OpenFormula Format for Office Applications (OpenFormula)

Rough Draft 2005-02-26

Document identifier:

Location:

http://www.dwheeler.com/openformula

Editors:

David A. Wheeler, <dwheeler@dwheeler.com>

Contributors:

Abstract:

This is the specification of the OpenFormula Format, an open format for exchanging formulas in office documents - particularly for spreadsheets. This is intended for use as a supporting document to the Open Document Format for Office Applications (OpenDocument) format. OpenDocument specifies a table:formula attribute and several related attributes, but it does not define the syntax and semantics of those attributes; this specification intends to provide this additional information.

Status:

This document is a draft, and will be updated periodically on no particular schedule.

Table of Contents

1 Introduction

1.1 Introduction

This is the specification of the OpenFormula format, an open format for exchanging formulas used for calculations such as those used in spreadsheet applications. This is intended for use as a supporting document to the Open Document Format for Office Applications (OpenDocument) format [OOo] to define more precisely the semantics of certain formula-related attributes. OpenDocument specifies a table:formula attribute and several related attributes, but it does not define the syntax and semantics of those attributes; this specification intends to provide this additional information. Future versions may expand beyond this limited role.

This is a rough draft; it leaves some things unspecified, and many functions need to be defined in detail. Still, this document provides a lot of information that doesn't seem to be available elsewhere, so even as a rough draft it should be very helpful to implementors of the OpenDocument specification who must process formulas.

1.2 Notation

Within this specification, the key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" are to be interpreted as described in [RFC2119] if they appear in uppercase bold letters.

1.3 Namespaces

Table 1 lists the namespaces that are defined by the OpenFormula format and their default prefixes. For more information about XML namespaces, please refer to the *Namespaces in XML* specification [xml-names].

Table 1: XML Namespaces defined by OpenFormula

Prefix	***Description***	***Namespace***
formula	For formulas (such as for spreadsheets).	FIXME – perhaps: urn:oasis:names:tc:opendocument:xmlns:formula:1.0

1.4 Document Processing and Conformance

Conforming applications **MUST** read and process data that is valid against the minimum OpenFormula specification as described here. Applications **MAY** accept extensions (e.g., additional formulas and operations) as long as they do not interfere with processing data in the format defined here.

1.5 History

On 2004-11-01 David A. Wheeler recommended that the Open Document's specification of formulas be strengthened to allow exchange of formulas (particularly for spreadsheets), including a grammar to eliminate ambiguities. On 2005-01-04 he posted an updated version of his proposal:

http://lists.oasis-open.org/archives/office-comment/200501/msg00000.html

This proposal was formally considered on 2005-01-14 and rejected, as documented in the minutes at:

http://lists.oasis-open.org/archives/office/200501/msg00004.html

The rejection was explained as follows: "A comment was submitted concerning the [inclusion] of a grammar for spreadsheet formulas which conforming implementations should support. While we think that having interoperability on that level would be of great benefit to users, we do not belive that this is in the scope of the current specification. Especially since it is not specifically related to the actual XML format the specification describes. The TC will work on a solution concerning the documentation of interoperabilty standards that go beyond what is defined in the specification. The submitted grammar should be published on the TC web pages."

David A. Wheeler noted that without such a specification of the formulas, implementations would not have a standard way to exchange spreadsheet formulas, rendering the entire specification incomplete with respect to spreadsheets.

James Clark independently discovered this shortcoming in the Open Document format, and posted on 2005-02-04 his concerns. He said he searched “for the section where the Calc formula syntax is specified, and I search, and I search...and I find things like [string]... I really hope I'm missing something, because, frankly, I'm speechless. You cannot be serious. You have virtually zero interoperability for spreadsheet documents. To put this spec out as is would be a bit like putting out the XSLT spec without doing the XPath spec. How useful would that be? It is essential that in all contexts that allow expressions the spec precisely define the syntax and semantics of the allowed expressions. Using a namespace prefix is a nice idea, but it needs to be required (and enforced by the schema), and then you need to define a namespace that contains at least the basic functionality needed by a spreadsheet, and preferably everything supported by OOo 2.0, and precisely define the syntax and semantics of expressions associated with that namespace. OpenDocument has the potential to be [extraordinarily] valuable and important standard. I urge you not to throw away a huge part of that potential by leaving such a gaping hole in your specification.”

Claus Agerskov posted on 2005-02-06 that “it has come to my knowledge that OpenDocument doesn't specify the [formulas] used in spreadsheets so every spreadsheet vendor can implement formulars in their own way without being an open standard. This way a vendor can create lock-in to their spreadsheets. Please [delay] announcing OpenDocument 1.0 until the format for spreadsheet [formulas is] specified.

David A. Wheeler then developed the first draft of this OpenFormula specification to allay these concerns.

1.6 Information Sources

Information used to develop this specification included the following:

1. *OpenOffice.org's Documentation of the Microsoft Excel File Format*. http://sc.openoffice.org. (This location has much other useful information).

2. *Excel 2000 in a Nutshell: A Power User's Quick Reference*. Jinjer Simon. August 2000. O'Reilly.

3. Ximian has a nice LXR setup that lets you easily surf the OpenOffice.org spreadsheet source code at http://ooo.ximian.com/lxr/source/sc/. Of particular interest is the Calc compiler.cxx file, see: http://ooo.ximian.com/lxr/source/sc/sc/source/core/tool/compiler.cxx.

