Table of Contents

1 Introduction

XiMpLe is a simple editor for XML and JSON files based on a grid table view. It can be used also for splitting large xml files and joining them back together or as a simple text editor. From version 1.3 it supports comparing of xml files displayed in a grid view and a comparison of text files.

1.1 Features

- No installation - one small file ready to use
- An easy manipulation with tables containing simple texts (emphasis on transfers between XiMpLe and a spreadsheet e.g. Excel) [see the section 11. `Copy`]
- Sub-tables for mining deeply nested data [see the section 7. `Sub-tables`]
- A possibility to configure properties affecting the result file [see the section 21.2 `Data and Table...`]
- A possibility to configure appearance of the editor according to file extension [21.2]
- Easy walking through a document using a keyboard [see the section 27. `Shortcuts`]
- Wild cards searching and replacing [see the section 20.3 `Find and Replace`]
- Replacing using a lookup table [see the section 20.3.2]
- Find and modify functionality [see the section 20.4 `Find and Modify`]
- Transparent clipboard operations [see the section 4. `Clipboard operations`]
- XML splitter and XML joiner [see the sections 23.2 and 23.3]
- Light and quite fast text editor [section 22. `Text Editor`]
- Comparison (and merging) of text files and xml files [section 23.1 `Compare Mode`]
- Support for JSON preserving white spaces [section 18. `JSON files`]
- Validation by XSD schemas [section 23.4 `Validation Mode`]

1.2 Requirements

XiMpLe is running on Windows 7 or later and .Net Framework 4.5 is required. Up to version 1.5 the editor works also in Windows XP with .Net framework 3.5 or 4.0.

2 XML structure - basics

For the purpose of this text we will explain an XML data structure in a very simplified way without unnecessary details.

The "brick" of the xml structure is an "element". Element has its name and can have a value.
An element can contain one or more other elements. An element can have one or more attributes.

An attribute has its name and must have some value (it can be an empty string as well). The attribute can contain nothing more than its value. An attribute is something like a label of an element.

A simple example follows – there are 3 elements and the first element has two attributes; the first and the third element don't have a value but the first element contains two elements):

```
<Element1 attribute1="attribute value1" attribute2="attribute value 2">
        <Element2>value of element 2</Element2>
        <EmptyElement3></EmptyElement3>
</Element1>
```

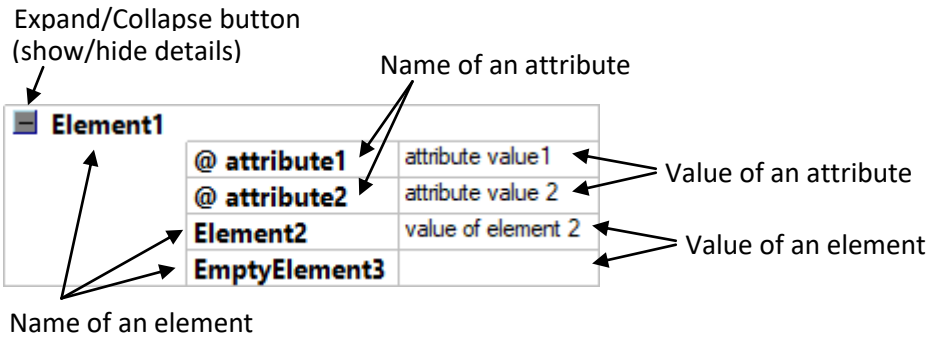
Example 1

There are some other ingredients of a data structure like comments, processing elements etc. which we can ignore now. On the top level of a structure there is always only one element which is called the root element.

An XML structure can be viewed by any text editor but usually some xml editor is used to display and manipulate data more comfortably.

In XML data there can frequently exists elements with the same name on the same structure level. These repeating elements and structures can be collected and displayed in tables which can make the whole structure easier for a reading.

A mentioned *Example 1* can look like this in XiMpLe editor.

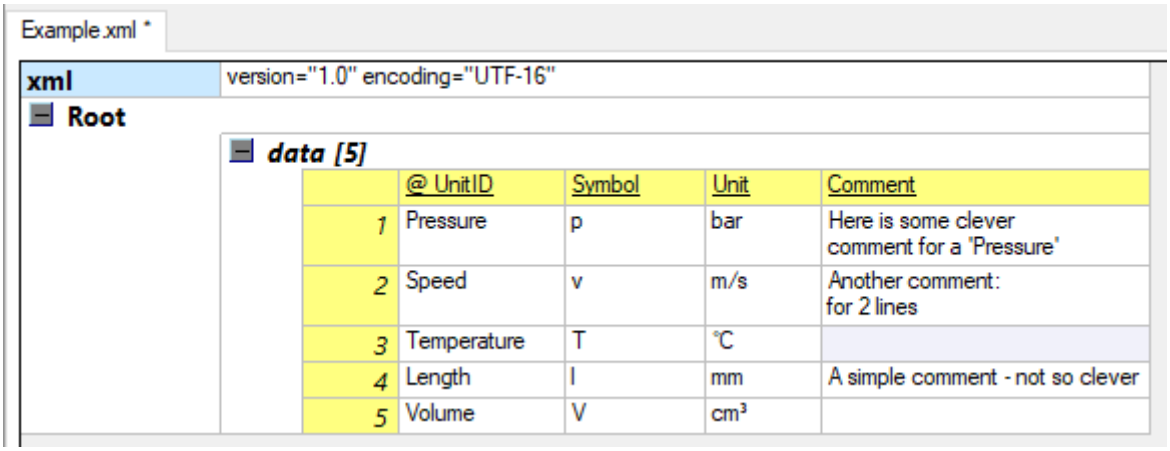


3 Quick overview – some frequently used tasks

This section is arranged especially for experienced, curious or impatient users to demonstrate quickly in pictures some tasks which can be done in XiMpLe with a minimal effort.

