XMLmind XML Editor - Using the Integrated Spreadsheet Engine

Hussein Shafie
Pixware

<xmleditor-support@xmlmind.com>

XMLmind XML Editor - Using the Integrated Spreadsheet Engine

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

Publication date October 28, 2010

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.

Part I. Tutorial

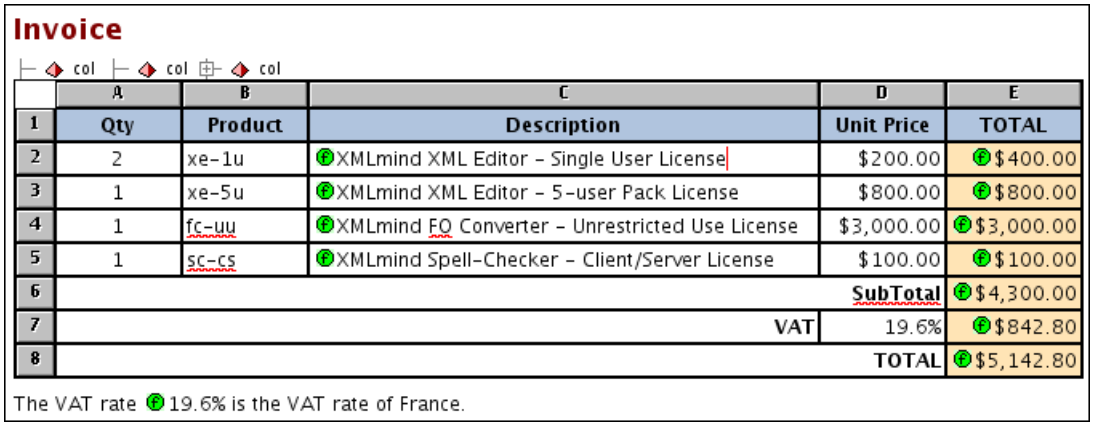
Chapter 1. Introduction

This feature is available only in XMLmind XML Editor Professional Edition, where it is hidden by default. You need to enable it by checking "Enable the Integrated Spreadsheet Engine" in Options → Preferences, Features section.

An easy to use and yet extremely powerful integrated spreadsheet engine is built into XMLmind XML Editor. This engine may be described as follows:

- It does *not* work by embedding an external spreadsheet component in the document. *The XML document is the spreadsheet*. That is, a formula can address any part of the XML document using XPath. (More information in Section 6, “XPath escapes” [18].)
- When an XML element is rendered on screen as a table, a formula can address table cells using the usual ``A1 notation" (example: `"table1"!$A$1:$C$3`). If the formula is itself inside a table cell, it can even use *relative cell references* (example: `A1:C3`).
- The formula language and the predefined functions are very similar to those found in other spreadsheet software (Microsoft Office Excel, OpenOffice.org Calc, etc). Example: `"sum is " & SUM(A1:A3)`. More than 80 predefined functions are provided.
- A formula is represented by processing instruction `<?xxe-formula>`. Using such specific processing instructions is allowed by the XML standard. `<?xxe-formula>`s are ignored by XML software other than XXE.

You'll find a demo you can play with in *`XXE_install_dir`*`/demo/spreadsheet-demo.xhtml`.



Invoice

col col col

	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	$4,300.00
7			VAT	19.6%	$842.80
8				TOTAL	$5,142.80

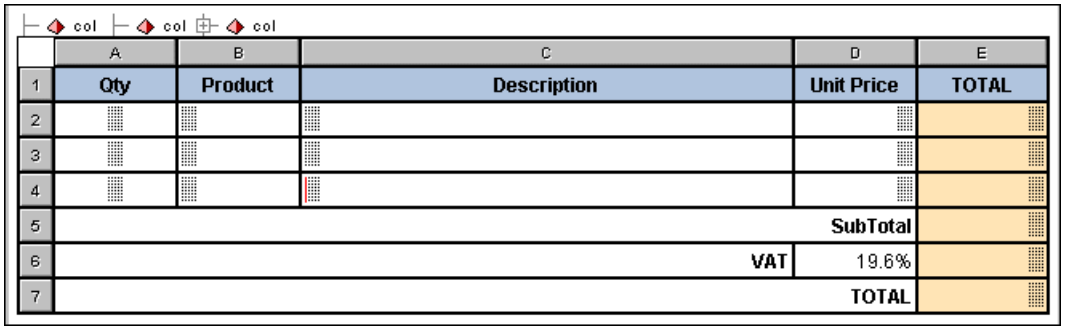
The VAT rate 19.6% is the VAT rate of France.

Chapter 2. Tutorial

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

1. Basics

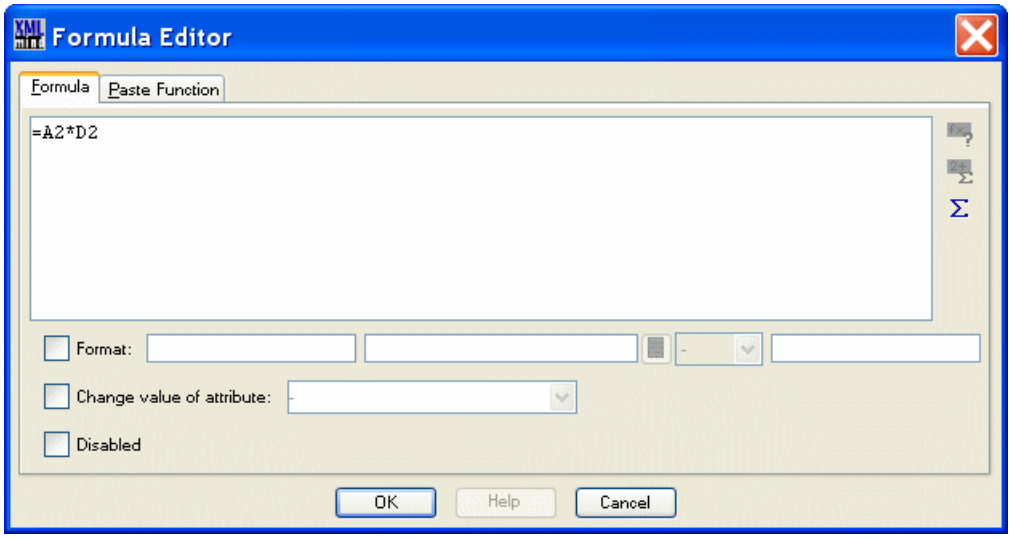
Use Tools → Spreadsheet → Show Table Labels to make the table look like a spreadsheet.



	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 (**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).

[1] In *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.

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

1. Click in cell E5 and insert formula: =sum(E2:E4)
2. 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.
3. 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

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 [18]). 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 [15]).

Double click
: Opens the Formula Editor to edit the formula.

Click with middle button
: Disables (gray icon) or enables (green icon) the formula.

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 =$A$2*$D$2.

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.

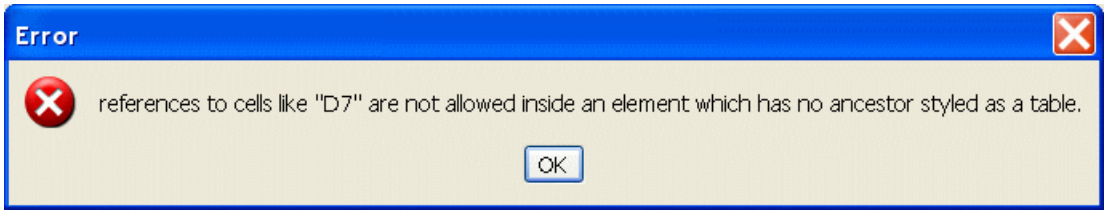
	A	B	C	D	E
1	**Qty**	**Product**	**Description**	**Unit Price**	**TOTAL**
2	2	xe-1u		200	400
3	1	xe-5u		800	800
4	1	fc-uu		3000	3000
5	1	sc-cs		100	100
6				**SubTotal**	4300
7			**VAT**	19.6%	842.8
8				**TOTAL**	5142.8

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**.

