
XMLmind XML Editor - Using the Integrated Spreadsheet Engine

Hussein Shafie, Pixware <xmleditor-support@xmlmind.com>

December 2, 2005

Abstract

This guide contains everything you need to know to use the spreadsheet engine integrated in XXE. This document starts with an easy-to-follow tutorial.

Table of Contents

1. Tutorial	1
1.1. Basics	2
1.1.1. Three more formulas to finish first version of the invoice	3
1.2. How does this really work?	4
1.3. Relative references	5
1.4. Absolute references	6
1.4.1. Better absolute references	6
1.5. External references	7
1.6. Formatting	8
2. The language used to write formulas	11
2.1. Statements and comment lines	11
2.2. Expressions	12
2.3. Function calls	12
2.4. Literals	13
2.4.1. Numbers	13
2.4.2. Strings	13
2.4.3. Booleans	13
2.5. References	13
2.5.1. Reference to a variable	13
2.5.2. Reference to an element having a specific ID	13
2.5.3. Reference to table cells	13
2.6. XPath escapes	15
2.7. Value types	15
2.8. Automatic conversion between different value types	16
3. Defining custom spreadsheet functions	18
3.1. Registering custom spreadsheet functions with XXE	18
3.2. Specifying custom spreadsheet functions	19
3.3. Custom spreadsheet functions written in the Java™ programming language	21

1. Tutorial

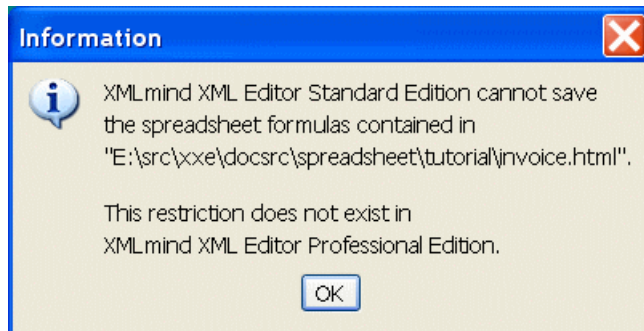
Please open `XXE_install_dir/doc/spreadsheet/tutorial/invoice_table.html`¹ and immediately save it as `invoice.html` in the same directory.

¹In `XXE_install_dir/doc/spreadsheet/tutorial/`, you'll also find:

- `products.html`, the product list used in this tutorial.
- `invoice_done.html`, the invoice after finishing this tutorial.
- `invoice_template.html`, a ready-to-use, empty, invoice, having all the needed formulas.

Note

Standard Edition users, do not be surprised when you'll see this popup:



XMLmind XML Editor Standard Edition will display it

- when you insert the first formula in a document not containing formulas,
- and if you open a document already containing formulas (created using Professional Edition).

This popup informs you that XMLmind XML Editor Standard Edition *will strip all the formulas* from the files you are going to save.

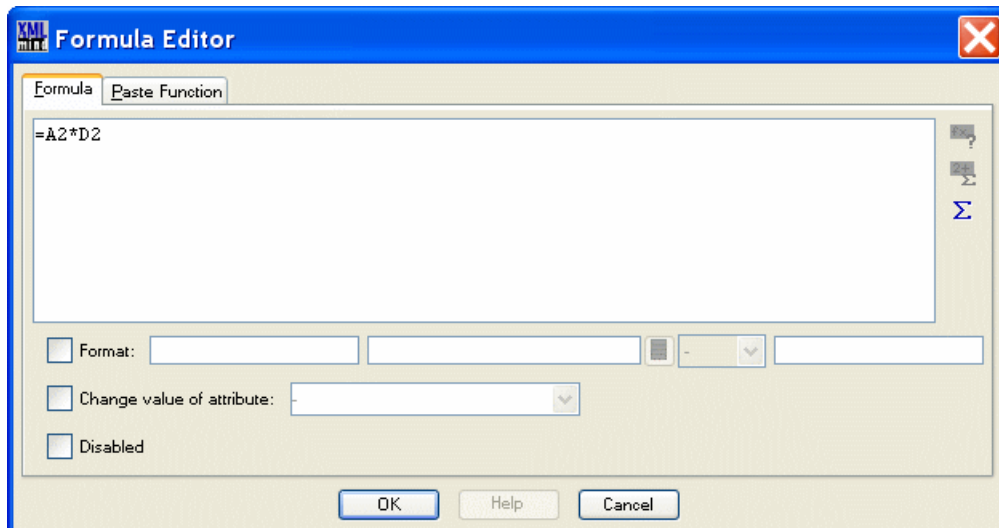
1.1. Basics

Use Tools|Spreadsheet|Show Table Labels (shortcut **Ctrl-Shift-Y**) to make the table look like a spreadsheet.

	A	B	C	D	E
1	Qty	Product	Description	Unit Price	TOTAL
2					
3					
4					
5	SubTotal				
6	VAT			19.6%	
7	TOTAL				

Click in cell E2 and use Tools|Spreadsheet|Insert/Edit Formula (shortcut **Ctrl-Shift-I**) to insert a new formula at caret position.

The Formula Editor is displayed:



After sign '=', type A2*D2 and then click OK (shortcut **Ctrl-Enter**).

You have inserted your first formula in the document. A formula is visually represented by a small F icon. Don't worry if you find it ugly: it will disappear when you'll print the document or when you'll convert it to other formats.

Click on the F icon. Notice that:

- The node path bar displays: /html/body/table/tbody/tr/td/#processing-instruction(xxe-formula), which means that a formula is processing instruction xxe-formula (more on this later).
- A red line is drawn around the F icon, which means that you have explicitly selected this processing instruction.
- The status bar displays: =(A2*D2). The small F icon is in fact a special purpose button (more on this later).

Copy the formula (**Ctrl-C**) to the clipboard and paste it (**Ctrl-V**) in cells E3 and E4.

	A	B	C	D	E
1	Qty	Product	Description	Unit Price	TOTAL
2					NaN
3					NaN
4					NaN
5	SubTotal				
6	VAT			19.6%	
7	TOTAL				

Now type in cell A2: 2, tab to cell B2 to type: xe-1u, tab again to cell D2 and type: 200. Do the same in the next two rows: 1,xe-5u,800 and 1,fc-uu,3000. Then click in cell E5 to force an update. The table should now look like this:

	A	B	C	D	E
1	Qty	Product	Description	Unit Price	TOTAL
2	2	xe-1u		200	400
3	1	xe-5u		800	800
4	1	fc-uu		3000	3000
5	SubTotal				
6	VAT			19.6%	
7	TOTAL				

1.1.1. Three more formulas to finish first version of the invoice

1. Click in cell E5 and insert formula: =sum(E2:E4)

- Click in cell E6 and insert formula: `=rounddown(E5 * left(D6, len(D6)-1)%, 2)`

`left(D6, len(D6)-1)` is string "19.6%" without its last character '%'.
Note that you can use spaces in a formula and that a formula is case-insensitive. For example: `SUM(e2:e4)` works fine too.