3.1 Copy a simple table to Excel

Let's have some table with no nested elements inside (we will call such table a simple table). It can look like this:


Example.xml *

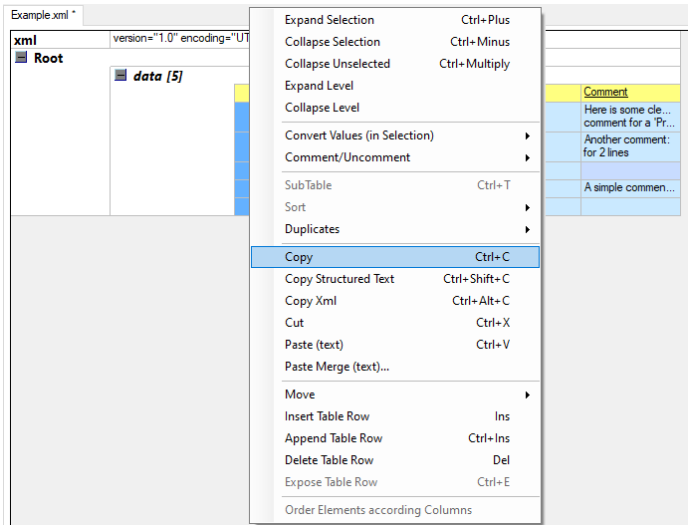
xml version="1.0" encoding="UTF-16"

Root

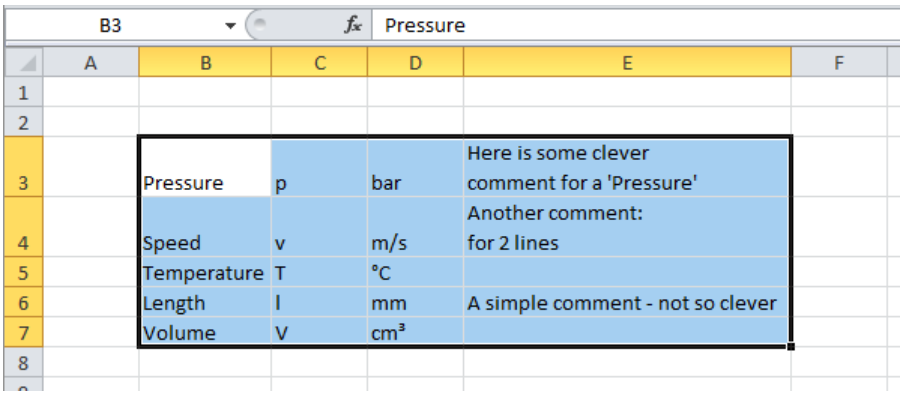
data [5]

	@ UnitID	Symbol	Unit	Comment
1	Pressure	p	bar	Here is some clever comment for a 'Pressure'
2	Speed	v	m/s	Another comment: for 2 lines
3	Temperature	T	°C	
4	Length	l	mm	A simple comment - not so clever
5	Volume	V	cm³	

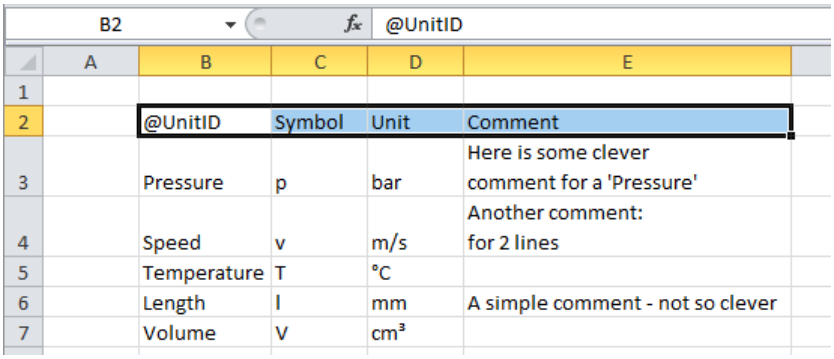
We can select the whole table and copy it to clipboard…



…and then paste it in Excel.

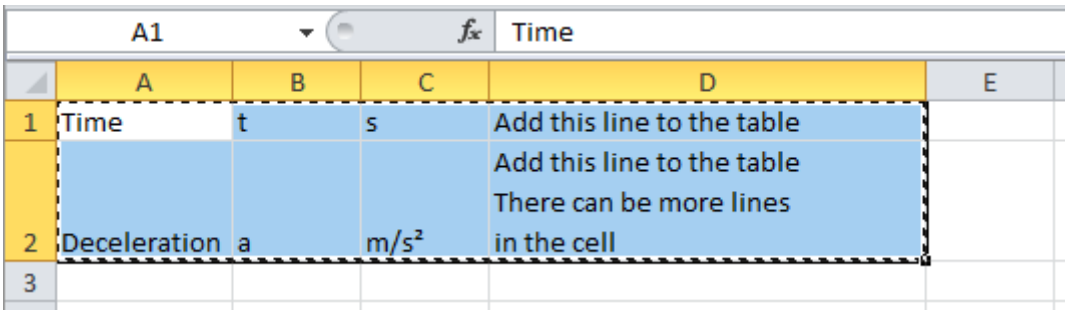


If we need we can additionally add the column head names as well (there is a command Copy Column Headings which copy column names for all selected columns):