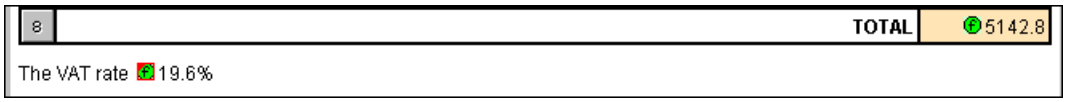
Add a paragraph containing sentence "`The VAT rate is the VAT rate of France.`". After the "`The VAT rate `", use **Ctrl**+**Shift**+**I** to insert a new formula.

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

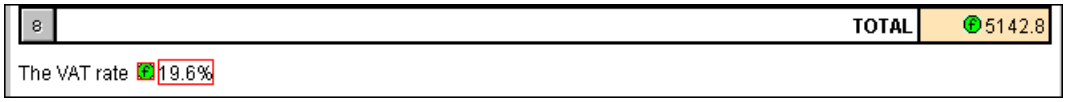


Outside a table, relative references are not allowed. Now specify ="invoice_table"!$D$7.

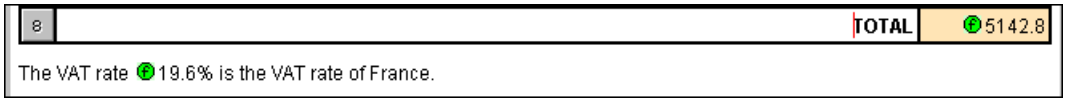
It works because we have used absolute references but this has deleted everything after "`The VAT rate `". This is normal. Remember that the value computed by a formula is added just after the `xxe-formula` processing instruction. If there is already some text just after the `xxe-formula` processing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



Click on the formula to select it. Use Select → Extend Selection to Following Sibling (**EscRight-Arrow**) and then Edit → Convert (**Ctrl+T**) to convert the two selected nodes to a `span`.



Now press **Insert** to insert a new text node after the newly created `span` and type once again "`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).

4.1. Better absolute references

Using ="invoice_table"!$D$7 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 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).

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 [18]. 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

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 work[2]. 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 numbervalue() must be passed a number format and optionally, a locale which specifies how to interpret this format.

First of all, set the `xml:lang` attribute of the `html` root element to `en`. By doing this, all the number formats found in spreadsheet formulas will by default use the English locale.

Then, click on the formula of cell E2 and replace =(A2 * D2) by =(A2 * numbervalue(D2, "$#,##0.00")).

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

The language of an XML element

Unless explicitly passed a locale argument[3], spreadsheet functions which have to interpret formats: numbervalue(), datetimevalue(), datevalue(), timevalue(), text(), etc, will use the language of the element containing the formula.

By default, this language is specified by the standard `xml:lang` attribute. This attribute is inherited, that is, the language of an element is also the language of all its descendant elements (of course, unless a descendant overrides this language using its own `xml:lang` attribute).

However, not all document types make use of the standard `xml:lang` attribute. For example, DocBook has a `lang` attribute. In such cases, how does XMLmind XML Editor determine the language of an element? The answer is: this must be specified in the configuration which parametrizes the behavior of XXE for a given document type.

The configuration element is called `spellCheckOptions`. Its `languageAttribute` and `defaultLanguage` attributes are used, not only by the integrated spell checker, but also by the integrated spreadsheet engine. DocBook example:

```
<cfg:spellCheckOptions xmlns=""
  useAutomaticSpellChecker="true"
  languageAttribute="lang"
  skippedElements="address funcsynopsisinfo classsynopsisinfo
                   literallayout programlisting screen synopsis" />
```

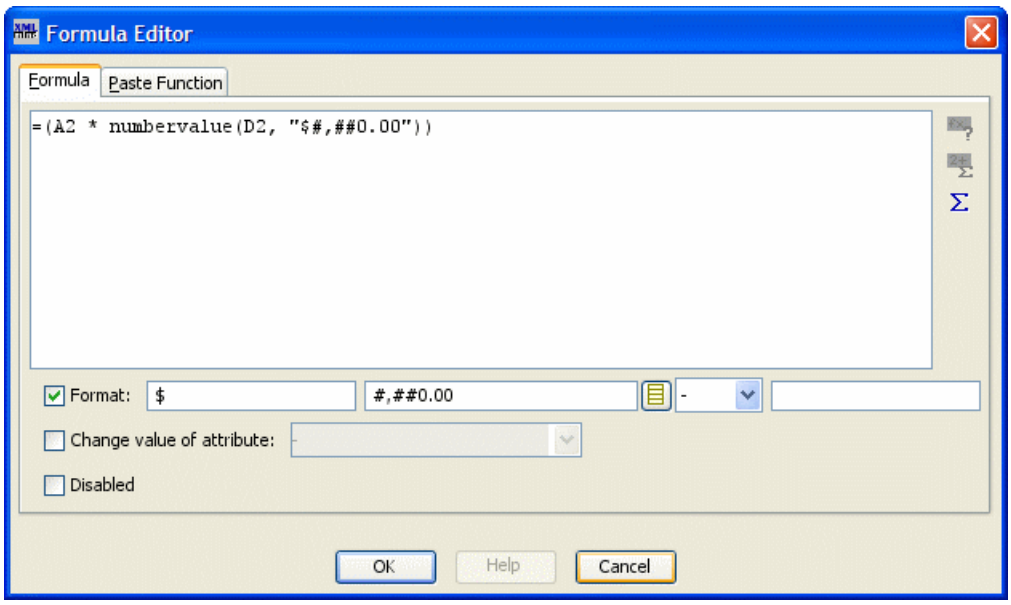
More information in Section 24, “spellCheckOptions” in *XMLmind XML Editor - Configuration and Deployment*.

When neither the standard `xml:lang` attribute nor the `spellCheckOptions` configuration element are used, the default language of XML elements is English ("`en`") and this, whatever the locale of the computer used to run 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.

[3] Example: numbervalue(D2, "$#,##0.00"**, "en-US"**).



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.

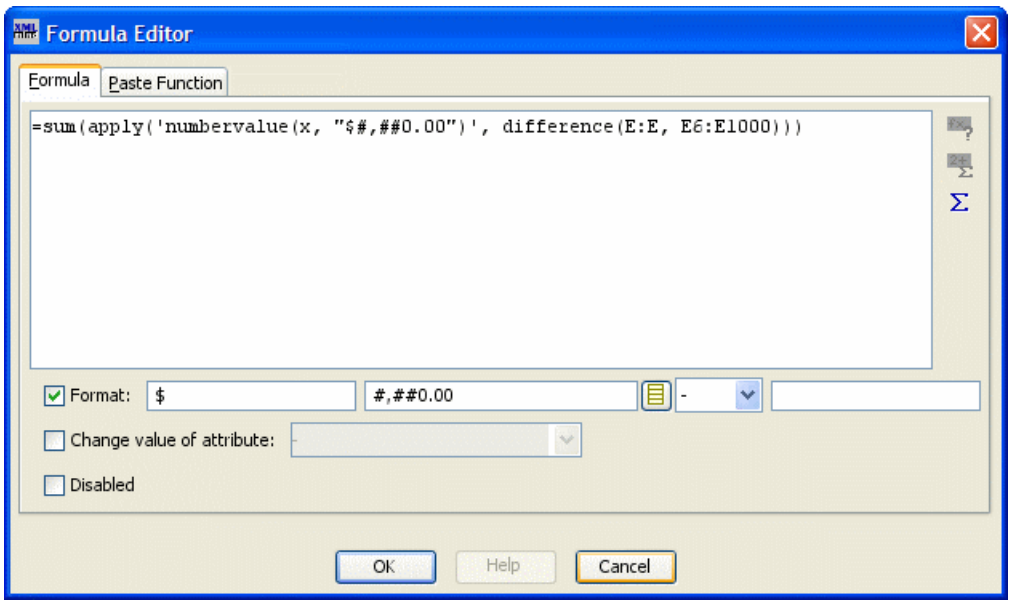
	A	B	C	D	E
1	**Qty**	**Product**	**Description**	**Unit Price**	**TOTAL**
2	2	xe-1u	XMLmind XML Editor - Single User License	$200.00	$400.00
3	1	xe-5u	XMLmind XML Editor - 5-user Pack License	$800.00	NaN
4	1	fc-uu	XMLmind FO Converter - Unrestricted Use License	$3,000.00	NaN
5	1	sc-cs	XMLmind Spell-Checker - 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.