4. Many special tests of various spreadsheet implementations. See http://www.dwheeler.com/openformula.

1.7 Rationale

The following are characteristics of the entire OpenFormula specification, along with rationale for those characteristics:

1. The specification described here is primarily based on what one implementation does (OpenOffice). Basing the specification on an existing implementation means that the approach is known to work in something that already implements the Open Document specification. This text also include notes on the semantics of the widely-used spreadsheet program Microsoft Excel.

2. It is expected that OpenFormula formulas will be preceded by a "namespace selector." For example, OpenDocument implementations' formulas can begin with a namespace prefix to identify the formula specification language to use, or they can begin with an "=" which indicates the OpenFormula specification's namespace. Allowing namespace specification in formulas permits great flexibility and permits upgrading to better formats if OpenFormula is found to be inadequate for that use. However, if there is no "default" formula specification format implemented by all implementations, no two implementations will necessarily be able to exchange formulas. Since this default is expected to be widely used, a simple "=" prefix makes it easy to use the common case, while permitting great flexibility.

3. OpenFormula is defined in a separate document from the OpenDocument standard. This makes it possible to define upgrade the formula language separately from the OpenDocument specification, and allows other implementations to use OpenFormula independently of OpenDocument if they wish to do so.

4. OpenFormula's format does not use XML, but instead uses the traditional mathematical infix notation. Formulas used for calculation have not traditionally been written using XML. Instead, computer-calculatable formulas are traditionally written using a modified mathematical infix notation such as "value*5+2". This non-XML format is much easier for humans to understand (because it is the form people have used for decades), is shorter than XML formats, and there are many tools specifically designed to handle these formats. Note that formulas used primarily for display *are* often written using a nested notation, such as MathML. However, formulas not used for calculation are not the primary subject of this specification.

5. OpenFormula's semantics are intentionally based on previous spreadsheet programs, such as VisiCalc, Lotus 1-2-3, and Microsoft Excel. This was done to simplify transition of spreadsheet documents to this format.

6. OpenFormula unifies the various formula formats defined in OpenDocument. There are many disadvantages to having several similar yet subtly different formula syntaxes, in particular, they tend to increase implementation size (since multiple calculation engines and user interfaces must be developed), and they tend to increase user confusion. As compared with "Open Document Format for Office Applications (OpenDocument) 1.0 Committee Draft 2, 21 Dec 2004", the latest version available at the time of writing, here are the OpenDocument fields that use formulas:

    a) Attribute text:formula defined in 6.7.6 (Formula), and noted in 6.3.2, 6.3.5, 6.3.9, 6.3.10. Note that text:formula is deprecated when inside a table cell (use table:formula instead), though it's allowed; see 6.6.11. Note that variables and most variable fields have a current value, as described in 6.7.1.

    b) Attribute text:condition is defined as being the same as text:formula in 6.3.5.

    c) Attribute table:formula is defined in 8.1.3 (Table Cell) - this is used for spreadsheets.

    d) Attribute table:condition is described in 8.5.3 (Table cell content validations); this format allows certain conditions, many of which can end up calling functions. For example, "cell-content() > [.A5]" would be valid table:condition.

    e) Named expressions as described in section 8.5.5 define cell ranges that may be used as formula variables, and can also include a table:expression that is an OpenFormula formula (in fact, since they cannot begin with "=", they are always in the "formula:" namespace of the document).

    f) This *may* also be used for attribute draw:formula as defined in 9.5.5 (Enhanced Geometry – Equation). It's mentioned in 9.5.3 (Enhanced Geometry - Path Attributes) for enhanced path, text areas, and glue points, and in 9.5.6 too. This is defined in some detail; note that it includes requirements for a number of functions (like sqrt()), and note also that "," is used as the function parameter separator instead of ";". It also permits a "modifier reference" using prefixed "$".

    g) This *may* also be used for attribute anim:formula as defined in 13.3.2., in a way similar to (but not equal to) the draw formulas. Section 9.8.2 (Document Dependent SMIL Animation Attribute Values) subsection "formula" defines additional variables for use in anim:formula: e, x, y, width, height. Again, "," is used as the function parameter separator instead of ";".

    h) Note that attribute draw:equation is also defined in 9.5.5; this is outside the scope of this document.

    i) Section 8.9 (Consolidation) and 8.6.10 (Subtotal Field) notes the use of table:function for consolidating tables and creating subtotals, and gives a fixed list of allowed functions: auto, average, count,countnums, max, min, product, stdev, stdevp, sum, var, and varp. This is outside the scope of this document, though such functions might be implemented using the same code that implements OpenFormula, and implementations MAY allow other OpenFormula functions to be used in table:function.

    j) Section 8.7 describes table filters; these are similar to formulas but are outside the scope of this document (perhaps in future versions they will be brought in-scope, by simply defining them as an OpenFormula formula with an implied "cell-contents()" prefix and permitting cell-contents()).

k) Note that section 6.3.5 gives an example of how to use a namespace to provide a formula that is *not* in the OpenFormula namespace:

```
text:formula='ooo-w:[address book file.address.FIRSTNAME] == "Julie"'
```

Throughout the document rationales are provided. These rationales are included to speed specification development (so that questions need not be re-asked and answered), and to aid implementors (since rationales often help in avoiding common mistakes and "helpful" options that turn out to be harmful). Much discussion about existing implementations is included; where practical, existing implementations should help guide any standard, and these additional notes should help ensure that this is true.

