation functions
- 6.6 Byte-position text functions
- 6.7 Complex Number Functions
- 6.8 Database Functions
- 6.9 Date and Time Functions
- 6.10 External Access Functions
- 6.11 Financial Functions
- 6.12 Information Functions (mostly)
- 6.13 Lookup Functions
- 6.18 Number Representation Conversion Functions
- 6.19 Text Functions

# Open Formula Specifications Supported by the ecoEditor for ecoinvent Version 3

This section lists the ODF functions that are currently supported by the ecoEditor. Each function is listed with its ODF definition and (where necessary) the difference between its ODF definition and its implementation in the ecoinvent context.

Functions not listed here are not supported by the ecoEditor and the ecoinvent calculation. Using other ODF functions in an ecoSpold02 document will cause errors when calculation its amounts and will lead to its rejection when it is submitted to the ecoinvent database.

The operators described in chapter 6.3 of the ODF are supported by the ecoEditor and the ecoinvent calculation.

## ABS

Summary: Return the absolute (nonnegative) value.

Syntax: ABS( Number N )

Returns: Number

Constraints: None

Semantics: If N < 0, returns -N, otherwise returns N

## AND

Summary: Compute logical AND of all parameters.

Syntax: AND( { Logical|NumberSequenceList L }+ )

Returns: Logical

Constraints: Must have 1 or more parameters

Semantics: Computes the logical AND of the parameters. If all parameters are True, returns True; if any are False, returns False. When given one parameter, this has the effect of converting that one parameter into a logical value. When given zero parameters, applications may return a Logical value or an error.

Changes compared to ODF definition: Calling AND() with 0 parameters causes an error.

## AVERAGE

Summary: Average the set of numbers

Syntax: AVERAGE( { NumberSequence N }+ )

Returns: Number

Constraints: At least one number included. Returns an error if no numbers provided.

Semantics: Computes SUM(List) / COUNT(List).

## CEILING

Summary: Round a number N up to the nearest multiple of the second parameter, significance.

Syntax: CEILING( Number N [ ; Number significance] )

Returns: Number

Constraints: Both N and significance must be numeric and have the same sign if not 0.

Semantics: Rounds a number up to a multiple of the second number. If significance is omitted it is assumed to be -1 if N is negative and +1 if N is non-negative, making the function act like the normal mathematical ceiling function. Rounding is toward positive infinity; the number is rounded to the smallest multiple of significance that is equal-to or greater than N. If any of the two parameters N or significance is zero, the result is zero.

Changes compared to ODF definition: the optional third parameter mode is not supported.

## COUNT

Summary: Count the number of Numbers provided

Syntax: COUNT( { NumberSequenceList N }+ )

Returns: Number

Constraints: One or more parameters.

Semantics: Counts the numbers in the list of NumberSequences provided. Only numbers in references are counted; all other types are ignored. Errors are not propogated. It is implementation-defined what happens if 0 parameters are passed, but it should be an Error or 0.

## EVEN

Summary: Rounds a number up to the nearest even integer. Rounding is away from zero.

Syntax: EVEN( Number N )

Returns: Number

Constraints: None

Semantics: Returns the even integer whose sign is the same as N's and whose absolute value is greater than or equal to the absolute value of N. That is, if rounding is required, it is rounded away from zero.

## EXP

Summary: Returns e raised by the given number.

Syntax: EXP( Number X )

Returns: Number

Constraints: None

Semantics: Computes $e^X = 1 + \frac{X}{1!} + \frac{X^2}{2!} + \frac{X^3}{3!} + \frac{X^n}{n!} + \ldots$

## FALSE

Summary: Returns constant FALSE

Syntax: FALSE()

Returns: Logical

Constraints: Must have 0 parameters

Semantics: Returns logical constant FALSE. This may be a Number or a distinct type. Syntactically this is a function call, but semantically this is a constant, and applications typically optimize this as a constant.