	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().

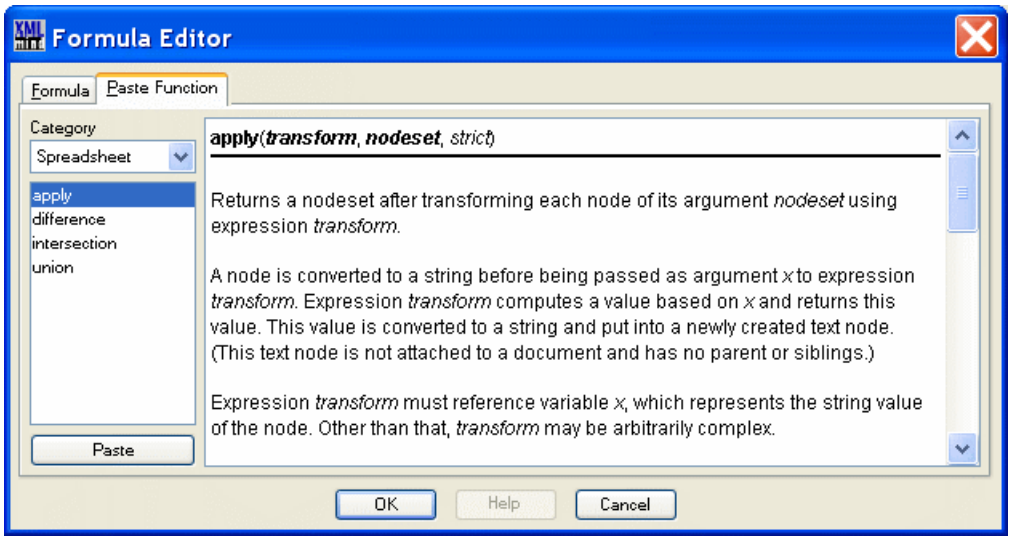
Function apply() applies a transformation to each node of a nodeset. Here the nodeset is: difference(E:E, E6:E1000). The transformation that we need to apply is numbervalue(*x*, "$#,##0.00"), where *x* represents the string value of the iterated node. This gives:



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

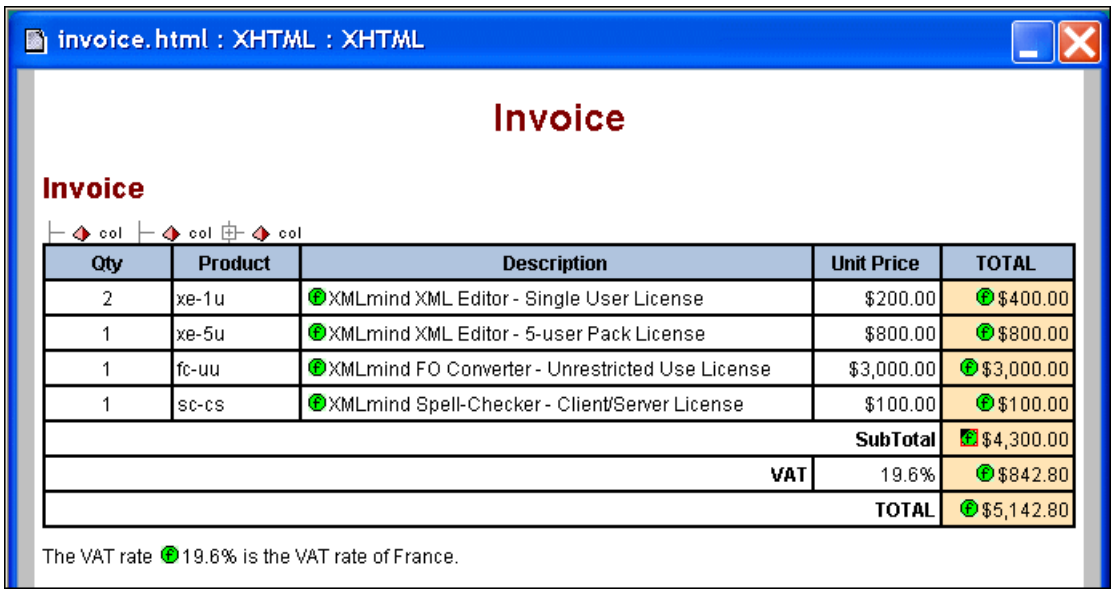
The documentation of spreadsheet functions is available online. If, for example, you don't remember how to use function apply(), simply select the word `apply` in the text area of the Formula Editor and press **F1**.



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



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%	$842.80
			TOTAL	$5,142.80

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

Part II. Reference

Chapter 3. The language used to write formulas

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

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. 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().[1]

Parentheses may be used to group subexpressions. Example:

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

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

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
```

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:

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

4. Literals

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.

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"?'
```

4.3. Booleans

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

5. References

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
```

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 a 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.

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.

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: $A$1 = $[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 $A$3: A1 = $[-2,0] 2:2 = $[-1,] B:B = $[,+1] 1:2 = $[-2,]:$[-1,] A:B = $[,0]:$[,+1] A1:B2 = $[-2,0]:$[-1,+1]

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"!$C$4, "table-23"!A:A (relative references are allowed too if the formula has itself an ancestor element formatted as a table).

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"

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, & expect 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:$3, "table-23"!$A$1, $(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").

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 superfluous zeros after the point (e.g. it generates 1, not 1.0). Scientific notation (e.g. 314E-2) is never used.	0 and NaN are converted to FALSE. Other numbers are converted to TRUE.	ERROR	A number is taken to be the number 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 314E-2 or 1,000,000.0 cannot be parsed as numbers (use function numbervalue() 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 Datatypes) are successfully converted to date, time and datetime. Examples: 1960-03-15Z (date), 13:30:00Z (time), 1960-03-16T12:30:00Z (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.	A nodeset is TRUE if it is non empty.	-	ERROR

From\To	Number	String	Boolean	Nodeset	Date / Time / DateTime
		Example: `<ul><li>The <b>little</a></li><li> <!--pussy-->cat </li><li>is chasing a mouse.</li></ul>` converted to a string gives "`The little cat is chasing a mouse.`".			
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 Datatypes). Examples: 1960-03-15Z (date), 13:30:00Z (time), 1960-03-16T12:30:00Z (datetime)	Date, time and datetime are always TRUE.	ERROR	-

Chapter 4. Predefined functions

1. Date and time functions

1.1. datevalue