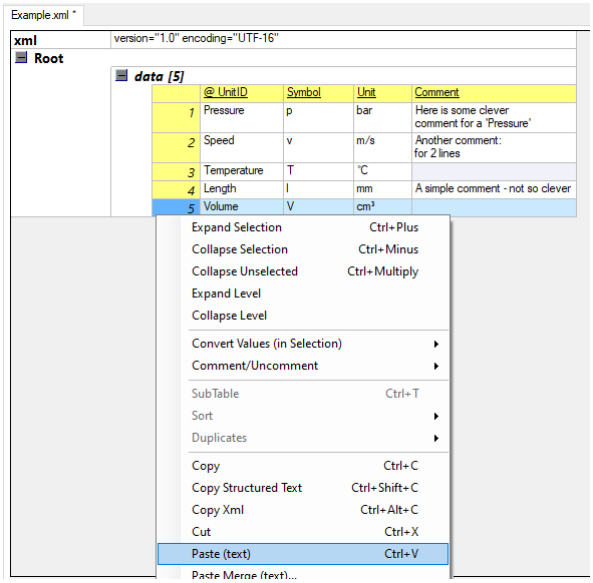


3.2 Add some new rows prepared in Excel to the XML table

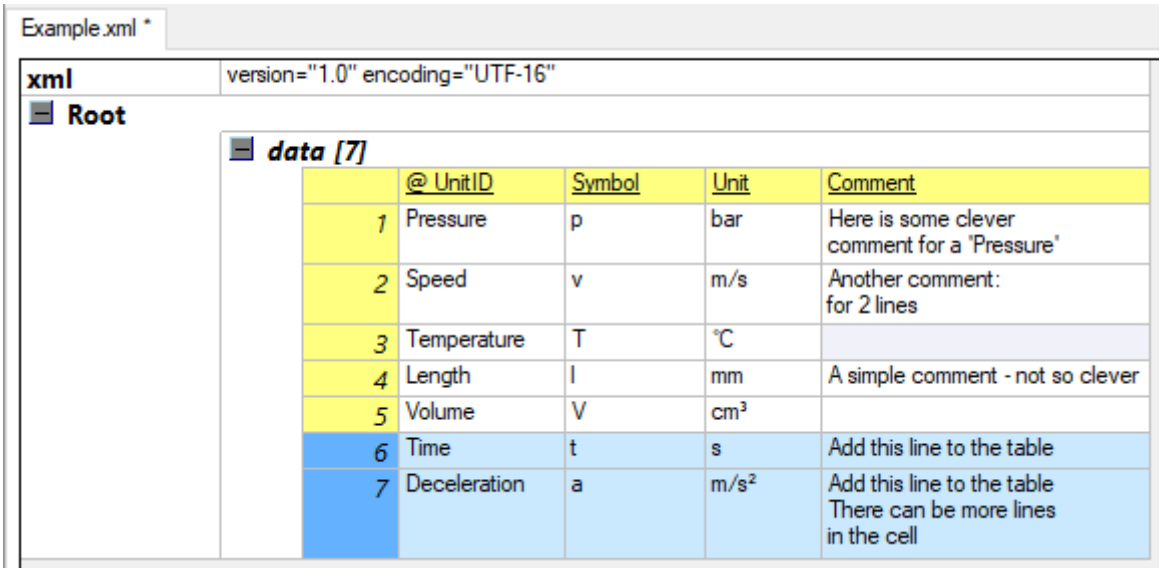
Another common task is to transfer some data from Excel to xml table. Let's have some prepared rows in Excel which we want to add to an XML table.



In the XiMpLe we just select the last table row and paste it.

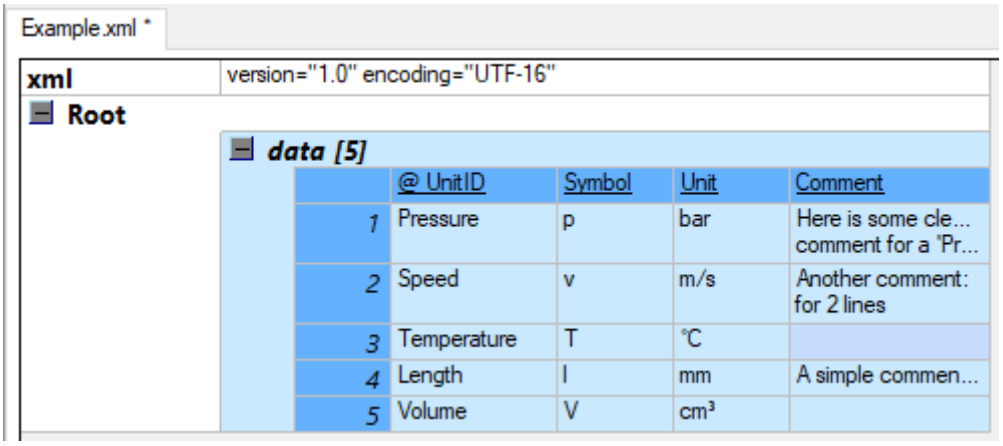


At the end it should look like this:

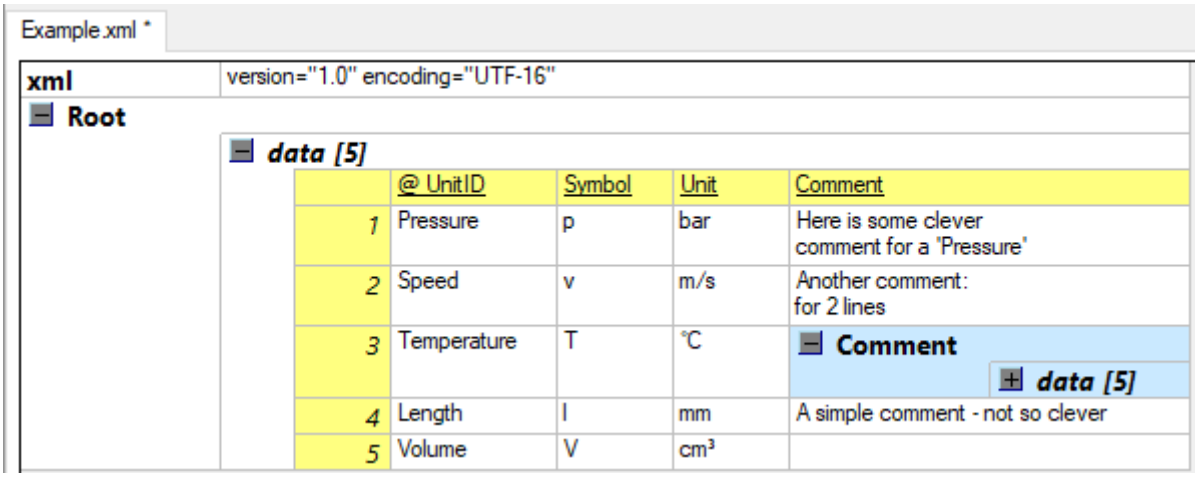


3.3 Copy a part of an xml document to some other place

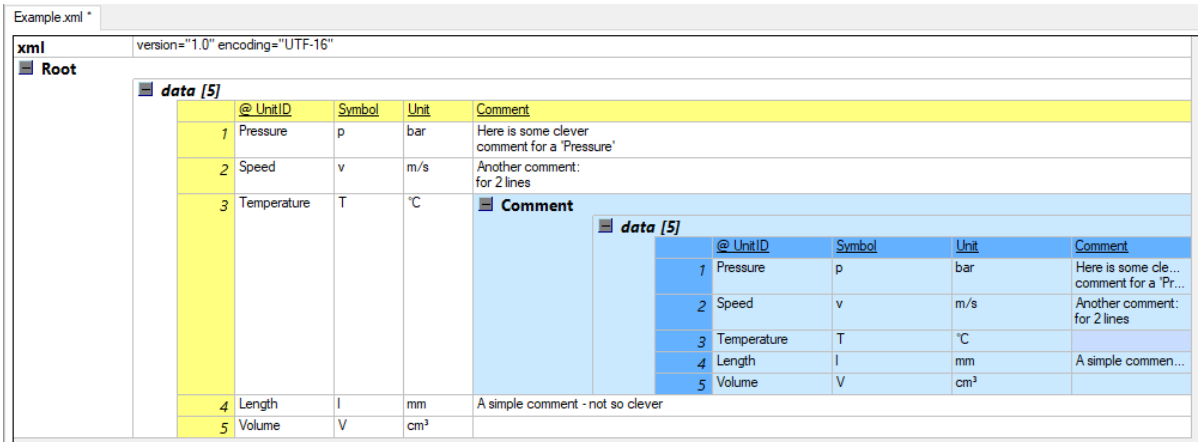
It's possible to select a part of xml data, copy it and paste to another place in the same or other document. We can copy a table for example but it could be anything we can select…



…and paste it inside the table on the place of a "Comment" element on the 3rd row:

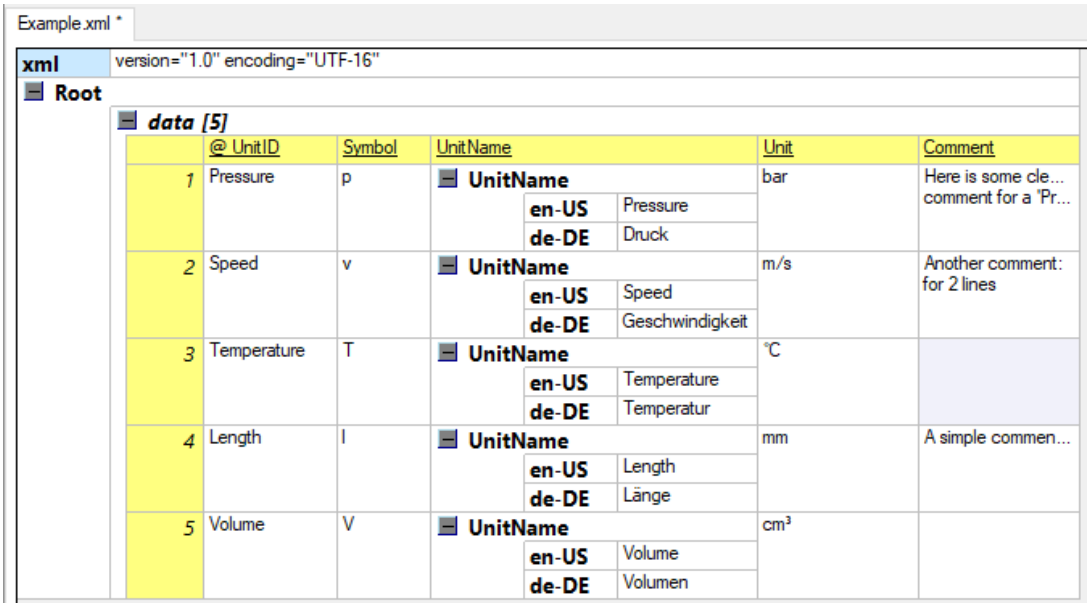


And the same picture once more time after we expand a table "data" in the "Comment" column.



3.4 Manipulate data nested inside a structure

The next demonstrated functionality can be very useful and effective in some cases. Let's have a table which contains some nested data.