## FLOOR

Summary: Round a number N down to the nearest multiple of the second parameter, significance.

Syntax: FLOOR( Number N [ ; Number significance] )

Returns: Number

Constraints: Both N and significance must be numeric and have the same sign.

Semantics: Rounds a number down to a multiple of the second number. If significance is omitted it is assumed to be -1 if N is negative and +1 if N is positive, making the function act like the normal mathematical floor function. Rounds toward negative infinity, and the result is the largest multiple of significance that is less than or equal to N. If any of the two parameters N or significance is zero, the result is zero.

Changes compared to ODF definition: the optional third parameter mode is not supported.

## IF

Summary: Return one of two values, depending on a condition

Syntax: IF( Logical Condition ; IfTrue ; IfFalse )

Returns: Any

Constraints: None.

Semantics: Computes Condition. If it is TRUE, it returns IfTrue, else it returns IfFalse.

Changes compared to ODF definition: All 3 parameters are required.

## ISERROR

Summary: Return TRUE if the parameter has type Error, else return FALSE

Syntax: ISERROR( Scalar X )

Returns: Logical

Constraints: None

Semantics: If X is of type Error, returns TRUE, else returns FALSE. Note that this function returns True if given NA(); if this is not desired, use ISERR. Note that this function does not propagate error values.

## ISEVEN

Summary: Return TRUE if the value is even, else return FALSE

Syntax: ISEVEN( Number X )

Returns: Logical

Constraints: None

Portable Constraints: X must not be Logical

Semantics: First, compute X1=TRUNC(X). Then, if X is even (a division by 2 has a remainder of 0), return True, else return False. The result is implementation-defined if given a logical value; an application may return either an Error or the result of converting the logical value to a number (per Conversion to Number).

## ISLOGICAL

Summary: Return TRUE if the parameter has type Logical, else return FALSE

Syntax: ISLOGICAL( Scalar X )

Returns: Logical

Constraints: None

Semantics: If X is of type Logical, returns TRUE, else FALSE. For applications that do not have a distinct logical type, also ISNUMBER(X) will return TRUE.

## ISNA

Summary: Return True if the parameter is of type NA, else return False.

Syntax: ISNA( Scalar X )

Returns: Logical

Constraints: None

Semantics: If X is NA, return True, else return False. Note that if X is a reference, the value being referenced is considered. This function does not propagate error values.

## ISNUMBER

Summary: Return TRUE if the parameter has type Number, else return FALSE

Syntax: ISNUMBER( Scalar X )

Returns: Logical

Constraints: None

Semantics: If X is of type Number, returns TRUE, else FALSE. Implementations may not have a distinguished logical type; in such implementations, ISNUMBER(TRUE()) is TRUE.

## ISODD

Summary: Return TRUE if the value is even, else return FALSE

Syntax: ISODD( Number X )

Returns: Logical

Constraints: None

Portable Constraints: X must not be Logical

Semantics: First, compute X1=TRUNC(X). Then, if X is odd (a division by 2 has a remainder of 1), return True, else return False. The result is implementation-defined if given a logical value; an application may return either an Error or the result of converting the logical value to a number (per Conversion to Number).

## LN

Return the natural logarithm of a number.

Syntax: LN( Number X )

Returns: Number

Constraints: X>0

Semantics: Computes the natural logarithm (base e) of the given number.

$$\ln x = 2\left[\frac{x-1}{x+1} + \frac{1}{3}\left(\frac{x-1}{x+1}\right)^3 + \frac{1}{5}\left(\frac{x-1}{x+1}\right)^5 + \ldots\right]$$

## LOG

Summary: Return the logarithm of a number in a specified base.

Syntax: LOG( Number N [ ; Number Base = 10 ] )

Returns: Number

Constraints: N > 0

Semantics: Computes the logarithm of a number in the specified base. Note that if the base is not specified, the logarithm base 10 is returned.

## LOG10

Summary: Return the base 10 logarithm of a number.