2 Basic Concepts

OpenFormula is a specification for defining and exchanging mathematical formulas to be automatically calculated. It is designed to be used inside typical office documents, such as table cells in a spreadsheet, automatically-calculated values in a word processing document, and for situations in drawings and animations where a formula is needed. It is primarily intended as an exchange format, but this specification includes guidelines on how to present formulas to users and how to accept formulas as user input. Future versions of this specification may expand this role; in particular, since its formula specification is very general, it may find use as a more general way to specify mathematical equations.

Implementations using OpenFormula MAY mark formulas with a "formula specification selector" that identifies which formula specification to use. This marking may be as a separate attribute, or a prefix before the actual formula. As described in the OpenDocument profile, in the OpenDocument attributes table:formula and text:formula formulas MUST be preceded with a namespace prefix (a series of one or more letters) followed by ":", or by a single "=" which has the same meaning as the prefix "formula:". The "formula:" prefix is then defined to be the namespace of a version of this OpenFormula specification (with an appropriate default if the namespace is not defined). The formula specification selector is NOT considered part of the formula, but users will typically see the formula specification selector as the first character(s) of the formula. Implementations MAY accept formula syntaxes other than OpenFormula, and they MAY accept various compatible extensions to the default OpenFormula syntax. However, all implementations that claim to accept OpenFormula formulas MUST accept the minimum formula syntax and semantics as described in this specification. Rationale: This provides additional flexibility so that other languages can be used when desired. The selector is not considered part of OpenFormula itself, so that specifications that use OpenFormula can choose how to best include this selector.

OpenFormula formulas are often embedded inside an XML document. When this occurs, various characters (such as "<", ">", '"', and "&") must be escaped, as described in the XML specification.

Implementations use OpenFormula as a method to exchange formulas. Once an implementation reads a formula in this format, it may choose to represent the formula internally in an arbitrary way (such as a tree of nodes). Formulas are executed whenever they are needed to compute a specific result. Typical implementations will note what values a formula depends on, and when those dependent values are changed, it will re-execute the formulas that depend on them to produce the new results (choosing the formulas in the right order based on their dependencies). When a formula is computed, it is notionally provided a "context" as input. The context may include formula variables (including named ranges, document variables, fields, and so on), and/or additional function definitions that the formula can call. A formula may also be provided as input an ordered list of zero or more parameters (though the syntax for parameters is not given in this version). A formula may include calls to functions, which are normally provided the same context but with their own set of ordered parameters. Any formula computes a single result (thought that single result may actually be a set of values).

OpenFormula is not a full-fledged programming language. It is specifically designed to describe how to calculate a single result, which it returns as its answer. OpenFormula is, in general, side-effect-free; it does not have built-in operations to "assign" a value, and unless otherwise noted its

functions do not have side-effects. Built-in functions, when passed the same values, generally produce the same result (with the notable exception of random number functions like RAND()). It is designed so that it is not possible to describe a calculation in a single function that can “loop forever.” By design, all formula calculations that only reference built-in functions must eventually end (though this may take an extremely long time for a few computationally-intense functions). However, formulas may reference values (such as table cells) which in turn depend on the the value being computed by the formula, creating an endless loop. Implementations SHOULD guard against being stuck in an endless loop that can result from circular calls between multiple formulas (A calls B which calls A which calls B...). Implementations of OpenDocument normally forbid circular references unless the table:iteration element is set, and if set, iterations are allowed up to a specified maximum number of iterations; for more information see OpenDocument section 8.5.2.

OpenFormula is a not a full-fledged mathematical notation. It does not support algebraic symbolic manipulation. Also, it does not support arbitrary control over the display of formulas; see MathML and other notations for this capability.

3 Syntax

The minimum syntax of OpenFormula is provided below; any OpenFormula MUST support this syntax for data exchange of formulas. It is defined using the BNF notation of XML version 1.1, as described at http://www.w3.org/TR/2004/REC-xml11-20040204/#sec-notation. A formula is defined as follows:

```
formula ::= expression

expression ::=  NUMBER | STRING |
                source_location formula_variable |
                expression binary_op expression |
                unary_op expression |
                "(" expression ")" |
                IDENTIFIER "(" expression_list ")" |  /* "function" */
                cell_specifier |
                array_spec

formula_variable ::= IDENTIFIER

/* See precedence rules, described below, for these operations. */
unary_op ::= '+' | '-'

binary_op ::= comparison_op | '+' | '-' | '&' | '*' | '/' | '^'

comparison_op ::= '<' | '>' | '<=' | '>=' | '=' | '<>'

parameter ::= expression

nonempty_expr_list ::= parameter | nonempty_expr_list ';' parameter

expression_list ::=  /* empty */ | nonempty_expr_list

cell_specifier ::= "[" barecellspecifier "]"

bare_cell_specifier ::= source_location cell_range
                      cell_intersection cell_concatenation

cell_range ::= cell ( ":" cell )?

cell ::= "$"? sheet_name "." "$"? [A-Z]+ "$"? [0-9]+

sheet_name ::= /* empty */ | "$"? [A-Za-z0-9]+ |
              "$"? "'" [^']+ "'"

/* Note: MUST support absolute URLs.  SHOULD support
   relative URLs, which can be distinguished because they
   do not begin with [A-Za-z]+ ":".  If the name looks like
   an absolute URL, save using "./" prefix, e.g., './mystuff.xls' */
source_location ::= /* empty */ | "'" url "'" "#"

cell_intersection ::= ( "!" cell_range )*

cell_concatenation ::= NOT_IMPLEMENTED /* see below for this extension.
*/

array_spec ::= NOT_IMPLEMENTED /* see below for this extension */

/* TODO: check if sourcelocation prefix to IDENTIFIER is valid. */
```