Example.xml *

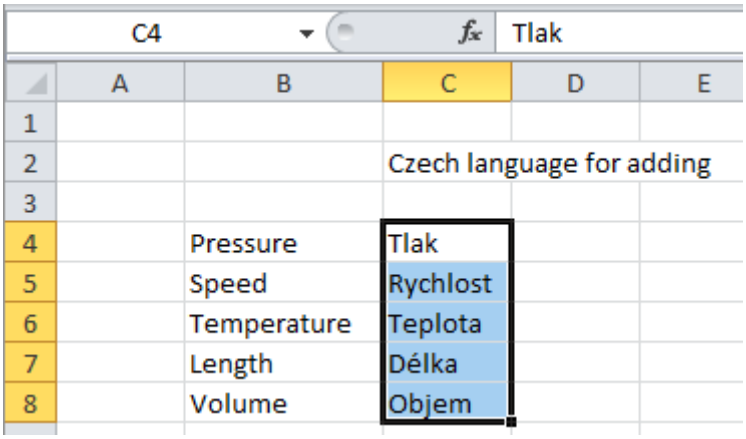
xml version="1.0" encoding="UTF-16"

Root

data [5]

	@ UnitID	Symbol	UnitName		Unit	Comment
1	Pressure	p	UnitName		bar	Here is some cle... comment for a 'Pr...
			en-US	Pressure		
			de-DE	Druck		
2	Speed	v	UnitName		m/s	Another comment: for 2 lines
			en-US	Speed		
			de-DE	Geschwindigkeit		
3	Temperature	T	UnitName		°C	
			en-US	Temperature		
			de-DE	Temperatur		
4	Length	l	UnitName		mm	A simple commen...
			en-US	Length		
			de-DE	Länge		
5	Volume	V	UnitName		cm^3	
			en-US	Volume		
			de-DE	Volumen		

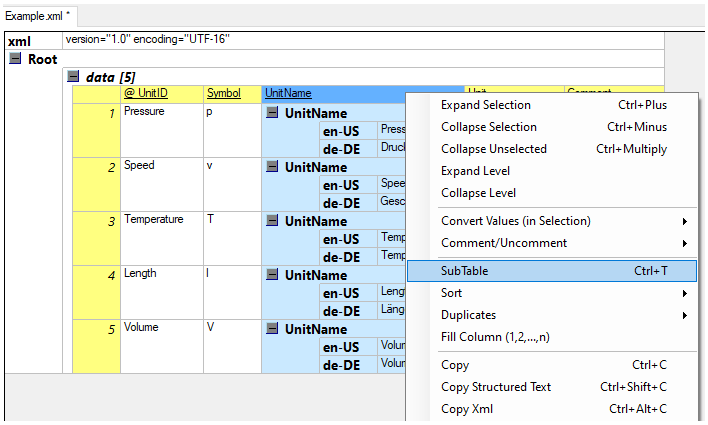
Now we would want to add a new Czech language data for the "UnitName" column. We have already prepared data in Excel...



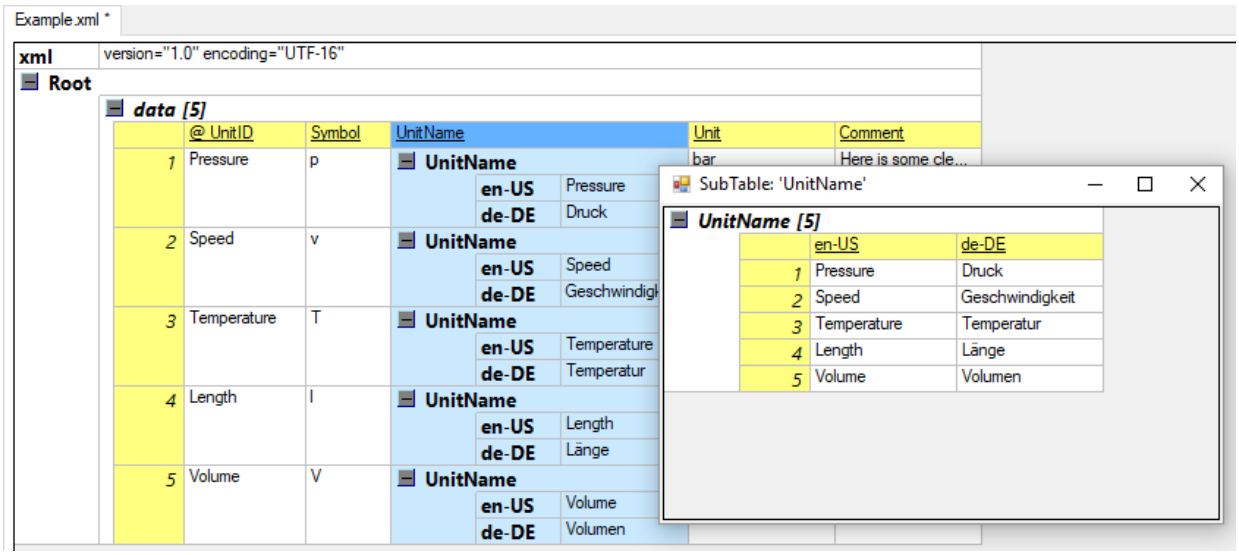
C4 | fx | Tlak

	A	B	C	D	E
1					
2			Czech language for adding		
3					
4		Pressure	Tlak		
5		Speed	Rychlost		
6		Temperature	Teplota		
7		Length	Délka		
8		Volume	Objem		