Syntax: LOG10( Number N )

Returns: Number

Constraints: N > 0

Semantics: Computes the base 10 logarithm of a number.

## MAX

Summary: Return the maximum from a set of numbers.

Syntax: MAX( { NumberSequenceList N }+ )

Returns: Number

Constraints: None.

Semantics: Returns the value of the maximum number in the list passed in. Non-numbers are ignored. Note that if logical types are a distinct type, they are not included. What happens when MAX is provided 0 parameters is implementation-defined, but MAX with no parameters should return 0.

Changes compared to ODF definition: Calling MAX() with 0 parameters causes an error.

## MEDIAN

Summary: Returns the median (middle) value in the list.

Syntax: MEDIAN( { NumberSequenceList X}+ )

Returns: Number

Semantics:

MEDIAN logically ranks the numbers (lowest to highest). If given an odd number of values, MEDIAN returns the middle value. If given an even number of values, MEDIAN returns the arithmetic average of the two middle values.

$n = is\ the\ count\ of\ the\ ranked\ number sequence$

$$\tilde{x} = x_{\left(\frac{(n+1)}{2}\right)}$$
$$for\ n = odd$$

$$\tilde{x} = \frac{1}{2}\left(x_{\left(\frac{n}{2}\right)} + x_{\left(\frac{n}{2}+1\right)}\right)$$
$$for\ n = even$$

## MIN

Summary: Return the minimum from a set of numbers.

Syntax: MIN( { NumberSequenceList N }+ )

Returns: Number

Constraints: None.

Semantics: Returns the value of the minimum number in the list passed in. Returns zero if no numbers are provided in the list. What happens when MIN is provided 0 parameters is implementation-defined, but MIN() with no parameters should return 0.

Changes compared to ODF definition: Calling MIN() with 0 parameters causes an error.

## MOD

Summary: Return the remainder when one number is divided by another number.

Syntax: MOD( Number a ; Number b )

Returns: Number

Constraints: b != 0

Semantics: Computes the remainder of a/b. The remainder has the same sign as b.

## NA

Summary: Return the constant error value #N/A.

Syntax: NA()

Returns: Error

Constraints: Must have 0 parameters

Semantics: This function takes no arguments and returns the error NA.

## NOT

Summary: Compute logical NOT

Syntax: NOT( Logical L )

Returns: Logical

Constraints: Must have 1 parameter

Semantics: Computes the logical NOT. If given TRUE, returns FALSE; if given FALSE, returns TRUE.

## OR

Summary: Compute logical OR of all parameters.

Syntax: OR( { Logical|NumberSequenceList L }+ )

Returns: Logical

Constraints: Must have 1 or more parameters

Semantics: Computes the logical OR of the parameters. If all parameters are False, it shall return False; if any are True, it shall returns True. When given one parameter, this has the effect of converting that one parameter into a logical value. When given zero parameters, applications may return a Logical value or an error.

Changes compared to ODF definition: Calling OR() with 0 parameters causes an error.

## POWER

Summary: Return the value of one number raised to the power of another number.

Syntax: POWER( Number a ; Number b )

Returns: Number

Constraints: None

Semantics: Computes a raised to the power b.

## PRODUCT

Summary: Multiply the set of numbers, including all numbers inside ranges

Syntax: PRODUCT( { NumberSequence N }+ )

Returns: Number

Constraints: None

Semantics: Returns the product of the numbers (and only the numbers, i.e., not text inside ranges). This is equivalent to SUM except that it uses the * operator instead of +.

## QUOTIENT

Summary: Return the integer portion of a division.

Syntax: QUOTIENT( Number A ; Number B )

Returns: Number

Constraints: B <> 0

Semantics: Return the integer portion of a division, e.g., QUOTIENT(-14;5) is -2.

## RAND

Summary: Return a random number between 0 (inclusive) and 1 (exclusive).