```
datevalue(text, format?, locale?)
```

Converts *text* to a date. Unless *format* is specified, the date must be represented using the ISO 8601 format (also used by W3C XML Schema Datatypes).

Date format *format* may be used to parse text using a localized format. Supported formats are described in http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html. The empty string ("") specifies the default format.

Locale *locale* may be used to specify how to interpret format *format*. Without locale argument *locale*, this format is interpreted using the current language of the XML document (typically specified using the standard `xml:lang` attribute, but this can be configured).

Locales are specified using a standard 2-letter language code, optionally followed by a dash and a standard 2-letter country code. Examples: en, en-US, de, de-CH, etc.

Examples, (assume that the language of the XML document being edited is "`en-US`" and that XMLmind XML Editor is running on a machine in France=GMT+1):

- datevalue("1960-03-16Z") = March 16, 1960 UTC1960-03-16Z
- datevalue("03/16/60", "") = 1960-03-15Z
- datevalue("03/16/60 UTC", "MM/dd/yy z") = 1960-03-16Z
- datevalue("16/03/60", "", "fr") =1960-03-15Z
- datevalue("16 Mars 1960", "dd MMMM yyyy", "fr") = 1960-03-15Z

datevalue can also be used to convert a number to a date. In such case, *format* must not be specified. A date can be represented by the number of seconds since January 1, 1970, 00:00:00 GMT.

Example: datevalue(-309139200.000) = March 16, 1960 UTC

See also timevalue [22], datetimevalue [23], numbervalue [28], text [40].

1.2. timevalue

```
timevalue(text, format?, locale?)
```

Converts *text* to a time. Unless *format* is specified, the time must be represented using the ISO 8601 format (also used by W3C XML Schema Datatypes).

Date/time format *format* may be used to parse text using a localized format. Supported formats are described in http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html. The empty string ("") specifies the default format.

Locale *locale* may be used to specify how to interpret format *format*. Without locale argument *locale*, this format is interpreted using the current language of the XML document (typically specified using the standard `xml:lang` attribute, but this can be configured).

Locales are specified using a standard 2-letter language code, optionally followed by a dash and a standard 2-letter country code. Examples: en, en-US, de, de-CH, etc.

Examples, (assume that the language of the XML document being edited is "en-US" and that XMLmind XML Editor is running on a machine in France=GMT+1):

- timevalue("13:30:00Z") = 13:30:00Z
- timevalue("13:30:00") = 12:30:00Z
- timevalue("01:30 pm", "") = 12:30:00Z
- timevalue("01:30:00 pm UTC", "hh:mm:ss a z") = 13:30:00Z
- timevalue("13:30", "", "fr") = 12:30:00Z
- timevalue("13:30:00.123", "HH:mm:ss.SSS", "fr") = 12:30:00.123Z

timevalue can also be used to convert a number to a time. In such case, *format* must not be specified. A time can be represented by the number of seconds since 00:00:00 GMT.

Example: timevalue(13.5*3600) = T13:30:00Z

See also datevalue [22], datetimevalue [23], numbervalue [28], text [40].

1.3. datetimevalue

datetimevalue(*text*, *format*?, *locale*?)

Converts *text* to a date+time. Unless *format* is specified, the date+time must be represented using the ISO 8601 format (also used by W3C XML Schema Datatypes).

Date/time format *format* may be used to parse text using a localized format. Supported formats are described in http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html. The empty string ("") specifies the default format.

Locale *locale* may be used to specify how to interpret format *format*. Without locale argument *locale*, this format is interpreted using the current language of the XML document (typically specified using the standard xml:lang attribute, but this can be configured).

Locales are specified using a standard 2-letter language code, optionally followed by a dash and a standard 2-letter country code. Examples: en, en-US, de, de-CH, etc.

Examples, (assume that the language of the XML document being edited is "en-US" and that XMLmind XML Editor is running on a machine in France=GMT+1):

- datetimevalue("1960-03-16T13:30:00Z") = 1960-03-16T13:30:00Z
- datetimevalue("03/16/1960 01:30 pm", "") = 1960-03-16T12:30:00Z
- datetimevalue("03/16/60 01:30:00 pm UTC", "MM/dd/yy hh:mm:ss a z") = 1960-03-16T13:30:00Z
- datetimevalue("16/03/60 13:30", "", "fr") =1960-03-16T12:30:00Z
- datetimevalue("13:30:00, 16 Mars 1960", "HH:mm:ss, dd MM yyy", "fr") = 1960-03-16T12:30:00Z

datetimevalue can also be used to convert a number to a date+time. In such case, *format* must not be specified. A date+time can be represented by the number of seconds since January 1, 1970, 00:00:00 GMT.

Example: datetimevalue(-309139200 + 13.5*3600) = 1960-03-16T13:30:00Z

See also datevalue [22], timevalue [22], numbervalue [28], text [40].

1.4. date

```
date(year, month, day, time_zone?)
```

Returns a date representing specified *year*, *month*, *day*. Unless *time_zone* is specified, the local time zone is used.

Examples (XMLmind XML Editor running on a machine in France=GMT+1):

- date(1960, 03, 16) = 1960-03-15Z
- date(1960, 03, 16, "GMT+01:00") = 1960-03-15Z
- date(1960, 03, 16, "GMT") = 1960-03-16Z

Specifying time zones is explained in this document http://java.sun.com/j2se/1.4.2/docs/api/java/util/TimeZone.html.

See also datevalue [22], timevalue [22], datetimevalue [23], time [24], datetime [24].

1.5. time

```
time(hour, minute, second, time_zone?)
```

Returns a time representing specified *hour*, *minute*, *second*. Unless *time_zone* is specified, the local time zone is used.

Examples (XMLmind XML Editor running on a machine in France=GMT+1):

- time(13, 30, 45) = 12:30:45Z
- time(13, 30, 45, "GMT-01:00") = 14:30:45Z
- time(13, 30, 45.123, "GMT") = 13:30:45Z (note that seconds have been rounded)

Specifying time zones is explained in this document http://java.sun.com/j2se/1.4.2/docs/api/java/util/TimeZone.html.

See also datevalue [22], timevalue [22], datetimevalue [23], date [24], datetime [24].

1.6. datetime

```
datetime(year, month, day, hour, minute, second, time_zone?)
```

Returns a date+time representing specified *year*, *month*, *day*, *hour*, *minute*, *second*. Unless *time_zone* is specified, the local time zone is used.

Examples (XMLmind XML Editor running on a machine in France=GMT+1):

- datetime(1960, 03, 16, 13, 30, 45) = 1960-03-16T12:30:45Z
- datetime(1960, 03, 16, 13, 30, 45, "GMT-01:00") = 1960-03-16T14:30:45Z
- datetime(1960, 03, 16, 13, 30, 45, "GMT") = 1960-03-16T13:30:45Z

Specifying time zones is explained in this document http://java.sun.com/j2se/1.4.2/docs/api/java/util/TimeZone.html.

See also datevalue [22], timevalue [22], datetimevalue [23], date [24], time [24].

1.7. year

```
year(date)
```

Returns the year in specified *date*, a date or date+time. This function uses the local calendar (i.e. local time zone) to compute its result.

Examples (XMLmind XML Editor running on a machine in France=GMT+1):

- year(date(1960,03,16)) = 1960
- year(date(1960,03,16, "UTC")) = 1960
- year("1960-03-16T13:30:45+01:00") = 1960
- year("1960-12-31T23:00:00Z") = 1961

See also month [25], day [25], weekday [25], hour [26], minute [26], second [27].

1.8. month

```
month(date)
```

Returns the month in specified *date*, a date or date+time. This function uses the local calendar (i.e. local time zone) to compute its result.

Examples (XMLmind XML Editor running on a machine in France=GMT+1):

- month(date(1960,03,16)) = 3
- month(date(1960,03,16, "UTC")) = 3
- month("1960-03-16T13:30:45+01:00") = 3
- month("1960-03-31T23:00:00Z") = 4

See also year [24], day [25], weekday [25], hour [26], minute [26], second [27].

1.9. day

```
day(date)
```

Returns the day in specified *date*, a date or date+time. This function uses the local calendar (i.e. local time zone) to compute its result.

Examples (XMLmind XML Editor running on a machine in France=GMT+1):

- day(date(1960,03,16)) = 16
- day(date(1960,03,16, "UTC")) = 16
- day("1960-03-16T13:30:45+01:00") = 16
- day("1960-03-16T23:00:00Z") = 17

See also year [24], month [25], weekday [25], hour [26], minute [26], second [27].

1.10. weekday