- Click in cell E7 and insert formula: `=E5+E6`

	A	B	C	D	E
1	Qty	Product	Description	Unit Price	TOTAL
2	2	xe-1u		200	400
3	1	xe-5u		800	800
4	1	fc-uu		3000	3000
5	SubTotal				4200
6		VAT	19.6%		823.2
7	TOTAL				5023.2

1.2. How does this really work?

In XMLmind XML Editor, a formula is stored as the `xxe-formula` processing instruction. A processing instruction such as `xxe-formula` is allowed by the XML standard. Such processing instructions will be ignored by all XML software except XMLmind XML Editor.

Unlike in "real" spreadsheet software:

- A formula is *not* a "computed table cell", which is itself a value you can reference in other formulas. A formula is a special XML object which can be inserted anywhere in the XML document (including inside a table cell element, at an arbitrary nesting level).
- A formula computes a value, but is not itself a value you can reference in other formulas. This computed value is added just after the `xxe-formula` processing instruction. If there is already some text just after the `xxe-formula` processing instruction, this text is replaced.

Optionally the computed value can be used to set/replace the value of an attribute of the element containing the `xxe-formula` processing instruction.

- A formula can access any part of the XML document (using XPath escapes). When the formula has an ancestor element which is formatted as a table cell, using the customary A1 notation to reference table cells is possible. When no styled view is used to render the XML document or when the formula has no ancestor element which is formatted as a table cell, A1-style cell references will not work.

In a styled view, a formula is rendered using a special purpose gadget, which is at the same time an indicator and a button. Its color gives you a hint about the state of the formula.

Icon	Description
	Unknown state. Formulas contained in document modules included in the document being edited are ignored by the spreadsheet engine.
	Parse error. Should not happen if you use the Formula Editor.
	Evaluation error.
	OK.
	Disabled. Disabling a formula means passivating it. That is, it is no longer used to update the document. In some cases, this is a handy alternative to removing it.

Clicking on the icon triggers special actions:

Simple click

If the formula cannot be parsed (red icon) or evaluated (orange icon), displays the corresponding error message in the status bar. If the formula is working, displays the last statement of the formula (a formula can contain several statements, see The language used to write formulas).

Double click

Opens the Formula Editor to edit the formula.

Click with middle button

Disables (gray icon) or enables (green icon) the formula.

1.3. Relative references

Select row #4 (for example by Ctrl-clicking 3 times in a cell of row #4), copy it to the clipboard (**Ctrl-C**) and paste the copied row after row #4 (**Ctrl-W**).

	A	B	C	D	E
1	Qty	Product	Description	Unit Price	TOTAL
2	2	xe-1u		200	400
3	1	xe-5u		800	800
4	1	fc-uu		3000	3000
5	1	fc-uu		3000	3000
6	SubTotal				6800
7	VAT			19.6%	1332.8
8	TOTAL				8132.8

Ouch! SubTotal E6 is wrong. It should be 7200. What happened here?

The first formula we have created was `=A2*D2`. This formula was inserted in cell E2. This formula uses relative cell references which means that XXE understands it as: add the cell which is 4 columns to my left to the cell which is 1 column to my left.

If you really wanted to add cell at (row 2,column A) to cell at (row 2,column D), whichever is the cell containing the formula, you should have typed `=A2*D2`.

Using relative references in a formula is handy because it allows the formula to be copied and pasted elsewhere. Remember that we have copied first formula to E3 and E4 and that we have duplicated row #4, which means that we have copied the formula of E4 to E5.

If you click on the formula of E3 (which is a copy of the one in E2), you'll see `=(A3 * D3)`. Similarly, on E4, you'll see `=(A4 * D4)` and on E5, `=(E5 * D5)`.

No, XXE has not modified the formulas that you have copied. XXE has stored exactly the same formula in E2, E3, E4, E5 but it displays it differently when you click on different cells.

And because, unlike "real spreadsheet software", XXE never modifies your formulas, SubTotal E6 is wrong! Click on E6 and you'll see `=sum(E3:E5)`. Real spreadsheet software would have modified the formula to be `=sum(E2:E5)`.

Fortunately, there is a way to fix this kind of problem. Double-click on the formula of E6 to open the Formula Editor and replace `=sum(E3:E5)` by `=sum(difference(E:E,E6:E1000))`. This means add everything in column E (E:E) except all cells after E6 (E6:E1000). Et voilà, this is fixed once for all! You can now freely add and remove purchased products to the table without worrying about the SubTotal.

Duplicate row #5 as we did for row #4. In new row #6, replace 1,fc-uu,3000 by 1,sc-cs,100. Then delete row #5 which is a copy of row #4.

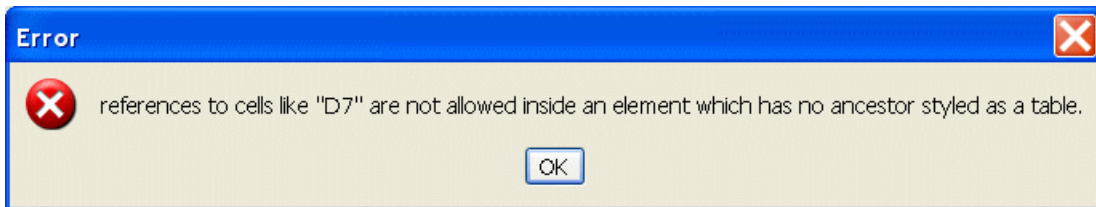
	A	B	C	D	E
1	Qty	Product	Description	Unit Price	TOTAL
2	2	xe-1u		200	€ 400
3	1	xe-5u		800	€ 800
4	1	fc-uu		3000	€ 3000
5	1	sc-cs		100	€ 100
6	SubTotal				€ 4300
7	VAT			19.6%	€ 842.8
8	TOTAL				€ 5142.8

1.4. Absolute references

Use the Attribute tool to add an ID to the table. Select the table using the node path bar, click on the line starting with `id` in the Attribute tool and type `invoice_table`, then press **Enter**.

Add a paragraph containing sentence "The VAT rate is the VAT rate of France.". After the "The VAT rate", use **Ctrl-Shift-I** to insert a new formula.

Try to specify this formula as `= "invoice_table"!D7`. This does not work.



Outside a table, relative references are not allowed. Now specify `= "invoice_table"!D7`.

It works because we have used absolute references but this has deleted everything after "The VAT rate ". This is normal. Remember that the value computed by a formula is added just after the `xxe-formula` processing instruction. If there is already some text just after the `xxe-formula` processing instruction, this text is replaced.

8	TOTAL	€ 5142.8
The VAT rate € 19.6%		

Click on the formula to select it. Use Select|Extend Selection to Following Sibling (shortcut **Esc Right-Arrow**) and then Edit|Convert (shortcut **Ctrl-T**) to convert the two selected nodes to a span.

8	TOTAL	€ 5142.8
The VAT rate € 19.6%		

Now press **Insert** to insert a new text node after the newly created span and type once again " is the VAT rate of France.".