Syntax: RAND()

Returns: Number

Semantics: This function takes no arguments and returns a random number between 0 (inclusive) and 1 (exclusive). Note that unlike most functions, this function will typically return different values when called each time with the same (empty set of) parameters.

## ROUND

Summary: Rounds the value X to the nearest multiple of the power of 10 specified by Digits.

Syntax: ROUND( Number X [ ; Number Digits = 0 ] )

Returns: Number

Constraints: None

Semantics: Round number X to the precision specified by Digits. The number X is rounded to the nearest power of 10 given by 10 −Digits. If Digits is zero, or absent, round to the nearest decimal integer. If Digits is non-negative, round to the specified number of decimal places. If Digits is negative, round to the left of the decimal point by -Digits places. If X is halfway between the two nearest values, the result must round away from zero. Note that if X is a Number, and Digits <= 0, the results will always be an integer (without a fractional component).

## SQRT

Summary: Return the square root of a number

Syntax: SQRT( Number N )

Returns: Number

Constraints: N>=0

Semantics: Returns the square root of a non-negative number. This function must produce an error if given a negative number.

## STDEV

Summary: Compute the sample standard deviation of a set of numbers.

Syntax: STDEV( { NumberSequenceList N }+ )

Returns: Number

Constraints: At least two numbers must be included. Returns an error if less than two numbers are provided.

Semantics: Computes the sample standard deviation s, where

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \frac{1}{n-1}\left(\left(\sum_{i=1}^{n} x_i^2\right) - n\bar{x}^2\right)$$

Note that s is not the same as the standard deviation of the set, σ, which uses n rather than n − 1. Implementors must be wary when implementing this function; a naive implementation can easily run into trouble and end up trying to take the square root of a negative number. This is due to various effects of floating point arithmetic, in particular, the problem of using subtraction between two numbers that may be close together in value.

## SUM

Summary: Sum (add) the set of numbers, including all numbers in ranges

Syntax: SUM( { NumberSequenceList N }+ )

Returns: Number

Constraints: None

Semantics: Adds numbers (and only numbers) together (see the text on conversions). Applications may allow SUM to receive 0 parameters (and return 0), but portable documents must not depend on SUM() with zero parameters returning 0.

Changes compared to ODF definition: Calling SUM() with 0 parameters causes an error.

## TRUE

Summary: Returns constant TRUE

Syntax: TRUE()

Returns: Logical

Constraints: Must have 0 parameters

Semantics: Returns logical constant TRUE. Although this is syntactically a function call, semantically it is a constant, and typical applications optimize this because it is a constant. Note that this may or may not be equal to 1 when compared using “=”. It always has the value of 1 if used in a context requiring Number (because of the automatic conversions), so if ISNUMBER(TRUE()), then it must have the value 1.

## TRUNC

Summary: Truncate a number to a specified number of digits.

Syntax: TRUNC( Number a ; Number b )

Returns: Number

Constraints: None

Semantics: Truncate number a to the number of digits specified by b. If b is zero, or absent, truncate to a decimal integer. If b is positive, truncate to the specified number of decimal places. If b is negative, truncate to the left of the decimal point. If b is not an integer, it is truncated.

## VAR

Summary: Compute the sample variance of a set of numbers.

Syntax: VAR( { NumberSequence N }+ )

Returns: Number

Constraints: At least two numbers must be included. Returns an error if less than two numbers are provided.

Semantics: Computes the sample variance s2, where

$$s^2=\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2=\frac{1}{n-1}\left(\left(\sum_{i=1}^{n}x_i^2\right)-n\bar{x}^2\right)$$

Note that s2 is the unbiased variance, which is not the same as the true variance of the set, σ2, which uses n rather than n − 1.

## Additions to the Open Formula Specifications

The mathematical relations are used to define how to calculate the amount of exchanges, properties and parameters. There must be a way to reference several values of the dataset defined outside of the formula. Examples for such values are the amounts of exchanges, properties and parameters. The original Open Formula defines a reference function (p.56) but it is only able to handle references to cells (within a spreadsheet). EcoSpold02 needs an extended reference mechanism defined below.