```
weekday(date, option?)
```

Returns the day in specified *date*, a date or date+time. This function uses the local calendar (i.e. local time zone) to compute its result.

Option	Description
1 (default)	Returns 1 for Sunday to 7 for Saturday.
2	Returns 1 for Monday to 7 for Sunday

3	Returns 0 for Monday to 6 for Sunday

Examples (XMLmind XML Editor running on a machine in France=GMT+1):

- weekday(date(1960,03,16)) = 4 (Wednesday)
- weekday(date(1960,03,16, "UTC")) = 4
- weekday("1960-03-16T13:30:45+01:00") = 4
- weekday("1960-03-16T23:00:00Z") = 5 (Thursday)

With *option*=2:

- weekday(date(1960,03,16), 2) = 3 (Wednesday)
- weekday(date(1960,03,16, "UTC"), 2) = 3
- weekday("1960-03-16T13:30:45+01:00", 2) = 3
- weekday("1960-03-16T23:00:00Z", 2) = 4 (Thursday)

With *option*=3:

- weekday(date(1960,03,16), 3) = 2 (Wednesday)
- weekday(date(1960,03,16, "UTC"), 3) = 2
- weekday("1960-03-16T13:30:45+01:00", 3) = 2
- weekday("1960-03-16T23:00:00Z", 3) = 3 (Thursday)

See also year [24], month [25], day [25], hour [26], minute [26], second [27].

1.11. hour

```
hour(date)
```

Returns the hours in specified *date*, a time or date+time. This function uses the local calendar (i.e. local time zone) to compute its result.

Examples (XMLmind XML Editor running on a machine in France=GMT+1):

- hour(time(13,30,45)) = 13
- hour(time(13,30,45, "UTC")) = 14
- hour("1960-03-16T13:30:45+01:00") = 13
- hour("1960-03-16T13:30:45.123") = 13
- hour("1960-03-16T13:30:45.123Z") = 14

See also year [24], month [25], day [25], weekday [25], minute [26], second [27].

1.12. minute

```
minute(date)
```

Returns the minutes in specified *date*, a time or date+time. This function uses the local calendar (i.e. local time zone) to compute its result.

Examples (XMLmind XML Editor running on a machine in France=GMT+1):

- minute(time(13,30,45)) = 30
- minute("1960-03-16T13:30:45+01:00") = 30

See also year [24], month [25], day [25], weekday [25], hour [26], second [27].

1.13. second

```
second(date)
```

Returns the seconds in specified *date*, a time or date+time. This function uses the local calendar (i.e. local time zone) to compute its result.

Examples (XMLmind XML Editor running on a machine in France=GMT+1):

- second(time(13,30,45)) = 45
- second("1960-03-16T13:30:45.123") = 45 (note that seconds are rounded)

See also year [24], month [25], day [25], weekday [25], hour [26], minute [26].

1.14. today

```
today()
```

Returns current date. The time zone used is the local time zone.

Example (XMLmind XML Editor running on a machine in France=GMT+1): today() = 2004-10-15Z

1.15. now

```
now()
```

Returns current date+time. The time zone used is the local time zone.

Example (XMLmind XML Editor running on a machine in France=GMT+1): now() = 2004-10-16T11:12:14.155Z

2. Logical functions

2.1. and

```
and(boolean1, boolean2+)
```

Returns the logical AND of its arguments (after converting them to booleans, if needed to).

- A number is TRUE if and only if it is neither positive or negative zero nor NaN.
- A string is TRUE if and only if its length is non-zero.
- A date/time is always TRUE.
- An XML nodeset is TRUE if and only if it is non-empty.

See also or [27], not [28].

2.2. or

```
or(boolean1, boolean2+)
```

Returns the logical OR of its arguments (after converting them to booleans, if needed to).

Conversion of values to booleans is described here [27].

See also and [27], not [28].

2.3. not

```
not(boolean)
```

Returns TRUE if its argument is FALSE, and FALSE otherwise.

Conversion of values to booleans is described here [27].

See also and [27], or [27].

2.4. if

```
if(test1, value1, alternative*, fallback)
```

Alternative is: *testi valuei*.

Evaluates each *testi* in turn as a boolean, if the result of *testi* is TRUE, returns corresponding *valuei*. Otherwise, if all *testi* evaluates to FALSE, returns *fallback*.

Conversion of values to booleans is described here [27].

See also and [27], or [27], not [28].

2.5. true

```
true()
```

Returns TRUE. Note that it is also possible to write TRUE instead of TRUE().

2.6. false

```
false()
```

Returns FALSE. Note that it is also possible to write FALSE instead of FALSE().

3. Mathematical functions

3.1. numbervalue

```
numbervalue(text, format?, locale?)
```

Returns its first argument after converting it to a number.

Unless number format *format* is specified, values are converted to numbers as follows:

- A string that consists of optional whitespace followed by an optional minus sign followed by a real number followed by whitespace is converted to the IEEE 754 number that is nearest (according to the IEEE 754 round-to-nearest rule) to the mathematical value represented by the string; any other string is converted to NaN.

  Note that scientific notation (example: 0.314E1) is not supported.

- Boolean TRUE is converted to 1; boolean FALSE is converted to 0.

- A date/time is converted to the number of seconds since January 1, 1970, 00:00:00 GMT. This number can be negative and can have a fractional part.

- An XML nodeset is first converted to a string and then converted in the same way as a string argument.

  ### Note

  An XML nodeset is converted to a string by returning all the text contained in 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.

  Example: `<ul><li>The <b>little</a></li><li> <!--pussy-->cat </li><li>is chasing a mouse.</li></ul>` converted to a string gives "`The little cat is chasing a mouse.`".

The important thing to remember here is that unless a format is specified, *numbers cannot be specified using the localized notation*. For example: in France, write "`3.14`" to specify number PI and not "`3,14`".

In order to parse a localized number, number format *format* must be specified. Without locale argument *locale*, this format is interpreted using the current language of the XML document (typically specified using the standard `xml:lang` attribute, but this can be configured).

Examples, (assume that the language of the XML document being edited is "`en-US`"):

- numbervalue("3.14") = 3.14
- numbervalue("3.14", "#.#") = 3.14
- numbervalue("3.14", "") = 3.14 ("" is a shorthand notation for the default format)
- numbervalue("3,14", "#.#") = 3 (everything which is not a number after the number -- that is, the "," after the "3" -- is ignored)
- numbervalue("3,14", "#.#", "fr") = 3.14
- numbervalue("3,14", "", "fr-FR") = 3.14

Number formats are explained in the following document http://java.sun.com/j2se/1.4.2/docs/api/java/text/DecimalFormat.html.

Locales are specified using a standard 2-letter language code, optionally followed by a dash and a standard 2-letter country code. Examples: en, en-US, fr, fr-CA, etc.

3.2. checknumber

checknumber(*value*)

Returns TRUE if its argument can be successfully converted to a number (that is, which is not NaN). Otherwise returns FALSE.

Conversion of values to numbers is explained here [28].

See also numbervalue [28].

3.3. sum

sum(*value+*)

Returns the sum of all its arguments.

- If an argument is an XML nodeset (example: sum(A1:D4)), *each node in the nodeset* is converted to a number and this number is added to the total value. Nodes which cannot converted to numbers are simply ignored.
- If an argument is not a XML nodeset, it is converted to a number if needed to and then added to the total value. If the argument cannot be successfully converted to a number, an error is reported.

Conversion of values to numbers is explained here [28].

Example:

	A
1	10
2	20
3	Thirty

- sum(A:A, 30, "40.0", FALSE()) = 100.
- sum(3.1416, "Thirty") reports an error.

3.4. product

```
product(value+)
```

Same as sum [29] except that the product all the arguments is returned instead of the sum.

3.5. abs

```
abs(number)
```

Returns the absolute value of its argument (after converting it to a number [28], if needed to).

3.6. acos