8	TOTAL	€ 5142.8
The VAT rate € 19.6% is the VAT rate of France.		

What you have learned here is that, unless a formula is inserted in a table cell which contains nothing else, you'll almost always have to wrap it in its own element (typically `span` for XHTML and `phrase` for DocBook).

1.4.1. Better absolute references

Using `= "invoice_table"!D7` to copy the content of cell D7 is not a good idea. Adding and removing purchased products to the table would change the last sentence to something that does not make sense.

Formulas can very easily reference elements by their ID and that's what we are going to do. Click on cell D7 and specify an attribute `id` for it. We have already done that for the table. This time, specify `VAT` as the value of attribute `id` of `td` D7.

Now double click on the formula contained in last sentence and, using the Formula Editor, change `= "invoice_table"!D$7` to `=(VAT)`.

Notice that this time, the end of the sentence, `"is the VAT rate of France."`, has not disappeared.

We are going to double-check this by manually triggering a full calculation of the spreadsheet. Use Tools|Spreadsheet|Update (shortcut **Ctrl-Shift-R**) for that.

About automatic calculation of the spreadsheet

By default, the spreadsheet engine is in auto-update mode.

In auto-update mode, a full calculation is automatically performed, if needed to, when the *editing context* changes. For example: type some text in a paragraph, then click in (or tab to) another paragraph to trigger a spreadsheet calculation.

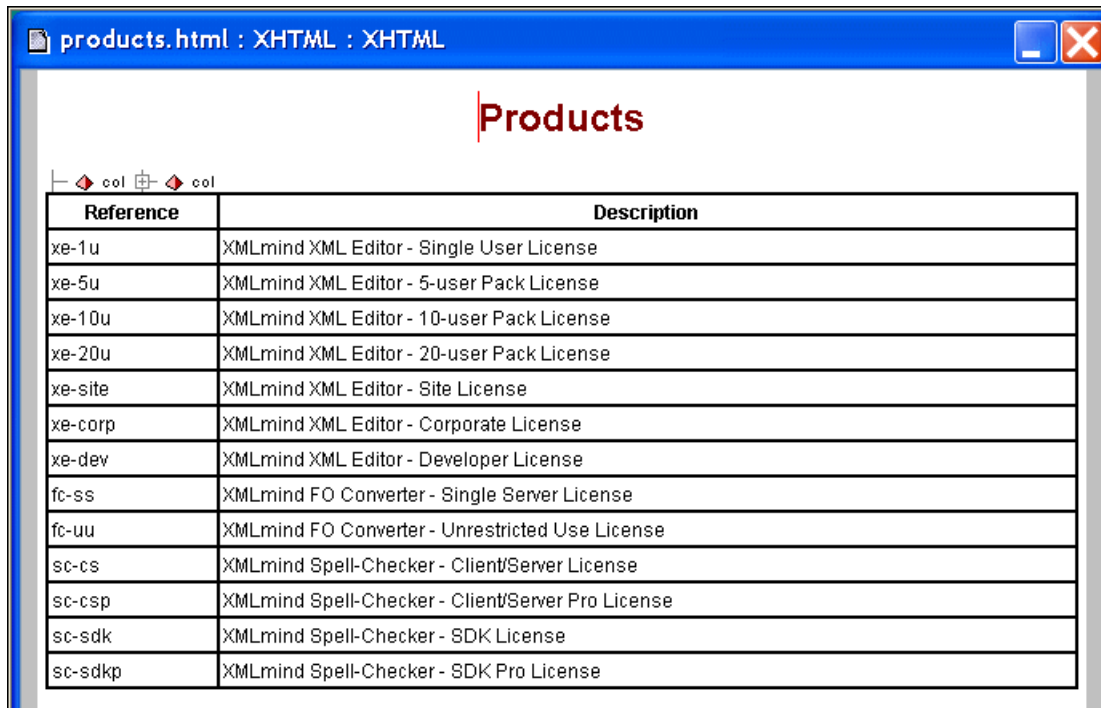
In manual update mode, only newly inserted formulas are computed. To force a full calculation, the user has to explicitly use Tools|Spreadsheet|Update.

Note that in both modes, a full calculation is automatically performed, if needed to, before validating or saving the document.

Using manual update mode is recommended if you have a slow computer or if you have inserted a lot of formulas in your document or if your formulas access many external documents (more about this in next section).

1.5. External references

Instead of typing product descriptions, we are going to use a formula to fetch them from an external XML document. This document is `xxe_install_dir/doc/spreadsheet/tutorial/products.html`.



Reference	Description
xe-1u	XMLmind XML Editor - Single User License
xe-5u	XMLmind XML Editor - 5-user Pack License
xe-10u	XMLmind XML Editor - 10-user Pack License
xe-20u	XMLmind XML Editor - 20-user Pack License
xe-site	XMLmind XML Editor - Site License
xe-corp	XMLmind XML Editor - Corporate License
xe-dev	XMLmind XML Editor - Developer License
fc-ss	XMLmind FO Converter - Single Server License
fc-uu	XMLmind FO Converter - Unrestricted Use License
sc-cs	XMLmind Spell-Checker - Client/Server License
sc-csp	XMLmind Spell-Checker - Client/Server Pro License
sc-sdk	XMLmind Spell-Checker - SDK License
sc-sdkp	XMLmind Spell-Checker - SDK Pro License

Click in cell C2 and use **Ctrl-Shift-I** to insert a new formula. Enter this multi-line formula:

```
location = "products.html#" & trim(B2) & "_desc"
=document($location, .)`
```

First line is easy to understand. It assigns to local variable `location` a string built using the code of the product: "products.html#xe-1u_desc". "product.html" is the URL, relative to the location of the document being edited, of the external XML document containing product descriptions. "xe-1u_desc" is the ID of the element containing the description of product `xe-1u`.

Second line contains an XPath escape. The expression between backquotes ```, which uses standard XPath function `document()`, allows to fetch a node found in an external document. Without the "#xe-1u_desc" fragment identifier, the whole document node is fetched. With the "#xe-1u_desc" fragment identifier, we fetch the element node having `xe-1u_desc` as its ID.

	A	B	C	D	E
1	Qty	Product	Description	Unit Price	TOTAL
2	2	xe-1u	XMLmind XML Editor - Single User License	200	400
3	1	xe-5u		800	800
4	1	fc-uu		3000	3000
5	1	sc-cs		100	100
6	SubTotal				4300
7	VAT			19.6%	842.8
8	TOTAL				5142.8

Copy the new formula to the clipboard (**Ctrl-C**) and paste it (**Ctrl-V**) into C3, C4, C5.