In short, a formula must be an expression. An expression must be a number, a constant string, an identifier, a pair of expressions connected by a binary operator, an expression prefixed by a unary operator, an expresssion surrounded by parentheses, a function call or logical operator, or a cell specifier. A function call and logical operator may have zero or more semicolon-separated parameters, and each parameter MUST be an expression. The syntax for each of these is as follows:

- Numbers. Numbers are written and read in this format using the "C" locale (which uses the "." as the decimal separator and never uses a thousands separator), using POSIX setlocale (LC_NUMERIC, "C") or equivalent. Numbers can end in %, which divides that number by 100. The “%” does not change the meaning of other operators, so 2+10% is 2.1 (not 2.2). Note that leading - and + signs are allowed as unary operators, described below. Writers MUST write numbers that match the pattern (note that it must begin with a digit):
  [0-9]+(\.[0-9]+)?([eE][+-]?[0-9]+)?%?
  Readers MUST be able to read these numbers, as well as accept numbers that begin with a leading “.”, so they must be able to read numbers in the form:
  ((\.[0-9]+)|([0-9]+(\.[0-9]+)?([eE][+-]?[0-9]+)?))%?

- Constant Strings.. Constant strings are surrounded by double-quotes; to embed a double-quote, the double-quote character is used twice. Strings are exchanged in UTF-8 format. Note that since when formulas are contained in an XML's attribute, all double-quotes in the formula are actually quoted (e.g., stored as " in the XML). Constant strings match the pattern: \"([^"]|\"\")*\"

- Formula variables. Formula variables are named values which can be retrieved from the formula's context. Examples include named ranges in a spreadsheet, and both variables and fields in a word processing document. Note that some formula variables may be a single value (e.g., a named range defined as a single cell) or a set of values (e.g., a named range of a selected set of cells). Names are not case-sensitive, so “a” and “A” refer to the same range. Implementations must accept at least named ranges whose names match the following regular expression: [A-Za-z][A-Za-z0-9_]*
  Note that the exchange format is unambiguous – any cell specifier is surrounded by “[“..”]”, while formula variables are not. However, most implementations display and allow users to enter formula variables *without* the surrounding square brackets, which means that the range of legal formula variable names cannot have the name as a cell address in such cases. *Excel 2000 in a Nutshell*, page 120, gives the following rules for its names. A name must:

  - begin with a letter or underscore
  - be followed with zero or more letters, digits, underscore (_), period (.), question mark (?), or backslash. FIXME: Note that it permits periods, which complicates entry (since “.” is used as the separator between sheetnames and cell addresses in OpenFormula). Note that spaces are not permitted.
  - cannot exceed 255 (and really can't exceed 253, since then selecting the name box is a problem).
  - cannot match cell's address. A cell address is the one or two letter column label A through IV (uppercase, lowercase, or combination) followed by a number between 1 and 65535. It also cannot match the obsolete RrCc row-column name format (where r is a number from 1 to 255 and c is a number from 1 to 65535).

- Operators. Ordinary infix and prefix operators are accepted. These have the following associativity and precedence (from lowest to highest priority):

Associativity	*Operator(s)*	*Comments*
left	<,=,>,<=,>=,<>,!=,==	Less than, equal to, greater than, less than or equal to, greater than or equal to, not equal to, not equal to (alternative), equal to (alternative). Writers should normally write = and <> for equals and not-equals, but be able to read == and !=. Rationale: OOo spreadsheet uses = and <> to implement table:formula, but != and == are widely-used C notation, and for interoperability's sake it appears wiser to accept both.
left	+,-,&	Binary operations add, subtract, string concatenation. Note that unary (prefix) + and – has a different priority. Note that "&" must be escaped when included in an XML document, typically as "&".
left	*,/	Multiply, divide. Division does not truncate, so 1 / 2 is equal to 0.5.
right	^	Power (2^3 is 8). Readers SHOULD also accept "**".
nonassociative	+,-	Prefix unary operators, e.g., -5 or -[.A1]. Note that these have a difference precedence than add and subtract.

Precedence can be overridden by using parentheses, so "2+3*4" computes 14 while "(2+3)*4" computes 20.