```
acos(number)
```

Returns the arc cosine of its argument (after converting it to a number [28], if needed to). Number *number* must be in the 0,PI range.

3.7. asin

```
asin(number)
```

Returns the arc sine of its argument (after converting it to a number [28], if needed to). Number *number* must be in the -PI/2,PI/2 range.

3.8. atan

```
atan(number)
```

Returns the arc tangent of its argument (after converting it to a number [28], if needed to). Number *number* must be in the -PI/2,PI/2 range.

3.9. atan2

```
atan2(x, y)
```

Converts rectangular coordinates (*x*, *y*) to polar coordinates (*r*, *theta*). This function returns *theta* by computing an arc tangent of *y*/*x*. *y*/*x* must be in the -PI,PI range.

3.10. cos

```
cos(number)
```

Returns the cosine of its argument (after converting it to a number [28], if needed to).

3.11. cosh

```
cosh(number)
```

Returns the hyperbolic cosine of its argument (after converting it to a number [28], if needed to).

3.12. sin

```
sin(number)
```

Returns the sine of its argument (after converting it to a number [28], if needed to).

3.13. sinh

```
sinh(number)
```

Returns the hyperbolic sine of its argument (after converting it to a number [28], if needed to).

3.14. tan

```
tan(number)
```

Returns the tangent of its argument (after converting it to a number [28], if needed to).

3.15. tanh

```
tanh(number)
```

Returns the hyperbolic tangent of its argument (after converting it to a number [28], if needed to).

3.16. degrees

```
degrees(angle)
```

Returns its argument, an angle measured in radians, after converting it to degrees.

3.17. radians

```
radians(angle)
```

Returns its argument, an angle measured in degrees, after converting it to radians.

3.18. pi

```
pi()
```

Returns the value of PI, the ratio of the circumference of a circle to its diameter.

3.19. exp

```
exp(number)
```

Returns Euler's number **e** raised to the power of its argument (after converting it to a number [28], if needed to).

3.20. acosh

```
acosh(number)
```

Returns the inverse hyperbolic cosine of its argument (after converting it to a number [28], if needed to). Number *number* must be greater than 1.

3.21. asinh

```
asinh(number)
```

Returns the inverse hyperbolic sine of its argument (after converting it to a number [28], if needed to).

3.22. atanh

```
atanh(number)
```

Returns the inverse hyperbolic tangent of its argument (after converting it to a number [28], if needed to). Number *number* must be in the -1,1 range.

3.23. log

```
log(number, base)
```

Returns the log base *base* of its argument (after converting it to a number [28], if needed to). Number *number* must be strictly positive.

See also ln [32], log10 [32].

3.24. mod

```
mod(dividend, divisor)
```

Returns the remainder of the division of *dividend* by *divisor*. *Divisor* and *dividend* are converted to numbers [28] if needed to. Equivalent to: *dividend* - *divisor**INT [33](*dividend*/*divisor*).

Example: mod(3,2) = 1

3.25. ln

```
ln(number)
```

Returns the natural logarithm of its argument (after converting it to a number [28], if needed to). Number *number* must be strictly positive.

See also log [32], log10 [32].

3.26. log10

```
log10(number)
```

Returns the log base 10 of its argument (after converting it to a number [28], if needed to). Number *number* must be strictly positive.

See also log [32], ln [32].

3.27. sign

```
sign(number)
```

Returns 1 if its argument is strictly positive, -1 if its argument is strictly negative, 0 if its argument is null. The argument is converted to a number [28] if needed to.

3.28. sqrt

```
sqrt(number)
```

Returns the square root of its argument. Number *number* must be positive. The argument is converted to a number [28] if needed to.

3.29. trunc

```
trunc(number)
```

Returns its argument after removing its fractional part. The argument is converted to a number [28] if needed to.

Example: trunc(-8.9) = 8

See also int [33].

3.30. int

```
int(number)
```

Returns the largest value that is not greater than the argument and is equal to a mathematical integer. The argument is converted to a number [28] if needed to.

Example: int(-8.9) = 9

See also trunc [33].

3.31. rand

```
rand()
```

Returns a pseudo-random number between 0 and 1. Use rand()*(*b* - *a*) + *a* to get a random number in the *a*,*b* range.

3.32. countif

```
countif(nodeset, test)
```

Count each node in *nodeset* if evaluating boolean expression *test* returns TRUE for this node.

Boolean expression *test* must reference variable *x*, which represents the string value of the node. Other than that, *test* may be arbitrarily complex.

French
10
12
09
15
08

Examples (the above XHTML table has attribute id="exams1"):

- Count students having 12/20 or more to their French exam: countif("exams1"!$A:$A, "and(checknumber(x), x >= 12)") = 2
- Count students having between 9/20 and 12/20 to their French exam: countif("exams1"!$A:$A, "and(checknumber(x), x >= 9, x <= 12)") = 3

3.33. sumif

```
sumif(nodeset, test, sum_nodeset?)
```

For each node in *nodeset* which can be converted to a number, evaluates boolean expression *test*. If *test* returns TRUE adds node converted to a number to the total. Returns the total.

If *sum_nodeset* is specified, nodes in *nodeset* are used to evaluate *test* but it is the corresponding nodes in *sum_nodeset* which are added. Ignores nodes in *sum_nodeset* which cannot be converted to numbers.

Boolean expression *test* must reference variable *x*, which represents the string value of the node. Other than that, *test* may be arbitrarily complex.

Investment	ROI
10000	10000
12000	15000
9000	17000
15000	-20000
8000	-1000

Examples (the above XHTML table has attribute id="roi1"):

- Compute the sum of all investments larger than EUR10000: sumif("roi1"!$A:$A, "x >= 10000") = 37000
- Compute the return on investment for all investments larger than EUR10000: sumif("roi1"!$A:$A, "x >= 10000", "roi1"!$B:$B) = 5000

3.34. round

```
round(number, digits)
```

Returns number *number* rounded to the specified number of digits *digits*.

Examples:

- round(33.14159, 0) = 33
- round(33.74159, 0) = 34
- round(33.14159, 2) = 33.14
- round(33.14159, -1) = 30
- round(-33.14159, 0) = -33
- round(-33.14159, 2) = -33.14
- round(-33.14159, -1) = -30

See also rounddown [34], roundup [35].

3.35. rounddown

```
rounddown(number, digits)
```

Returns number *number* rounded down to the specified number of digits *digits*.

Examples:

- rounddown(33.14159, 0) = 33
- rounddown(33.74159, 0) = 33

- rounddown(33.14159, 2) = 33.14
- rounddown(33.14159, -1) = 30
- rounddown(-33.14159, 0) = -33
- rounddown(-33.14159, 2) = -33.14
- rounddown(-33.14159, -1) = -30

See also round [34], roundup [35].

3.36. roundup

```
roundup(number, digits)
```

Returns number *number* rounded up to the specified number of digits *digits*.

Examples:

- roundup(33.14159, 0) = 34
- roundup(33.74159, 0) = 34
- roundup(33.14159, 2) = 33.15
- roundup(33.14159, -1) = 40
- roundup(-33.14159, 0) = -34
- roundup(-33.14159, 2) = -33.15
- roundup(-33.14159, -1) = -40

See also round [34], rounddown [34].

4. Spreadsheet functions

4.1. union

```
union(nodeset1, nodeset2+)
```

Returns the union of all its nodeset arguments.

References such as A1, A;A, $B$2:$C$3,etc, all return nodesets, with one XML node per table cell. For example: union(A1,A2,A3) is equivalent to A1:A3.

See also intersection [35], difference [36].

4.2. intersection

```
intersection(nodeset1, nodeset2)
```

Returns all XML nodes found in both *nodeset1* and *nodeset2*.

References such as A1, A;A, $B$2:$C$3,etc, all return nodesets, with one XML node per table cell. For example: intersection(A:A,2:2) is equivalent to A2.

See also union [35], difference [36].

4.3. difference