	A	B	C	D	E
1	Qty	Product	Description	Unit Price	TOTAL
2	2	xe-1u	XMLmind XML Editor - Single User License	200	400
3	1	xe-5u	XMLmind XML Editor - 5-user Pack License	800	800
4	1	fc-uu	XMLmind FO Converter - Unrestricted Use License	3000	3000
5	1	sc-cs	XMLmind Spell-Checker - Client/Server License	100	100
6	SubTotal				4300
7	VAT			19.6%	842.8
8	TOTAL				5142.8

1.6. Formatting

There is something obviously wrong in our invoice: numbers are very poorly formatted. Replace the unit prices of column D by \$200.00, \$800.00, \$3,000.00, \$100.00.

	A	B	C	D	E
1	Qty	Product	Description	Unit Price	TOTAL
2	2	xe-1u	XMLmind XML Editor - Single User License	\$200.00	NaN
3	1	xe-5u	XMLmind XML Editor - 5-user Pack License	\$800.00	NaN
4	1	fc-uu	XMLmind FO Converter - Unrestricted Use License	\$3,000.00	NaN
5	1	sc-cs	XMLmind Spell-Checker - Client/Server License	\$100.00	NaN
6	SubTotal				0
7	VAT			19.6%	0
8	TOTAL				0

The problem now is that the formulas no longer work². The reason is that, if string "3000" can be automatically be converted to a number when used in formula `=A4*D4`, a string such as "\$3,000.00" cannot be automatically be converted to a number.

In order to fix this, we need to use spreadsheet function `numbervalue()`. This function must be used to convert a string representing a localized number to something usable by the spreadsheet functions and operators. Function

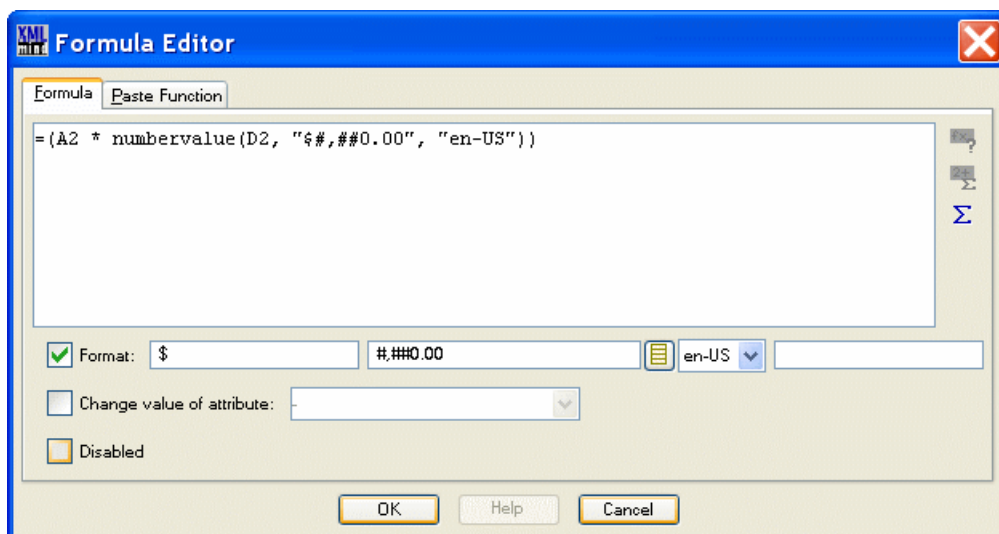
²NaN is a special number which means "Not a Number".

numbervalue() must be passed a number format and optionally, a locale which specifies how to interpret this format.

Click on the formula of cell E2 and replace =(A2 * D2) by =(A2 * numbervalue(D2, "\$#,##0.00", "en-US")). Specifying "en-US" is normally not necessary when you work on a computer using the en-US locale. We could have written =(A2 * numbervalue(D2, "\$#,##0.00")). But we want this tutorial to run unchanged whatever is the computer used to learn XMLmind XML Editor.

While we are at it, we'll also make the value computed by the formula of cell E2 good looking. In order to do that:

- Click on the Format toggle below the text area.
- Type "\$" in first text field.
- Type "#,##0.00" in second text field.
- Choose "en-US" from the combobox containing locales.



We could have used spreadsheet function text() to accomplish the same formatting task, but separating the calculation from the formatting of the result by the means of the Format fields will make your formulas easier to read.

	A	B	C	D	E
1	Qty	Product	Description	Unit Price	TOTAL
2	2	xe-1u	XMLmind XML Editor - Single User License	\$200.00	\$400.00
3	1	xe-5u	XMLmind XML Editor - 5-user Pack License	\$800.00	NaN
4	1	fc-uu	XMLmind FO Converter - Unrestricted Use License	\$3,000.00	NaN
5	1	sc-cs	XMLmind Spell-Checker - Client/Server License	\$100.00	NaN
6	SubTotal				0
7	VAT			19.6%	0
8	TOTAL				0

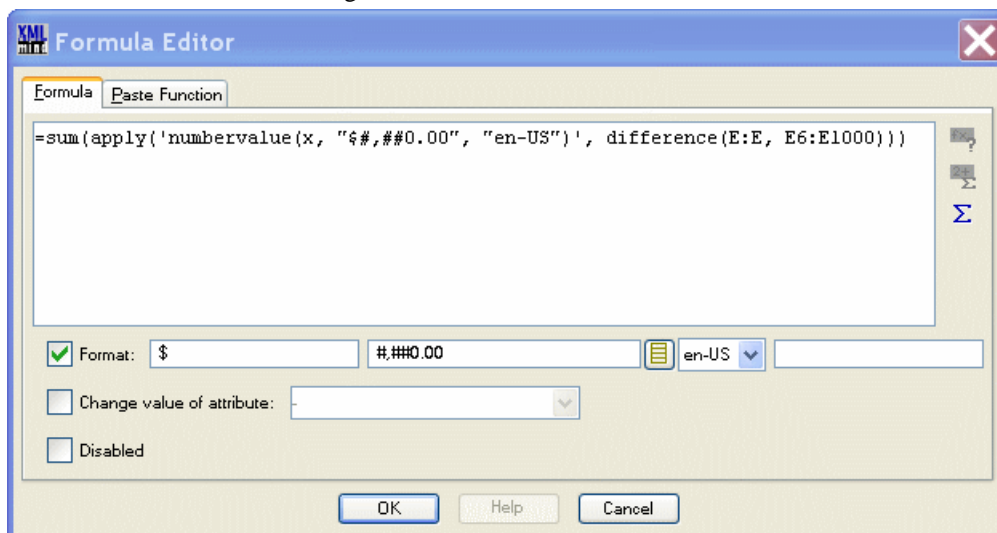
Do not bother fixing by hand the formulas of cells E3, E4, E5. Simply copy the formula found in cell E2 (**Ctrl-C**) then click to select the formula of cell E3 and finally use paste (**Ctrl-V**) to replace it by the content of the clipboard. Repeat the operation with cell E4 and cell E5.