Logical operators, function calls, and certain constants have the same syntax. They are:

- Logical operators. Logical operators have the same syntax as function calls; their names are case-insensitive, parameters are separated by semicolons, and their name MUST be followed by parentheses. The logical operators are:

Logical Operator	*No. of Parameters*	*Comments*
TRUE()	0	This is a boolean constant, though its syntax makes it appear like a function
FALSE()	0	This is a boolean constant
NOT(expression)	1	If expression is TRUE() returns FALSE(), else returns TRUE()
AND(e1; e2 [; e]*)	2 or more	If all expressions are TRUE() returns TRUE(), else returns FALSE()
OR(e1; e2 [; e]*)	2 or more	If all expressions are FALSE() returns FALSE(), else returns TRUE()

Logical Operator	*No. of Parameters*	*Comments*
IF(condition; true_exp; false_exp)	3	Evaluates *condition*. If it's true, return true_exp, else return false_exp

Implementations of AND(), OR(), and IF() MUST short-circuit, that is, they must evaluate left-to-right in turn, and only evaluate the expressions they *must* evaluate to compute the result. An implementation may choose to evaluate more, but only when the expressions have no side-effects. Implementations of AND() and OR() SHOULD accept an arbitrary number of parameters, but MUST accept at least 30 in each use. The operations NOT(), AND(), and OR (), as well as the condition in IF(), are intended for boolean values; if expressions of other types are used, an implementation SHOULD NOT consider 0 as false and any other numeric value as true, and SHOULD NOT consider a zero-length string as false and any other string value as true. If an error value is computed for an expression, then that first error is the result of the logical operation.

- Function calls. A function call has a function name matching the pattern [A-za-z][A-Za-z0-9_]* followed by an opening parenthesis, zero or more parameters, and a closing parenthesis. Parameters are separated by a semicolon (not a comma), though readers MAY optionally accept function calls using commas as separators as well. Function names are case-insensitive, so “sum” and “SUM” are the same function.

  If there are parameters, in the default syntax given above each must be an expression and none can be empty, so X(;) is not a legal function call while RAND() is perfectly legal. NOTE: Microsoft Excel accepts empty parameters in any position; OpenOffice 1.1.3 does not, so the current syntax above doesn't either. See below for how this rule can be relaxed. Typical implementations will have many built-in functions, and most implementations also support one or more ways to create user-defined functions.

  > Rationale: Many locales use “,” as the decimal separator; using the semicolon as the parameter separator eliminates confusion. It would be possible to use the “,” as a separator and use “.” as the decimal point in the exchange format, but this means either that the humans cannot use “,” as a decimal point (even where it's common) or that the exchange representation is unnecessarily different from the human input representation, increasing the likelihood of problems.

- Addresses of cell(s) that contain values. The addresses can be relative or absolute, see section 8.3.1 of the OpenDocument specification for more information about how to address a cell or cell range. Addresses in formulas being exchanged start with a “[” and end with a “]”; this makes parsing simpler, faster, and more reliable (but note that typical spreadsheet displays will often *not* display or require input of the square brackets). During computation these addresses will be replaced with the current value(s) of these cell(s). Cells may be turned into cell ranges by separating cells with “:”. Cell ranges may, in turn, be intersected with other ranges using the “!” operator (Excel displays this as a space). Note that if OpenFormula is used in situations where there are no tables that can be referenced, then it is an error to include cell addresses.

Potential future changes: Note that if future versions are to handle OpenDocument's anim:formula and draw:formula, they will need to add variable syntax such as $0, $a, and $. In addition, the issue of using “,” instead of “;” as a function parameter separator will need to be resolved to accep those formulas. Future versions may also accept constant array specifiers,

e.g., “{“ list “}” - and in that list, there must be a way to separate different values inside a row as well as a way to separato different rows (Excel uses “,” to separate entries within a row, and “;” to separate entries in different rows). Note that Excel also supports concatenation of cell addresses using the comma, e.g., (B1:B3,B2:B4); this must be surrounded by extra parentheses if within a parameter list (or the separator would be misinterpreted as ending the parameter).

Whitespace (space, tab, newline, and carriage return) is ignored in the default formulas syntax, except in the contents of string constants. Note that in Microsoft Excel's display format, the space is also used as a separator for multiple cell range addresses in a cell intersection (this is confusing, so OpenFormula uses “!” instead).

The syntax above requires that function call syntax have 0 parameters, *or* that it have 1 or more parameters each of which are non-empty. This is simply because OpenOffice.org 1.1.3 requires this. However, Microsoft Excel accepts empty parameters. Thus, a common optional extension would be to accept empty parameters at any point. One complication is that this creates a minor ambiguity - expressions such as “PI()” can be read as either having 0 parameters or as having 1 empty parameter. At this time, this specification suggests considering this as having 0 parameters (there is currently no way to call a function giving it one parameter, the empty parameter). This changes the syntax above as follows:

```
nonempty_expr_list ::= /* Not used */

parameter ::= /* empty */ | expression

expression_list ::=  parameter ( ";" parameter )*
```

The specification above has a NOT_IMPLEMENTED symbol, which identifies common extensions that are not required but are recommended for OpenFormula implementations.