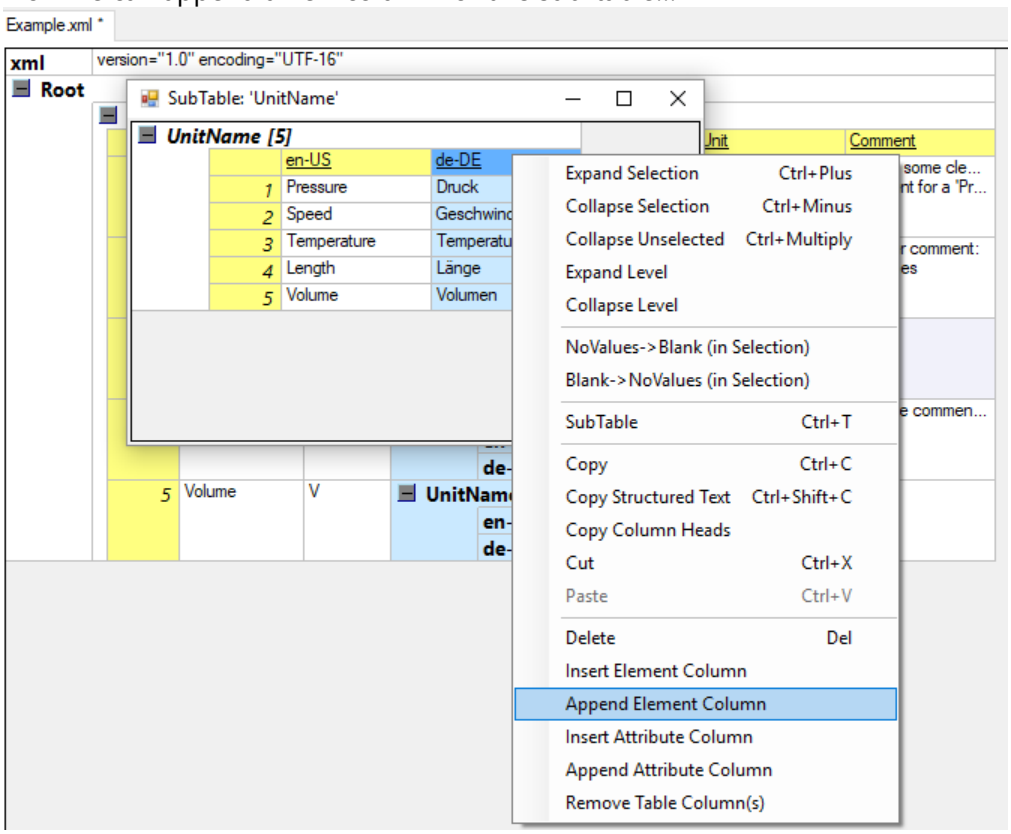
We use a function “SubTable” applied on the column “UnitName”…



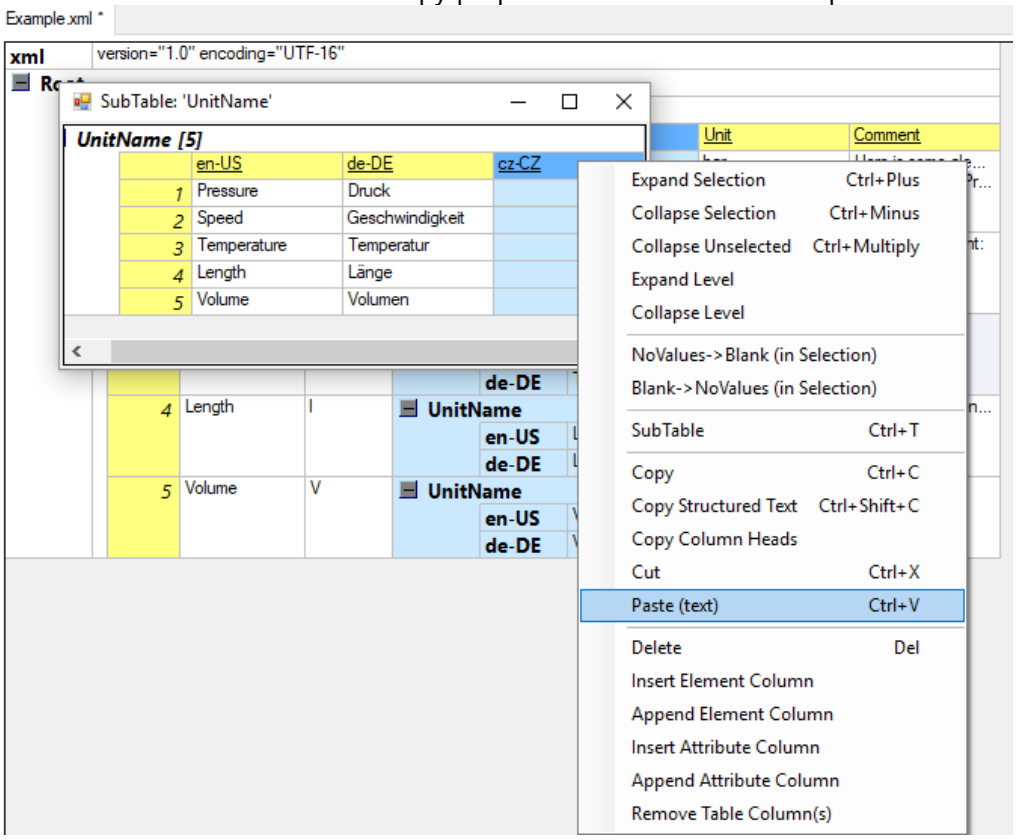
A column “UnitName” is displayed in a new form without the top horizontal level (if we need to manipulate data which are nested deeper we can call a sub-table repeatedly to expose the essence):