XMLmind XML Editor - Using the Integrated Spreadsheet Engine

	A	B	C	D	E
1	Qty	Product	Description	Unit Price	TOTAL
2	2	xe-1u	XMLmind XML Editor - Single User License	\$200.00	\$400.00
3	1	xe-5u	XMLmind XML Editor - 5-user Pack License	\$800.00	\$800.00
4	1	fc-uu	XMLmind FO Converter - Unrestricted Use License	\$3,000.00	\$3,000.00
5	1	sc-cs	XMLmind Spell-Checker - Client/Server License	\$100.00	\$100.00
6	SubTotal				0
7	VAT			19.6%	0
8	TOTAL				0

The formula of cell E6 is trickier to fix: `=sum(difference(E:E, E6:E1000))`. In order to do this, we need to use spreadsheet function `apply()`.

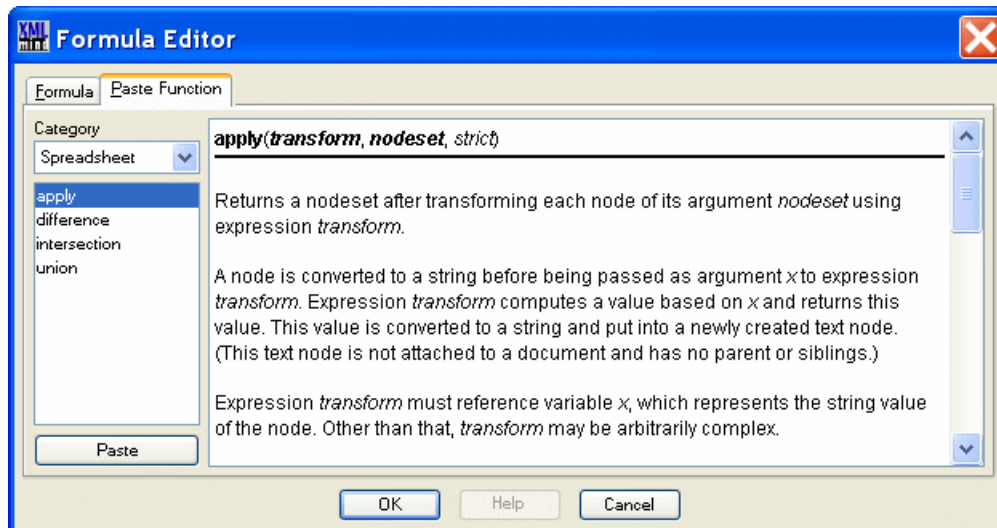
Function `apply()` applies a transformation to each node of a nodeset. Here the nodeset is: `difference(E:E, E6:E1000)`. The transformation that we need to apply is `numbervalue(x, "$#,##0.00", "en-US")`, where `x` represents the string value of the iterated node. This gives:



Note that the transform argument of `apply` is specified as a string and that we have alternated single and double quotes to make this string easier to read.

Tip

The documentation of spreadsheet functions is available online. If, for example, you don't remember how to use function `apply()`, simply select the word `apply` in the text area of the Formula Editor and press **F1**.



Finally fix the formulas found in cells E7 and E8:

- Double-click on the formula of cell E7 and replace `=rounddown((E6 * (left(D7, (len(D7) - 1))))%, 2)` by `=rounddown((numbervalue(E6, "$#,##0.00", "en-US") * (left(D7, (len(D7) - 1))))%, 2)`. Also use the Format fields.
- Double-click on the formula of cell E8 and replace `=(E6 + E7)` by `=(numbervalue(E6, "$#,##0.00", "en-US") + numbervalue(E7, "$#,##0.00", "en-US"))`. Also use the Format fields.

The invoice is finished. It is now possible to remove the labels we put around the rows and columns of the table. This is done by using Tools|Spreadsheet|Hide Table Labels (shortcut **Ctrl-Shift-Z**).

invoice.html : XHTML : XHTML

Invoice

Invoice

Qty	Product	Description	Unit Price	TOTAL
2	xe-1u	XMLmind XML Editor - Single User License	\$200.00	\$400.00
1	xe-5u	XMLmind XML Editor - 5-user Pack License	\$800.00	\$800.00
1	fc-uu	XMLmind FO Converter - Unrestricted Use License	\$3,000.00	\$3,000.00
1	sc-cs	XMLmind Spell-Checker - Client/Server License	\$100.00	\$100.00
			SubTotal	\$4,300.00
			VAT	19.6%
			TOTAL	\$5,142.80

The VAT rate 19.6% is the VAT rate of France.

2. The language used to write formulas

The language used to write formulas is *case insensitive*.

2.1. Statements and comment lines

A formula can contain several statements, blank lines and comment lines. Example:

```
x = 2
```

```
# This a comment line  
  
y = 2  
= x + y
```

- Spaces are allowed inside statements.
- Statements must end with a newline.
- All statements except last one are used to assign to local variables the values of intermediate expressions.
- Last statement is the result of the formula. It must start with '='.
- Some identifiers are reserved: TRUE, FALSE and all references to table cells: A1, B2, DC273, etc. Do not use reserved identifiers for the names of your local variables. Example: x=2 is OK, x1=2 is not.

2.2. Expressions

Expressions use the following operators to combine primary expressions (function calls, literals, references, etc). Operators are listed from highest priority to lowest priority.

- *-number* negation like in -1.
- *number%* percentage like in 10%
- *number1^number2* exponentiation: 2^3=8
- *number1*number2* multiplication. *number1/number2* division.
- *number1+number2* addition. *number1-number2* subtraction.
- *string1&string2* string concatenation: "black"&"white"="blackwhite".
- *value1=value2* (equal). *value1<>value2* (different). *number1<number2* (less than). *number1<=number2* (less than or equal). *number1>number2* (greater than). *number1>=number2* (greater than or equal).

Unlike in other spreadsheet software, there are no union and intersection operators. Use the union() and intersection() functions. (The difference() function is also very useful.)

Unlike in other languages, there are no test, logical and, logical or and logical negation operators. Use the following functions for that: if(), and(), or(), not().³

Parentheses may be used to group subexpressions. Example:

```
= 2+2*3
```

means:

```
= 2 + (2*3)
```

because the priority of '*' is higher than the priority of '+'. If, in fact, you did not intended to write this, you must use parentheses:

```
= (2 + 2)*3
```

2.3. Function calls

The syntax of a function call is: *function_name(argument, argument2, ..., argumentN)* whatever is the locale of the computer running XMLmind XML Editor. Examples:

³The documentation of functions is exclusively available online. Use the Paste Function tab of the Formula Editor to browse it.

```
PI()  
sin(x)  
log(x, 10)  
max(A1, A2, 100.0)
```

2.4. Literals

2.4.1. Numbers

Numbers are always written the same whatever is the locale of the computer running XMLmind XML Editor: 1, 2.3, 314E-2, 0.314e1, 0.314e+1, etc.

2.4.2. Strings

String must quoted using double quotes (") or single quotes ('). They cannot contain newline characters.

In a string quoted using double quotes, the double quote character must be escaped by doubling it. In a string quoted using single quotes, the single quote character must be escaped by doubling it.