```
difference(nodeset1, nodeset2)
```

Returns all XML nodes found in *nodeset1* but not in *nodeset2*.

References such as A1, A;A, $B$2:$C$3,etc, all return nodesets, with one XML node per table cell. For example: difference($A:$A,$A$10) returns all the cells of column $A except cell $A$10.

See also union [35], intersection [35].

4.4. apply

```
apply(transform, nodeset, strict?)
```

Returns a nodeset after transforming each node of its argument *nodeset* using expression *transform*.

A node is converted to a string before being passed as argument *x* to expression *transform*. Expression *transform* computes a value based on *x* and returns this value. This value is converted to a string and put into a newly created text node. (This text node is not attached to a document and has no parent or siblings.)

Expression *transform* must reference variable *x*, which represents the string value of the node. Other than that, *transform* may be arbitrarily complex.

By default apply is lenient. That is, if *transform* fails to be evaluated, the raw string value of the node is silently used. Specify *strict* as TRUE, if you want **apply** to report an error when *transform* fails to be evaluated.

Investment
10,000.00
12,000.00
9,000.00
15,000.00
8,000.00

Examples (the above XHTML table has attribute id="roi2"):

- Compute the sum of all investments larger than EUR10000: sumif("roi2"!$A$2:$A$6, "x >= 10000") = 0

  This does not work because investments such as 10,000.00 *use a localized format* which cannot be parsed. That is, "10000.00" can be parsed as a number. "10,000.00" cannot.

- Compute the sum of all investments larger than EUR10000: sumif(apply("numbervalue(x, '', 'en-US')", "roi2"!$A$2:$A$6), "x >= 10000") = 37000

  Note that in transform "numbervalue(x, '', 'en-US')", we have used *single quotes* to quote the default format '' and the 'en-US' locale. It could have been possible to specify the same transform as "numbervalue(x, """", ""en-US"")" but this is much less readable.

- sumif(apply("numbervalue(x, '', 'en-US')", "roi2"!$A:$A, TRUE), "x >= 10000") = ERROR

- sumif(apply("numbervalue(x, '', 'en-US')", "roi2"!$A:$A, FALSE), "x >= 10000") = 37000

5. Statistical functions

5.1. avedev

```
avedev(value+)
```

Returns the average of the absolute deviation of a sample from the mean. Formula is: (Σ|x - average|)/n.

- If an argument is an XML nodeset, *each node in the nodeset* is converted to a number and processed. Nodes which cannot converted to numbers are simply ignored.
- If an argument is not a XML nodeset and cannot be successfully converted to a number, an error is reported.

French
10
12
09
15
08

Examples (the above XHTML table has attribute id="exams2"):

- avedev("exams2"!$A:$A) = 2.16
- avedev("exams2"!$A:$A, "French") = ERROR (String "French" is ignored in column $A but not when passed directly as an argument)

See also stdev [37], var [37].

5.2. stdev

stdev(*value+*)

Returns the standard deviation of its arguments. Formula is: √((Σ(x - average)²)/(n - 1)).

- If an argument is an XML nodeset, *each node in the nodeset* is converted to a number and processed. Nodes which cannot converted to numbers are simply ignored.
- If an argument is not a XML nodeset and cannot be successfully converted to a number, an error is reported.

French
10
12
09
15
08

Examples (the above XHTML table has attribute id="exams3"):

- stdev("exams3"!$A:$A) = 2.774
- stdev("exams3"!$A:$A, "French") = ERROR (String "French" is ignored in column $A but not when passed directly as an argument)

See also avedev [36], var [37].

5.3. var

var(*value+*)

Returns the variance of its arguments. Formula is: (Σ(x - average)²)/(n - 1).

- If an argument is an XML nodeset, *each node in the nodeset* is converted to a number and processed. Nodes which cannot converted to numbers are simply ignored.
- If an argument is not a XML nodeset and cannot be successfully converted to a number, an error is reported.

French
10
12
09
15
08

Examples (the above XHTML table has attribute id="exams4"):

- var("exams4"!$A:$A) = 7.69999999
- var("exams4"!$A:$A, "French") = ERROR (String "French" is ignored in column $A but not when passed directly as an argument)

See also avedev [36], stdev [37].

5.4. max

max(*value+*)

Returns the maximum of its arguments. Returns 0 if all its arguments cannot be converted to numbers.

- If an argument is an XML nodeset, *each node in the nodeset* is converted to a number and processed. Nodes which cannot converted to numbers are simply ignored.
- If an argument is not a XML nodeset and cannot be successfully converted to a number, an error is reported.

French
10
12
09
15
08

Examples (the above XHTML table has attribute id="exams5"):

- max("exams5"!$A:$A) = 15
- max("exams5"!$A:$A, "French") = ERROR (String "French" is ignored in column $A but not when passed directly as an argument)

5.5. min

min(*value+*)

Returns the minimum of its arguments. Returns 0 if all its arguments cannot be converted to numbers.

- If an argument is an XML nodeset, *each node in the nodeset* is converted to a number and processed. Nodes which cannot converted to numbers are simply ignored.
- If an argument is not a XML nodeset and cannot be successfully converted to a number, an error is reported.

French
10
12
09
15
08

Examples (the above XHTML table has attribute id="exams6"):

- min("exams6"!$A:$A) = 8
- min("exams6"!$A$1:$A$1) = 0
- min("exams6"!$A:$A, "French") = ERROR (String "French" is ignored in column $A but not when passed directly as an argument)

5.6. average

```
average(value+)
```

Returns the arithmetic mean of its arguments.

- If an argument is an XML nodeset, *each node in the nodeset* is converted to a number and processed. Nodes which cannot converted to numbers are simply ignored.
- If an argument is not a XML nodeset and cannot be successfully converted to a number, an error is reported.

French
10
12
09
15
08

Examples (the above XHTML table has attribute id="exams7"):

- average("exams7"!$A:$A) = 10.8
- average("exams7"!$A:$A, "French") = ERROR (String "French" is ignored in column $A but not when passed directly as an argument)

5.7. count

```
count(value+)
```

Counts its arguments which can be converted to numbers.

- If an argument is an XML nodeset, *each node in the nodeset* is converted to a number and processed. Nodes which cannot converted to numbers are simply ignored.
- If an argument is not a XML nodeset and cannot be successfully converted to a number, an error is reported.

French
10
12

09
15
08

Examples (the above XHTML table has attribute id="exams8"):

- count("exams8"!$A:$A, 0, 20) = 7
- count("exams8"!$A:$A, "French") = ERROR (String "French" is ignored in column $A but not when passed directly as an argument)

6. Text functions

6.1. char

```
char(code)
```

Returns a string containing a single character, this character having specified Unicode code. Reports an error if argument *code* cannot be converted to a number in the 0,65535 range.

Example: char(65) = "A"

See also code [40].

6.2. code

```
code(string)
```

Returns the Unicode code of the first character contained in specified string. Reports an error if argument *string* cannot be converted to a non-empty string.

Example: code("ABC") = 65

See also char [40].

6.3. text