This specification does not mandate a cell concatenation operator, since OpenOffice.org 1.1.3 does not include one. However, Microsoft Excel includes this operation. In Excel this is represented using the comma (“,”) character, which is a very poor choice with a number of unfortunate ramifications. One problem is that concatentating cells in a function parameter requires surrounding the cells with additional parentheses in Excel display syntax. For example, AREAS(A1:A3,B2:B4) is a function call with two parameters, while AREAS((A1:A3,B2:B4)) is a function call with one parameter. Another problem is that the comma interferes with the use of “,” as a decimal separator (as it is used in many locales) when using traditional entry formats (which do not mark cell addresses with “[“..”]”). Gnumeric uses “+” as the cell concatenation operator, but this has its own problems: it interferes with the use of “+” as a matrix addition operator. There are many alternatives, e.g., other characters (such as “~”, “|”, and “\”), or requiring a function syntax for this purpose. The character “_” would be a poor choice because formula variables can also include this character in their name (complicating parsing when “[..]” are not used – is B3_B2 a formula variable, or B2 and B3 concatenated?). This draft proposes using “~” as the cell concatenation symbol. Thus, implementations that support the cell concatenation operation should support this syntax:

```
cell_concatentation ::= /* empty */ | '~' bare_cell_specifier
```

The syntax above does not include a specification for arrays specified in-line. Microsoft Excel includes a way to write arrays of constants without having to create a cell range for them. OpenOffice.org 1.1.3 does not support this. Excel's display format uses a comma as the separator between values in a row, and a semicolon for the separator between rows. Here

semicolon is used as the separator between values in a row (again, so different locales will have a simpler time entering data when entering data), and the pipe symbol “|” is proposed as the symbol separating rows (with absolutely no precedent). Here is a proposed syntax, which in addition permits non-constant values in the array (something Excel does not permit) and concatenation (FIXME: I don't think Excel permits them either):

```
array_spec ::= "{" matrix ( '_' matrix )* "}"

matrix ::= ( matrix_row ( column_separator matrix_row )* ) ?

matrix_row ::= expression ( ";" expression )*

column_separator ::= "|"
```

4 Types

4.1 Set of Types

Any cell may be empty, or it may have a value. A value may have one of the following types: **number**, **string**, **boolean** (also called logical), **error**, or **set**:

- A number is simply a numeric value such as 0, -4.5, or $1000. Numbers *must* be able to represent fractional values (they may *not* be limited to only integers); typical implementations store numbers as 64-bit IEEE floating point internally and use the CPU's floating-point instructions where available (so intermediate values may actually have more than 64 bits to represent them). Implementations MAY choose to use fixed-point, exact (fractional) representations, arbitrary-length numbers, and so on to implement numbers. A cell with a constant numeric value has the number type. The “number” type may be displayed in many different formats, including date, time, percentage, and currency. Typical implementations try to heuristically determine the “right” format for a cell when a formula is first created, based on the operations in the formula. However, users can override this format and the heuristic varies between implementations; thus this heuristic is outside the scope of this specification.
- A string is a string of 0 or more characters. A cell with a label has the string type; from the point of view of OpenFormula, a cell with a label is no different from a cell whose formula produced a string with the same value. Implementations *should* support Unicode strings, but MUST at least support strings of ASCII characters. In implementations that support Unicode, strings MAY be represented internally using a variable-width representation such as UTF-8 or UTF-16, but unless otherwise noted, any reference to a particular index inside a string MUST be by *character* not *byte position.* This means implementations must at maintain the illusion of fixed-width characters, even if they use variable-length encodings.
- A boolean is a value with one of two values: TRUE() and FALSE(). Note that these are a distinct type from numbers.
- An error is one of a set of possible error values. Implementations may havemany different error types, but one in particular is distinct: NA() (written as #N/A in Excel). Users may choose to enter some data values as NA(), so that this error value propogates to any other formula that uses it, and may test for this using the function ISNA().
- A set is an ordered collection of non-set values. It may be subdivided further into ranges, each of which have one or more rows and one or more columns. A special case of set is a *matrix*, a rectangular connected range of cells with rows and columns.

  FIXME: “References” should probably be their own type. See the sc.openoffice.org's docuemntation of Excel, b/c it appears several functions handle them specially.

4.2 Unary and Binary Operations

The binary operations +, -, *, /, and ^, when provided two numbers, *must* compute (respectively) the addition, subtraction, multiplication, division, and exponentiation of those numbers. The unary operation “-” must, when provided a number, produce the negative of this number. If for some reason the operation cannot be performed (e.g., division by zero or out-of-range for the

numeric representation being used), an error value must be returned. If any value provided to these operations is an error, then an error value must be the result. If both operands of a binary operator are errors, the first (left-hand) operand is returned.

This specification currently leaves undefined what happens if other types are used with these operations. Indeed, there is variance between current spreadsheets. In Excel and Gnumeric, +, -, *, /, ^, and unary - convert any boolean or string values they use into numbers before computing them (essentially wrapping them with a call to N() first). TRUE() is converted to 1, and FALSE() is converted to 0. Strings are turned into their numeric equivalent, and that includes strings that match date or time formats (which are turned into their numeric equivalents). Strings that cannot otherwise be converted are converted to the value 0. However, in OpenOffice.org 1.1.3, strings and logical values are ignored (or considered 0), even if they *could* be converted to a numeric type. This means that in Excel, SUM(A1:A2) and A1+A2 are different if A1 or A2 are not numbers (e.g., the string “11”); the “+” forces conversion to a numeric, while SUM(A1:A2) does not.

Unary + simply passes on its operand's value, unchanged, with exactly the same type.

The string concatenation operator “&” converts any non-string values into strings before concatenating them. Again, if any value being provided is an error, the result is an error; if more than one error is provided, the first (left-hand) operand is returned as the result.

The comparison operators compare two different values. If the values are both numbers, they are compared numerically. FIXME: other types.

5 Standard Functions

The following are definitions of standard functions. Function names ignore case, so “sum” and “SUM” refer to the same function. Unless otherwise noted, if any value being provided is an error, the result is an error; if more than one error is provided, the first one is returned. Some “functions”, such as PI(), always return the same constant value, and can be considered constants.