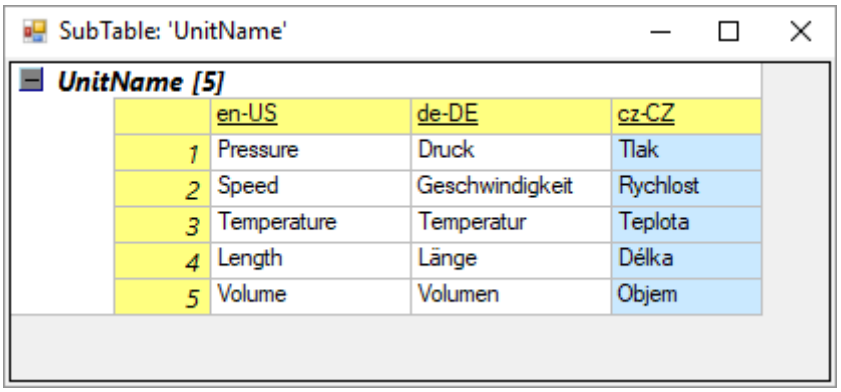


Now we can append a new column for this sub-table…

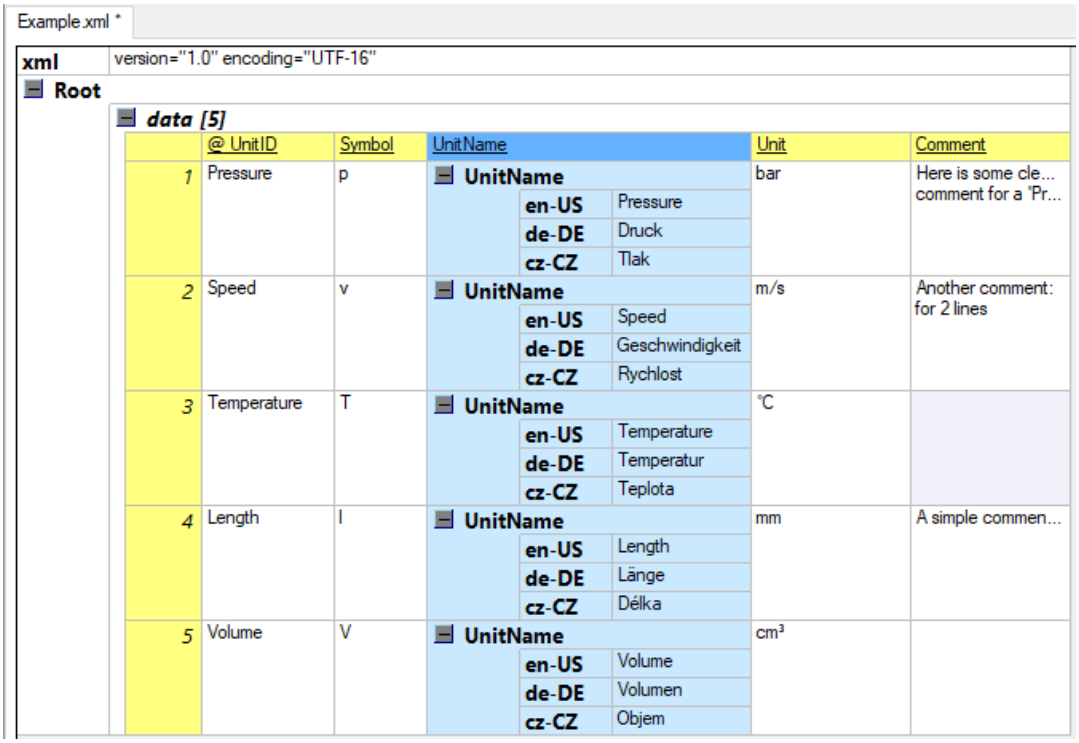


…rename the column to ‘cz-CZ’ copy prepared data from Excel and paste them to this new column





After that we close the sub-table, return back and we are done.



Effective and ximple, isn't it?

3.5 Copy and modify data in a table

We would want to create some new table rows based on already existed ones. Let's have a table…

Example3.xml

xml version="1.0" encoding="UTF-16"

Root

data [6]

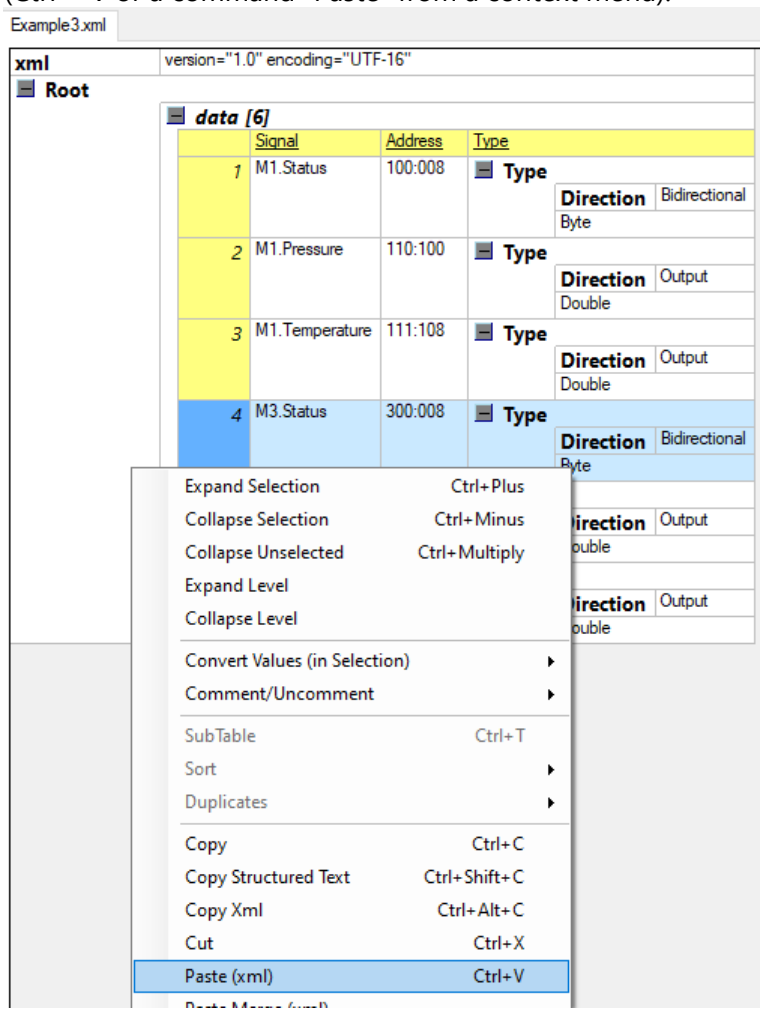
	Signal	Address	Type
1	M1.Status	100:008	Type Direction Bidirectional Byte
2	M1.Pressure	110:100	Type Direction Output Double
3	M1.Temperature	111:108	Type Direction Output Double
4	M3.Status	300:008	Type Direction Bidirectional Byte
5	M3.Pressure	310:100	Type Direction Output Double
6	M3.Temperature	311:108	Type Direction Output Double

…and we would want to add records for M2 which should be on the base address 200.