```
text(number_or_date, format?, locale?)
```

Converts its number or date/time argument to a string.

If the argument is a number and number format *format* is not specified, a standard, non-localized, format is used.

If the argument is a date/time and date/time format *format* is not specified, a standard, non-localized, format is used.

If a *format* is specified, argument *locale* may be used to specify the locale of this format. Without locale argument *locale*, this format is interpreted using the current language of the XML document (typically specified using the standard `xml:lang` attribute, but this can be configured).

Examples, (assume that the language of the XML document being edited is "`en-US`"):

- text(PI()) = 3.141592653589793
- text(PI(), "0.00") = 3.14
- text(PI(), "") = 3.142 ("" is a shorthand notation for the default format)
- text(PI(), "0.00", "fr") = 3,14

- text(PI(), "", "fr") = 3,142
- text(today()) = 2004-10-12Z
- text(today(), "MMMM dd, yyyy") = October 13, 2004
- text(today(), "") = 10/13/04 ("" is a shorthand notation for the default format)
- text(today(), "MMMM dd, yyyy", "fr") = octobre 13, 2004
- text(today(), "", "fr") = 13/10/04
- text(now()) = 2004-10-13T15:51:59.321Z (today() returns a date. now() returns a date+time)
- text(now(), "") = 10/13/04 5:53 PM
- text(now(), "dd/MM/yyyy HH:mm:ss", "fr-CA") = 13/10/2004 17:53:18

Number formats are explained in the following document http://java.sun.com/j2se/1.4.2/docs/api/java/text/DecimalFormat.html.

Date/time formats are explained in the following document http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html.

Locales are specified using a standard 2-letter language code, optionally followed by a dash and a standard 2-letter country code. Examples: en, en-US, de, de-CH, etc.

6.4. mid

```
mid(text, from, count)
```

Returns the substring of *text* containing *count* characters, starting at character #*from*. First character is character #1.

Example: mid("very long", 5, 4) = "long"

See also left [41], right [41].

6.5. left

```
left(text, count)
```

Returns the substring of *text* containing the first *count* characters.

Example: left("very long", 4) = "very"

See also mid [41], right [41].

6.6. right

```
right(text, count)
```

Returns the substring of *text* containing the last *count* characters.

Example: right("very long", 4) = "long"

See also left [41], mid [41].

6.7. trim

```
trim(text)
```

Removes leading and trailing whitespace from *text*.

6.8. lower

```
lower(text)
```

Returns *text* converted to lower case.

See also upper [42].

6.9. upper

```
upper(text)
```

Returns *text* converted to upper case.

See also lower [42].

6.10. len

```
len(text)
```

Returns the number of characters contained in *text*.

6.11. substitute

```
substitute(text, old_text, new_text, occurrence?)
```

Returns *text* after substituting in it occurrences of substring *old_text* by *new_text*. If *occurrence* is specified, only occurrence *#occurrence* is substituted (first occurrence is occurrence #1). Otherwise all occurrences of *old_text* are substituted.

Examples:

- substitute("Element wordasword", "word", "term") = "Element termasterm"
- substitute("Element wordasword", "word", "term", 2) = "Element wordasterm"
- substitute("Element wordasword", "word", "not found") = "Element wordasword"

See also replace [42], find [42], search [43].

6.12. replace

```
replace(text, from, count, new_text)
```

Returns *text* after replacing substring containing *count* characters and starting at character *#from* by *new_text*. First character of a string is character #1.

Example: replace("one two three", 4, 3, "2") = "one 2 three"

See also substitute [42], find [42], search [43].

6.13. find

```
find(what, text, from?)
```

Returns the index of first occurrence of *what* found in *text*. Unless *from* is specified, search is started at character 1. (First character of a string is character #1.)

Reports an error if *what* is not found.

Unlike with search [43], *what* cannot contain wildcards and the lookup is case-sensitive.

Examples:

- find("A", "A rainy day.") = 1
- find("a", "A rainy day.") = 4
- find("A", "A rainy day.", 2) = ERROR
- find("a", "A rainy day.", 2) = 4

6.14. search

```
search(what, text, from?)
```

Returns the index of first occurrence of *what* found in *text*. Unless *from* is specified, search is started at character 1. (First character of a string is character #1.)

Reports an error if *what* is not found.

Unlike with find [42], *what* can contain wildcards and the lookup is case-insensitive.

- Wildcard "*" matches any number of characters.
- Wildcard "?" matches a single character.
- Use "~?" and "~*" if you search a substring containing characters "?" and "*".

Examples:

- search("a", "A rainy day.") = 1
- search("a", "A rainy day.", 2) = 4
- search("It's", "A rainy day.") = ERROR
- search("a*y", "A rainy day.") = 4
- search("~*-tuple", "1-tuple 2-tuple *-tuple") = 17

Chapter 5. Defining custom spreadsheet functions

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/spreadsheet/custom_functions/`)

```
<?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 `spreadsheetFunctions`. See Section 25, “spreadsheetFunctions” in *XMLmind XML Editor - Configuration and Deployment*.

- 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:
  >
  > - *`$HOME`*`/.xxe4/` on Linux.
  > - *`$HOME`*`/Library/Application Support/XMLmind/XMLEditor4/` on the Mac.
  > - *`%APPDATA%`*`\XMLmind\XMLEditor4\` on Windows 2000, XP, Vista.
  >
  >   Example: `C:\Documents and Settings\john\Application Data\XMLmind\XMLEditor4\` on Windows 2000 and XP. `C:\Users\john\AppData\Roaming\XMLmind\XMLEditor4\` on Windows Vista.

- 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" />
  ```

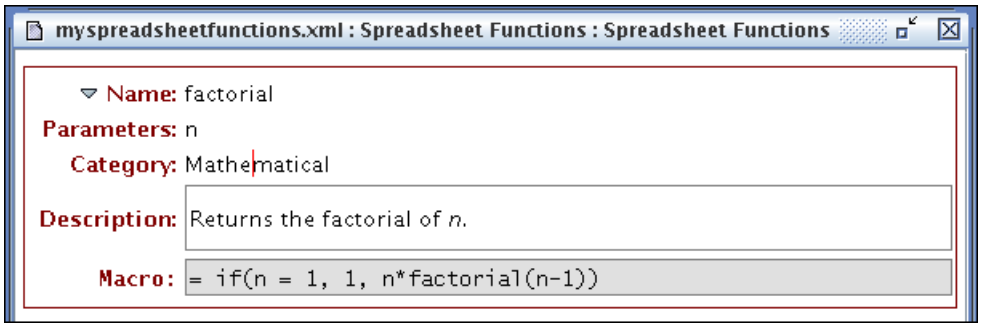
you'll be able to use custom function `factorial()` only when you open a DocBook document.

2. Specifying custom spreadsheet functions

The easiest way to create documents conforming to the `http://www.xmlmind.com/xmleditor/schema/spreadsheet/functions` schema, is to download and install the corresponding configuration. In order to do so, please use Options → Install Add-ons, select the add-on called "*XMLmind XML Editor Configuration Pack*" from the list and click OK.

After doing this, restart XXE, use File → New and choose XMLmind XML Editor Spreadsheet Functions/List of functions.

Figure 5.1. 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. 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 programming[1] in Chapter 5, *Using XPath* 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();
```

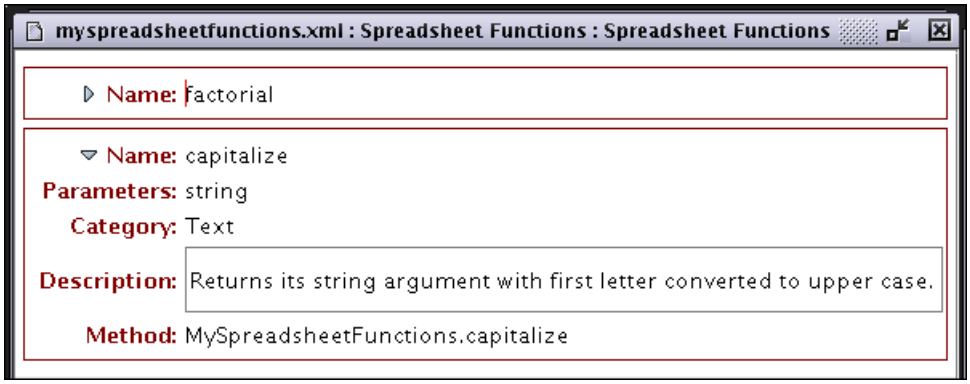
[1] Remember that the spreadsheet language used by XMLmind XML Editor is basically an easy-to-learn syntax for XPath expressions.

```
        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/spreadsheet/custom_functions/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" />
```