Common functions include:

- SUM() - Sums up all the numbers in the expressions it is given. Note that SUM *ignores* non-numerics such as string types or boolean types; it does *not* attempt to convert them to numeric values (this is true for Excel and OpenOffice.org 1.1.3).
- COUNT() - counts the number of numbers in its range.
- AVERAGE() - SUM(range) / COUNT(range)
- MIN() - minimum value of the referenced values
- MAX() - maximum value of the referenced values

FIXME: TBD. Eventually should provide a large set of functions and their semantics. But even without this information, this specification has some value.

FIXME: In Excel, SUM() skips non-numbers and does NOT convert boolean (logical) or text types. This means that A1+A2 is *not* the same as SUM(A1:A2): if A1 or A2 are not a number, the “+” operator attempts to convert the value to a number, while SUM() simply ignores the values. Perhaps define several standard function names (SUMA(), SUMN()) that have specific functions, and then state that SUM() must be an alias to one of them?

David A. Wheeler noted in a February 25, 2005 posting to the OpenDocument mailing list that OpenOffice.org 1.1.3's semantics are DIFFERENT than Microsoft Excel's. This means that syntactically converted spreadsheets can produce different answers in OOo, compared to Excel. It's not a matter of "right" or "wrong"; what OOo is doing is reasonable. But it's different than Excel, producing different results. From the point-of-view of compatibility, they are compatibility bugs, IF you believe that Excel semantics need to be matched to enable easy transition and interoperation. The test Excel spreadsheet at http://www.dwheeler.com/openformula/testss.xls shows some of the differences. For example:

- In Excel, Logical values (TRUE, FALSE) are not considered numbers; they aren't counted by COUNT(). They ARE in OOo.
- In Excel, SUM(...) is NOT the same as num + num + num ...; SUM _only_ looks at values of type "numeric", ignoring strings and logical values. In OOo, SUM(...) also ignores strings/text, but OOo converts any logical values to 0 and 1, so TRUE() adds one in OOo but not in Excel.
- In Excel, ISNUMBER(TRUE()) is false. In OOo, ISNUMBER(TRUE()) is true.

- ROMAN() works slightly differently when given TRUE() and FALSE() as format parameters.
- OOo does not support constant arrays, or cell concatenation.

If Excel's are the "right" semantics, then I can document that "+" requires certain operations, etc., etc. But if that's not the case, then I don't know what to document. For functions, it's easy to define one function name as doing one thing, another function name that does the other, and then let Excel converters match the names up... but that requires implementations of those functions, and agreement on the names.

6 User Input/Output

User interfaces of implementations MAY choose to accept and display formulas differently from how they are exchanged in this data format. For example, they MAY accept and display numbers using the format of the current locale, they MAY always use a particular locale for numeric formats, they MAY accept and display commas instead of semicolons for parameter separators, and they MAY accept and display cell addresses without requiring the use of square brackets.

But implementation user interfaces SHOULD allow users to directly enter OpenFormula format formulas as input as well where possible. Indeed, the OpenFormula specification has been designed to permit direct human entry of these formulas. For example, implementations SHOULD accept numbers that meet the “C” locale requirements (as well as the current locale's), SHOULD accept bracketed cell addresses, and SHOULD accept the semicolon as a function parameter separator. Also, implementation user interfaces SHOULD correct likely mistakes, possibly with a dialogue. For example, if an implementation normally uses semicolons as a parameter separators, an interactive implementation SHOULD attempt to detect if a user is using commas as function parameter separators instead of semicolons, and recommend a plausible correction. Implementations SHOULD have a mode or options so that a user can choose to view OpenFormula formulas in this format.

In the exchange format, “!” is the cell intersection operation; in Excel this is entered and displayed as a space.

Note that spreadsheet implementations SHOULD accept user input of a cell address without the “[..]” markings, and automatically determine if the user intended to reference a named range or a cell address. Excel simply forbids creating a rangename that could also be a cell address; in many ways this is the simplest solution. An implementation could simply search for a named range, and if there is no such range and the entry could be a cell address, the implementation automatically promotes it to a cell address. This means that if a table has a range named “Q1” (meaning “first quarter”), any future formula where “Q1” is entered will refer to the named range; referring to cell address Q1 will require being surrounded by square brackets. Rationale: It's normal for spreadsheet users to simply type in cell addresses and expect them to work, while the [] format is rarely used directly (even OpenOffice.org 1.1.1, which stores formulas using [..], doesn't accept user input using []). However, there should be a way for users to disambiguate between named ranges and cell addresses. By allowing direct user entry and display of the OpenFormula format, users are more likely to be able understand their formulas, users can more easily move between applications, and developers can more easily identify and debug any interoperability problems. Note that the OpenFormula format is highly accessible, aiding those with disabilities.

Excel and OpenOffice.org surround formulas that are written to a range of cells with “{“ and “}” when they are displayed. These characters are not inlcuded in the data interchange format.

7 OpenDocument Profile

OpenFormula is designed to smoothly fit with OpenDocument; the following provides specific information on what is required of implementations that implement, and documents that use, both OpenDocument and OpenFormula.

Implementations MAY accept extensions to the OpenFormula specification defined here, but they SHOULD NOT write information using an extension if there is a method described in this specification for representing the information instead.