Examples:

```
"It doesn't matter"  
'It doesn''t matter'  
"Did you say "bizarre"?"  
'Did you say "bizarre"?'
```

2.4.3. Booleans

TRUE and TRUE() (case insensitive of course) specify logical value true. FALSE and FALSE() specify logical value false.

2.5. References

2.5.1. Reference to a variable

There is nothing special to do to reference a variable, except that a variable needs to have been assigned a value before being referenced. Example:

```
x = 2  
x = x + 1  
= 2*x*x
```

2.5.2. Reference to an element having a specific ID

The syntax of this type of reference is: \$(ID), where ID is the ID of an element contained in the same document as the formula. Example taken from the tutorial:

```
= left($(vat), len($(vat)) - 1)
```

This type of reference returns an *XML nodeset* containing a single element. If there is no element having specified ID, the reference returns the empty nodeset. More on this below.

2.5.3. Reference to table cells

The following references are said to be *relative*:

- First cell of a table: A1.
- Second row: 2:2.
- Second column: B:B.

- First two rows: 1:2.
- First two columns A:B.
- Four cells in the top/left corner of the table: A1:B2.

The following references are said to be *absolute*:

- First cell of a table: \$A\$1.
- Second row: \$2:\$2.
- Second column: \$B:\$B.
- First two rows: \$1:\$2.
- First two columns \$A:\$B.
- Four cells in the top/left corner of the table: \$A\$1:\$B\$2.

Mixed references are possible too: A\$1, \$A1, 2:\$2, \$2:2, etc.

What does this mean? Example: cell \$A\$3 (first column, third row) contains the following formula:

```
= A1 + A2
```

Because the formula uses relative references, XXE translates this to:

```
= $[-2,0] + $[-1,0]
```

which means add the cell which is 2 rows above me to the cell which is 1 row above me.

If you copy the formula of \$A\$3 to \$B\$3 (second column, third row), the copied formula will add \$B\$1 to \$B\$2 because the cell which is 2 rows above \$B\$3 is \$B\$1 and the cell which is 1 row above \$B\$3 is \$B\$2.

On the other hand, if cell \$A\$3 contained the following formula:

```
= $A$1 + $A$2
```

Because the formula uses absolute references, XXE would have translated this to:

```
= $[1,1] + $[1,2]
```

which means add the cell which is at first column, first row to the cell which is at first column, second row.

With absolute references, after copying the formula of \$A\$3 to \$B\$3, the copied formula would still add \$A\$1 to \$A\$2.

Important

- Relative references are allowed only when the formula is inside a table cell.
- References to cells, whether relative or absolute, are really possible when the XML document is displayed using a styled view.

A reference such as \$A\$3 cannot be evaluated unless the formula has an ancestor element formatted as a table.

A reference such as "Income"!\$C\$4 cannot be evaluated unless element with ID Income is formatted as a table.

2.5.3.1. The `$(row,column)` notation

This notation is the one which is internally used by XXE. You can type `$(row,column)` references if you want, but XXE will never show you these references as you typed them. XXE (the status bar, the formula editor, etc) will automatically display cell references using the customary A1 notation, which is much more readable.

Absolute reference examples: `A1 = $(1,1)` `$2:$2 = $(2,)` `$B:$B = $(,2)` `$1:$2 = $(1,):$(2,)` `$A:$B = $(,1):$(,2)` `A1:B2 = $(1,1):$(2,2)`

Relative reference examples, the formula being inside cell `A3`: `A1 = $(-2,0)` `2:2 = $(-1,)` `B:B = $(,+1)` `1:2 = $(-2,):$(-1,)` `A:B = $(,0):$(,+1)` `A1:B2 = $(-2,0):$(-1,+1)`

2.5.3.2. Reference to cells in another table of the document

The element formatted using CSS property "display:table;" must have an ID. The syntax is: `"ID"!cell_reference`. Examples: `"Income"!C4`, `"table-23"!A:A` (relative references are allowed too if the formula has itself an ancestor element formatted as a table).

2.6. XPath escapes

The language used to write formulas is very close to the one used by other spreadsheet software. The rationale is of course to make it easy to learn. But in fact, formulas are internally translated to another, less known language: XPath 1.0. XPath can be considered as the standard, native, expression language of XML.

This design allows to freely mix XPath expressions with "ordinary" expressions. This is what we call XPath escapes.

The syntax of XPath escapes is simply: ``XPath_expression``. (The character used here is the backquote ``.)

Inside `XPath_expression`, the backquote character `` must be escaped by doubling it. Example: ``concat("XXE ", " is ````challenging to learn".")``.

An XPath expression can reference the local variables of the formula using the usual XPath syntax for that: `$variable_name`.

Using XPath escapes is often mandatory. Example 1: add 2 to the value of attribute `count` of the element containing the formula:

```
= `@count` + 1
```

Example 2 taken from the tutorial: get element with ID `B2&"_desc"` found in external document `"products.html"` (pathname relative to the file containing the document being edited).

```
location = (("products.html#" & trim(B2)) & "_desc")
=document($location, .)`
```

This XPath-based design also allows to use XPath functions as if they were regular spreadsheet functions. In order to do this, simply add an underscore `_` at the beginning of the XPath function name and, if this name contains dashes `-`, replace them by underscores. Examples:

- `_contains("Large", "e") = TRUE`
- `_substring_before("Large", "e") = "Larg"`

2.7. Value types

Most operators expect their operands to have a specific type and return a result having a specific type. Examples: `+` expects 2 numbers and returns a number, `&` expects two strings and returns a string.

Most functions expect their arguments to have a specific type and return a result having a specific type. Examples: `left()` expects a string and a number and returns a string, `cos()` expects a number and returns a number.

When the operand or the argument does not have the expected type, an automatic conversion is performed if this is possible; otherwise an error is reported.

Example that works: `cos(B2)` where cell B2 contains a string "3.14" which can be parsed as a number.

Example that sort of works: `2 + "two"` (gives ``number" NaN, which is not really a number).

Example that reports an error (because function `sum()` has been specified to do that): `sum(2, "two")`.

This automatic conversion process is detailed in next section.

The types of values returned by expressions are:

Number

Examples: .23, 3.14, 314E-2, `PI()`, `2+2`, `log(A2,10)`.

String

Examples: "Great", 'Not that great', "black" & "white", `mid(D3, 3, 4)`.

Boolean

Examples: TRUE, FALSE, `true()`, `false()`, `1>=0`, `and(A1 >= 1, A1 <= 99)`.

Nodeset

A set of XML nodes: Examples: `A1`, `A:A`, `$2:$3`, `"table-23"!A1`, `$(vat)`, ``id("product-list")/*``, `apply("x + 1", B:B)`.

Date

Year/month/day. Examples: `today()`, `date(1960,3,16)`, `datevalue("1960-03-16")`, `datevalue("3/16/60", "MM/dd/yy")`.

Time

Hour/minute/seconds/fraction of a second. Examples: `time(13,30,45)`, `timevalue("13:30:45")`, `timevalue("1:30:45 PM", "hh:mm:ss a")`.