### Reference to an Amount

When used in a mathematicalRelation the variableName of an entity (exchange, property and parameter) represents its current amount. It is not allowed to reference the variableName of an entity in the mathematicalRelation field of the same entity.

#### Reference by Variable Name

All entities that can be referenced in a mathematicalRelation have a variableName field.  A variable name is a string with a maximum length of 40 characters. It may only contain characters from A-Z, a-z, digits from 0-9 and the underscore "_". The first character must not be a digit. Variable names must be unique within the dataset.

Example:

```
<flowData>
    <productExchange id="4661f76c-513a-4e84-999f-7567bbf56031" unit="kg"
    amount="0.5" isCalculatedAmount="false" variableName="inputDiesel">
        <name xml:lang="en">diesel</name> <inputGroup>5</inputGroup>
    </productExchange>
    <elementaryExchange id="da1aa254-7c85-4b08-a8e9-023682dbdfc7" unit="kg"
    amount="1.5" isCalculatedAmount="true"
    mathematicalRelation="inputDiesel * 3.0">
        <name xml:lang="en">Carbon dioxide, fossil</name>
        <compartment xml:lang="en">air</compartment>
        <subcompartment xml:lang="en">unspecified</subcompartment>
        <outputGroup>4</outputGroup>
    </elementaryExchange>
</flowData>
```

#### Reference by REF Function

Syntax: REF(Text a [; Text b ])

Returns: Number

Constraints: Must have 1 or 2 parameters

Semantics: Returns the amount of the entity referenced by the given parameter(s). Exchanges and Parameters are referenced by the parameter a alone. If an exchange property is referenced both parameter a (the exchange) and b (the property) must be given. The referenced entities must be present in the same dataset as the mathematical relation they are referenced from otherwise an error will be thrown. The REF function can be uses as an alternative to reference by variable name if no variable is defined.

Example:

```
<flowData>
    <productExchange id="4661f76c-513a-4e84-999f-7567bbf56031" unit="kg"
    amount="0.5" isCalculatedAmount="false">
        <name xml:lang="en">diesel</name> <inputGroup>5</inputGroup>
    </productExchange>
    <elementaryExchange id="da1aa254-7c85-4b08-a8e9-023682dbdfc7" unit="kg"
    amount="1.5" isCalculatedAmount="true"
    mathematicalRelation="REF('4661f76c-513a-4e84-999f-7567bbf56031') * 3.0">
        <name xml:lang="en">Carbon dioxide, fossil</name>
        <compartment xml:lang="en">air</compartment>
        <subcompartment xml:lang="en">unspecified</subcompartment>
        <outputGroup>4</outputGroup>
    </elementaryExchange>
</flowData>
```

## Unit Conversion

Syntax: UC( Number a; Text b; Text c)

Returns: Number

Constraints: Must have 3 parameters

Semantics: Returns the parameter a converted from unit b to unit c. The conversion from unit b to unit c must be a supported unit conversion otherwise an error will be thrown.

Example: `UC(20, 'kg', 'g')` returns 20000

Restriction: This formula has not yet been implemented in the ecoinvent context and should not be used. A list of supported unit conversion will be added later.

## Molar Mass Calculation

Summary: Returns the molar weight of a chemical formula

Syntax: MW( Text T)

Returns: Number

Constraints: Must have 1 parameter

Semantics: Returns the molecular weight of the chemical formula given by parameter T. The parameter is case sensitive, e.g. use CO for carbon monoxide and Co for cobalt. If the chemical formula cannot be parsed an error is thrown.

Example: `MW('H2O')` returns 18.01528, `MW('C2H5OH')` returns 46.0694, `MW('CaCl2')` returns 110.9834

Restriction: This formula has not yet been implemented in the ecoinvent context and should not be used.