An OpenDocument document implementing the strict schema MAY NOT use any extensions in formulas (including additional functions) beyond the semantics and syntax described in this specification, since there is no guarantee that other receiving systems would be able to process it.

Users MUST NOT claim that an OpenDocument document completely meets the strict schema if it uses a formula namespace selector to reference any formula specification language other than OpenFormula, Again, this is because there is no guarantee that the receiver will understand the other formula specification. An OpenDocument document MAY be claimed to be conforming to "Strict schema plus X" if it meets the strict schema and only uses the additional formula specification namespace or set of functions labelled X. Rationale: Users need to be able to easily specify that a document meets a particular standard so that it can be easily exchanged.

The OpenDocument attributes that use formulas at the time of this writing are text:formula, text:condition, table:formula, draw:formula, and anim:formula. Note in particular that:

1. In text:formula and text:condition, the formula context MUST include all variables and fields' current values, which can be referenced by name as a formula variable.
2. In table:formula (which is used for spreadsheets and word processing tables) it is always legal to include a table cell reference, since there is at least one table. Not all syntactically legal table cell references are actually legal, e.g., a named table might not exist. The formula context MUST include all named ranges that apply to that table.
3. In draw:formula a "modifier reference" (using prefixed "$") MUST be legal. (FIXME: should this be supported at all?)
4. In anim:formula the context MUST always include the following: e, x, y, width, height, as described in the OpenDocument specification. (FIXME: should this be supported at all?)

OpenDocument attributes that use formulas MUST begin with a formula specification selector that identifies which formula specification to use. This selector MUST be either a namespace (one or more letters) followed by a colon, or the equal sign ("="). The prefixed "=" is equivalent to "formula:". OpenDocument documents SHOULD define the formula namespace; if they do not, the default value for the formula namespace is version 1.0 of OpenFormula (this specification). The "formula:" namespace MUST be some version of this specification.

The formula specification selector is NOT considered part of the formula, but users will typically see the formula specification selector as the first character(s) of the formula. Implementations MAY accept multiple formula syntaxes, and they MAY accept various extensions to the default formula syntax. However, all implementations that claim to accept OpenFormula formulas MUST accept the formula syntax and semantics as described in this specification. Rationale:

This provides additional flexibility so that other languages can be used when desired. The selector is not considered part of OpenFormula itself, so that specifications that use OpenFormula can choose how to best include this selector.

8 Examples (Non-normative)

Here are some examples of OpenFormula formulas – what is exchanged, as well as user displays for two common spreadsheet programs. Note that in the exchange format, cell references are surrounded by "[..]" (which simplifies parsing and disambiguates them from formula variables), but typical implementations do not display or require the square brackets to be entered. Function parameters are separated with ";". For simplicity, the leading "=" is displayed in all three formats, though technically that's just a selector for OpenFormula:

OpenFormula Exchange Formula	***Microsoft Excel Display***	***OpenOffice.org 1.1.3 Display (en_US Locale)***	***Comments***
`=1+2*3^4`	`=1+2*3^4`	`=1+2*3^4`	Arithmetic precedence – the answer must be 163
`=salary*2`	`=salary*2`	`=salary*2`	Variables permitted
`=[.A2]*3.5`	`=A2*3.5`	`=A2*3.5`	Exchange format surrounds cell references with "[".."]", and row/column address is preceded by ".". C locale used for numbers
`=sum ([.A1:.A5])`	`=sum(A1:A5)`	`=sum(A1:A5)`	":" creates a range
`=sum([.A1]; [.A2];[.A3])`	`=sum(A1,A2,A3)`	`=sum(A1;A2;A3)`	Parameters are ";" separated (Excel uses ",")
`=MMULT (Matrix1;Matri x2)`	`{=MMULT (Matrix1,Matri x2)}`	`{=MMULT (Matrix1;Matrix 2)}`	Array results not surrounded by "{..}"; variable names
`=[$Sheet1.B2]`	`=Sheet1!B2`	`=$Sheet1.B2`	Sheet references (in same workbook)
`=['file:///C:/ dwheeler/misc/ testssold.xls' #$Sheet1.B3]`	='C:\dwheeler\misc\ [testssold.xls] Sheet1'!B3	`='file:///C:/dw heeler/misc/tes tssold.xls'#$Sh eet1.B3`	Referencing another file
`=SUM ([.B2:.B3]! [.B1:.B2])`	=SUM(B2:B3 B1:B2)	`=SUM (B2:B3!B1:B2)`	'!' is cell inter-section (Excel uses space character)
`=SUM (B1:B3~B2:B4)`	`=SUM ((B1:B3,B2:B4))`	`=SUM (B1:B3~B2:B4)`	(Optional) Cell concatenation – NOT union- B2 will be added twice. ~ proposed.

OpenFormula Exchange Formula	*Microsoft Excel Display*	*OpenOffice.org 1.1.3 Display (en_US Locale)*	*Comments*
=AREAS(B1:B3~ B2:B4)	=AREAS((B1:B3, B2:B4))	=AREAS(B1:B3~ B2:B4)	(Optional) Cell concatenation again
=MDETERM({2;3| 4;5})	=MDETERM ({2,3;4,5})	=MDETERM({2;3| 4;5})	(Optional) Constant arrays, proposed syntax