DateTime

Year/month/day and hour/minute/seconds/fraction of a second: Examples: `now()`, `datetime(1960,3,16,13,30,45)`, `datetimevalue("1960-03-16T13:30:45")`, `datetimevalue("1:30:45.250 PM 3/16/60", "hh:mm:ss.SSS a MM/dd/yy")`.

2.8. Automatic conversion between different value types

The behavior of operators such as `+`, `*`, `=`, `<>`, `<=`, `>`, etc, is the one of equivalent XPath operators.

Surprising example: using `=` and `<>` to compare node-sets:

1	two
two	3

`(A:A = B:B)`, `(A:A = B1)` and `(A:A <> B2)` all work and evaluate to true.

Fortunately the behavior of spreadsheet function such as `sum()`, `rounddown()`, etc, is almost identical to the behavior of similar functions found in other spreadsheet software. This behavior has no XPath equivalent.

The automatic conversion process is the one described in the XPath standard.

From\To	Number	String	Boolean	Nodeset	Date / Time / DateTime
Number	-	Conversion does not add superflu-	0 and NaN are converted to	ERROR	A number is taken to be the number

From\To	Number	String	Boolean	Nodeset	Date / Time / DateTime
		ous zeros after the point (e.g. it generates 1, not 1.0). Scientific notation (e.g. 314E-2) is never used.	FALSE. Other numbers are converted to TRUE.		of seconds since January 1, 1970, 00:00:00 GMT. This number is converted to the corresponding datetime.
String	Strings such as 3.14, -2 can be parsed as numbers. Strings such as 3 1 4 E - 2 or 1,000,000.0 cannot be parsed as numbers (use function number-value() to do this). A string which cannot be parsed as a number is converted to NaN.	-	A string is TRUE if its length is non-zero.	ERROR	Strings using the ISO 8601 format (also used by W3C XML Schema Data-types) are successfully converted to date, time and datetime. Examples: 1960-03-15Z (date), 13:30:00Z (time), 1 9 6 0 - 0 3 - 1 6 T 1 2 : 3 0 : 0 0 Z (datetime)
Boolean	TRUE is converted to 1. FALSE is converted to 0.	TRUE is converted to "true". FALSE is converted to "false".	-	ERROR	ERROR
Nodeset	A nodeset is first converted to a string and then this string is converted to a boolean.	String value of the node in the nodeset that is first in document order. Text contained in descendant nodes of this first node is taken into account. Except that text contained in comments and processing-instructions is ignored. E x a m p l e : The little/a <!--pussy-->cat is chasing a mouse. converted to a string gives "The little cat is chasing a mouse.".	A nodeset is TRUE if it is non empty.	-	ERROR

From\To	Number	String	Boolean	Nodeset	Date / Time / DateTime
Date / Time / DateTime	Date, time and datetime are converted to the number of seconds since January 1, 1970, 00:00:00 GMT. (The ``date used for a time" is January 1, 1970 GMT.)	Date, time and datetime are represented using the ISO 8601 format (also used by W3C XML Schema Data-types). Examples: 1960-03-15Z (date), 13:30:00Z (time), 1960-03-16T12:30:00Z (datetime)	Date, time and datetime are always TRUE.	ERROR	-

3. Defining custom spreadsheet functions

3.1. Registering custom spreadsheet functions with XXE

Custom spreadsheet functions must be defined in an XML document conforming to (XMLmind proprietary) W3C XML Schema having `http://www.xmlmind.com/xmleditor/schema/spreadsheet/functions` as its target namespace.

Example, `myspreadsheetfunctions.xml` (found in `XXE_install_dir/doc/configure/samples2/`)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<f:functions xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ns="http://www.w3.org/1999/xhtml"
  xmlns:f="http://www.xmlmind.com/xmleditor/schema/spreadsheet/functions">
  <f:function>
    <f:name>factorial</f:name>
    <f:parameters>n</f:parameters>
    <f:category>Mathematical</f:category>
    <f:description>
      <body>
        <p>Returns the factorial of <i>n</i>.</p>
      </body>
    </f:description>
    <f:macro><![CDATA[= if(n = 1, 1, n*factorial(n-1))]></f:macro>
  </f:function>
</f:functions>
```

This XML document must be referenced in an XXE configuration file using configuration element `spreadsheet-Functions`.

- If you add this to `XXE_user_preferences_dir/addon/customize.xxe` (after copying `myspreadsheetfunctions.xml` to `XXE_user_preferences_dir/addon/`):

```
<spreadsheetFunctions location="myspreadsheetfunctions.xml" />
```

you'll be able to use custom function `factorial()` whatever is the type of the document that you open in XXE. That is, function `factorial()` and its documentation will always show up in the Formula Editor, just like `sin()` or `cos()`.

XXE user preferences directory is:

- `$HOME/.xxe/` on Unix,
- `%SystemDrive%\Documents and Settings\%USERNAME%\Application Data\XMLmind\XMLeditor\` on Windows 2000/XP,
- `%SystemDrive%\winnt\Profiles\%USERNAME%\Application Data\XMLmind\XMLeditor\` on Windows NT.

- If you add this to an XXE configuration file which is specific to an XML application, for example `XXE_install_dir/addon/config/docbook/docbook.xxe` (after copying `myspreadsheetfunctions.xml` to `XXE_install_dir/addon/config/docbook/`):

```
<spreadsheetFunctions location="myspreadsheetfunctions.xml" />
```

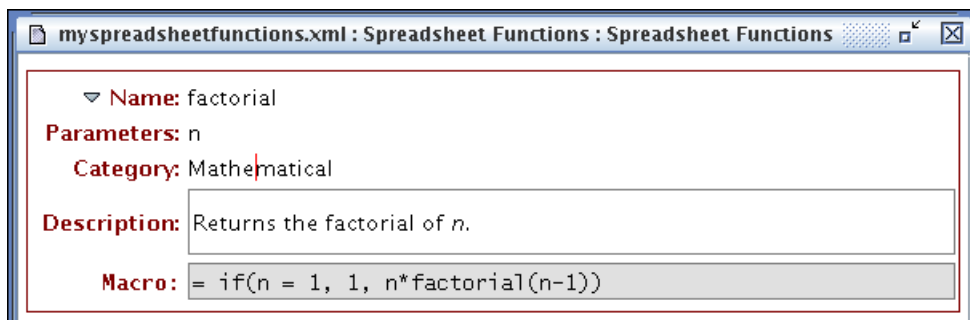
you'll be able to use custom function `factorial()` only when you open a DocBook document.

3.2. Specifying custom spreadsheet functions

The easiest way to create documents conforming to the `http://www.xmlmind.com/xmlmind/schema/spreadsheet/functions` schema, is to use XXE after installing the configuration found in `XXE_install_dir/doc/configure/functions_config/`.

In order to do this, recursively copy directory `XXE_install_dir/doc/configure/functions_config/` to `XXE_user_preferences_dir/addon/` and restart XXE. Then, use File|New and choose Spreadsheet Functions|Template.

Figure 1. Custom function `factorial()` edited in XXE using the Spreadsheet Functions configuration



The content model of the documents used to specify spreadsheet functions is:

```
<functions>
  Content: function+
</functions>

<function>
  Content: name parameters category description
         ( macro | method | runtime | intrinsic )
</function>

<name>
  Content: function name (any combination of letter, digit and _
         cannot start with digit or _)
</name>

<parameters>
```

```
Content: [ parameter name (any combination of letter, digit and _
        cannot start with digit) [ ?|*|+ ]? ]*
</parameters>

<category>
  Content:  non empty token
</category>

<description>
  Content:  an XHTML body (restrict yourself to
            HTML 3.2)
</description>

<macro
  xml:space = preserve
>
  Content: = definition of the function using the
            spreadsheet language
</macro>

<method>
  Content: a Java fully qualified method name
            (ASCII only)
</method>

<runtime
/>

<intrinsic
/>
```

name

Name of the custom spreadsheet function.

parameters

Specifies the name and the number of the formal parameters of the custom spreadsheet function.

Function `factorial` has a single, mandatory, formal parameter called `n`. Note that parameter `n` is referenced by its name in the specification of the macro-function (`=if(n=1,1,n*factorial(n-1))`).

The `parameters` element is as important as the "formula" of the custom spreadsheet function because it is used in many places. For example, it is used by the formula parser to check the number of arguments passed to functions (i.e. using `factorial(3, 4)` will cause the parse to report an error).

Examples:

- For standard spreadsheet function `and`: `boolean1 boolean2+`
- For standard spreadsheet function `numbervalue`: `text format? locale?`
- For standard spreadsheet function `sumif`: `nodeset test sum_nodeset?`
- For standard spreadsheet function `max`: `value+`
- For standard spreadsheet function `today`: `nothing at all: empty parameters element`.
- For standard spreadsheet function `if`: `test1 value1 alternative* fallback`

Without an *occurrence specifier*, a single argument must be passed for that parameter. Occurrence specifiers are:

?

0 or 1 argument corresponding to that parameter.

*

0 or more arguments corresponding to that parameter.

+

1 or more arguments corresponding to that parameter.

category

Category of the custom spreadsheet function.

You can use any of the predefined categories: Logical, Mathematical, Text, etc, or you can define your own categories.

This category is used by the Formula Editor.

description

Documentation in XHTML (restrict yourself to the HTML 3.2 subset) of the custom spreadsheet function.

This documentation is displayed by the Formula Editor.

macro

Specifies the custom spreadsheet function using the spreadsheet language (*macro-function*).

This specification must start with =.

This specification can reference the parameter names declared in the `parameters` sibling.

This specification can reference any other spreadsheet function, including itself (recursive macro-function).

method

Specifies the fully qualified name of the Java™ method used to implement the custom spreadsheet function.

The method name must be the name of the custom spreadsheet function, after converting it to lower case.

If the custom spreadsheet function has a name which is a reserved Java™ keyword (example: standard function `char`), the method name must be `'_'` (underscore) followed by the lower-case name of the custom spreadsheet function (example: `com.xmlmind.xmleditapp.spreadsheet.FunctionLibrary._char`).

More on this in next section.

runtime

Reserved to XMLmind. Do not use.

intrinsic

Reserved to XMLmind. Do not use.

3.3. Custom spreadsheet functions written in the Java™ programming language

Custom spreadsheet functions written in the Java™ programming language are implemented using static methods having this signature:

```
import com.xmlmind.xmledit.doc.XNode;
import com.xmlmind.xmledit.xpath.Variant;
import com.xmlmind.xmledit.xpath.VariantExpr;
import com.xmlmind.xmledit.xpath.EvalException;
import com.xmlmind.xmledit.xpath.ExprContext;

public static Variant method_name(VariantExpr[] args, XNode node,
                                  ExprContext context) throws EvalException;
```

There is not much to say about the above static methods. You'll need to read the chapter describing XPath programming⁴ in XMLmind XML Editor - Developer's Guide in order to be able to write such functions.

You'll find a template for spreadsheet functions in `XXE_install_dir/doc/dev/templates/FunctionLibraryTemplate.java`. You'll find a sample static method in `XXE_install_dir/doc/dev/samples/MySpreadsheetFunctions.java`. (Download developer's documentation and samples from www.xmlmind.com/xmleditor/download.shtml.)

Example:

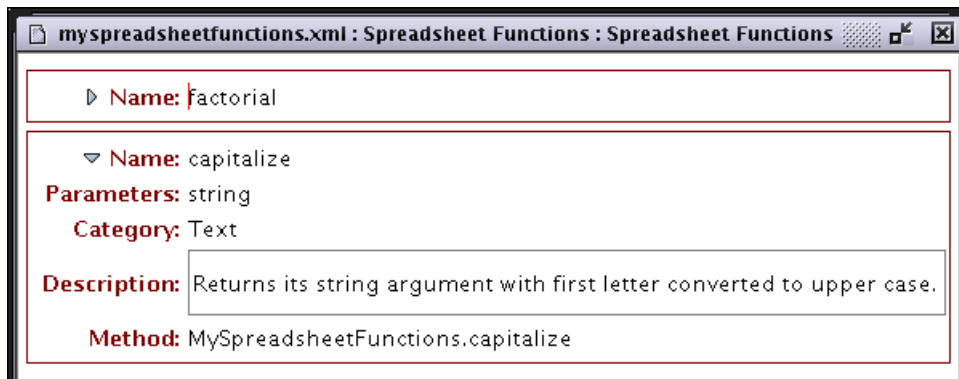
```
public final class MySpreadsheetFunctions {
    public static Variant capitalize(VariantExpr[] args, XNode node,
                                    ExprContext context)
        throws EvalException {
        if (args.length != 1)
            throw new EvalException("bad number of arguments");

        String string = args[0].eval(node, context).convertToString();
        int length = string.length();
        String transformed;

        if (length == 0)
            transformed = string;
        else if (length == 1)
            transformed = string.toUpperCase();
        else
            transformed = (Character.toUpperCase(string.charAt(0)) +
                           string.substring(1));

        return new StringVariant(transformed);
    }
}
```

This spreadsheet function needs to be declared in `myspreadsheetfunctions.xml` as follows:



The code of the `capitalize` spreadsheet function is found in `XXE_install_dir/doc/configure/samples2/myspreadsheetfunctions.jar`. Copy this jar file to one of the directories scanned by XXE at startup-time.

For example, add this to `XXE_user_preferences_dir/addon/customize.xxe`, after copying both `myspreadsheetfunctions.jar` and `myspreadsheetfunctions.xml` to `XXE_user_preferences_dir/addon/`.

```
<spreadsheetFunctions location="myspreadsheetfunctions.xml" />
```

⁴Remember that the spreadsheet language used by XMLmind XML Editor is basically an easy-to-learn syntax for XPath expressions.