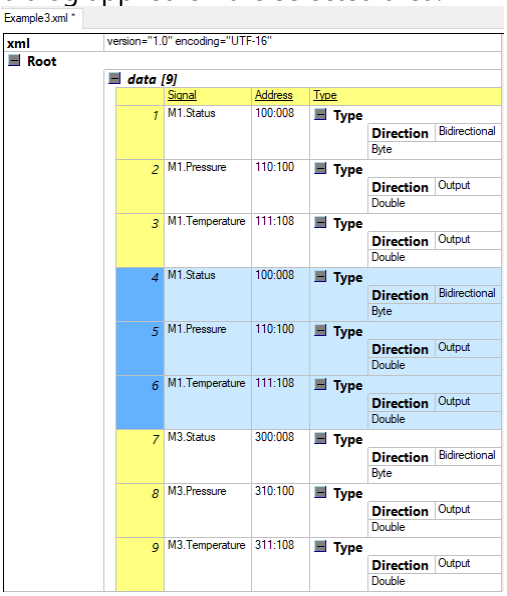
We reuse the first 3 rows of a table and duplicate them between M1 and M3. Then we replace M1 texts for M2 and change addresses 1xx:yyy for 2xx:yyy.

At first we will select the first three rows and copy them (Ctrl + C or a command "Copy" from a context menu) to clipboard.

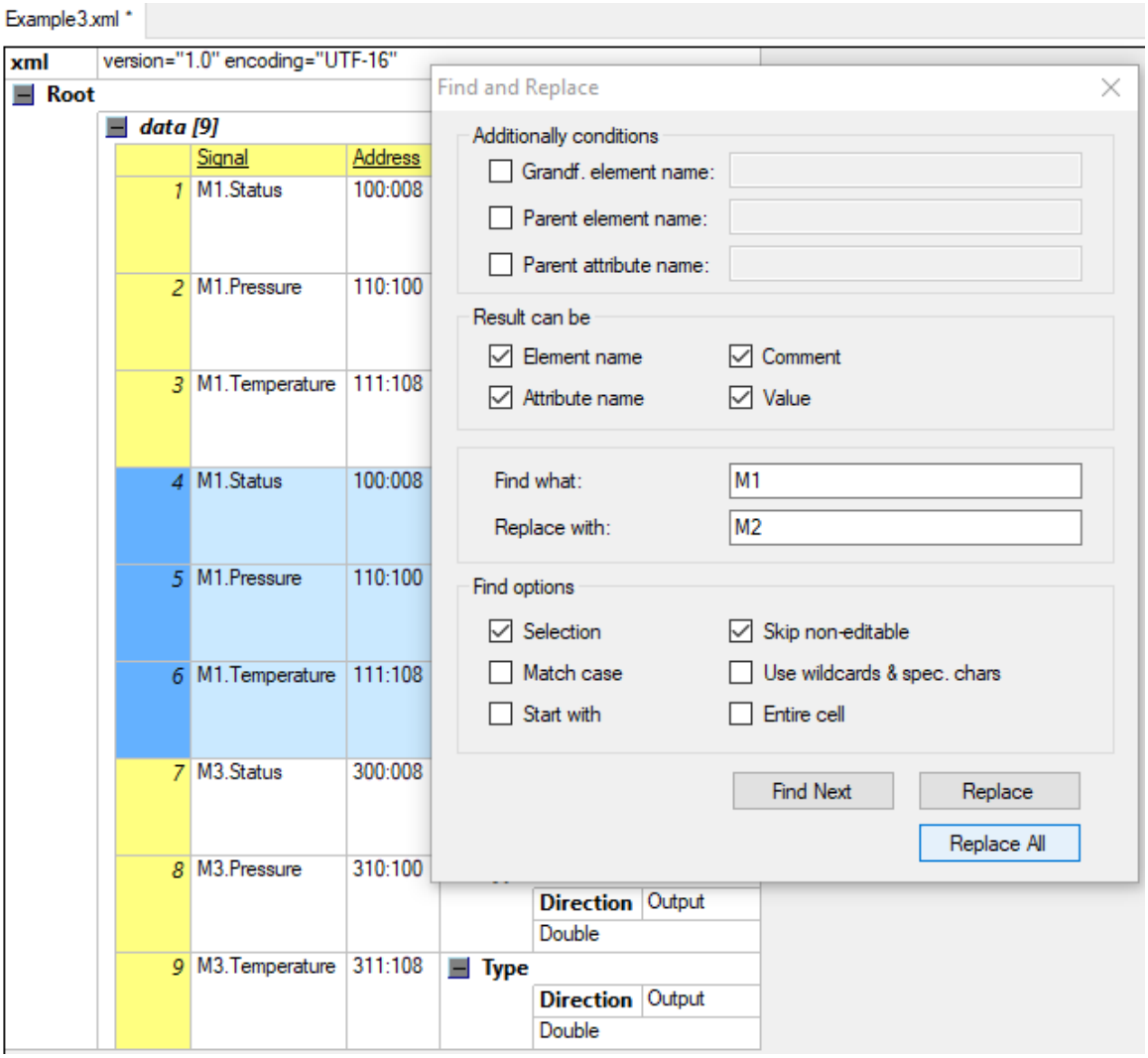
Then we paste data in the middle of a table before M3 records. Data are pasting into a table before the selected row (the only one exception is the last row). So we select the 4th row and paste data (Ctrl + V or a command “Paste” from a context menu).



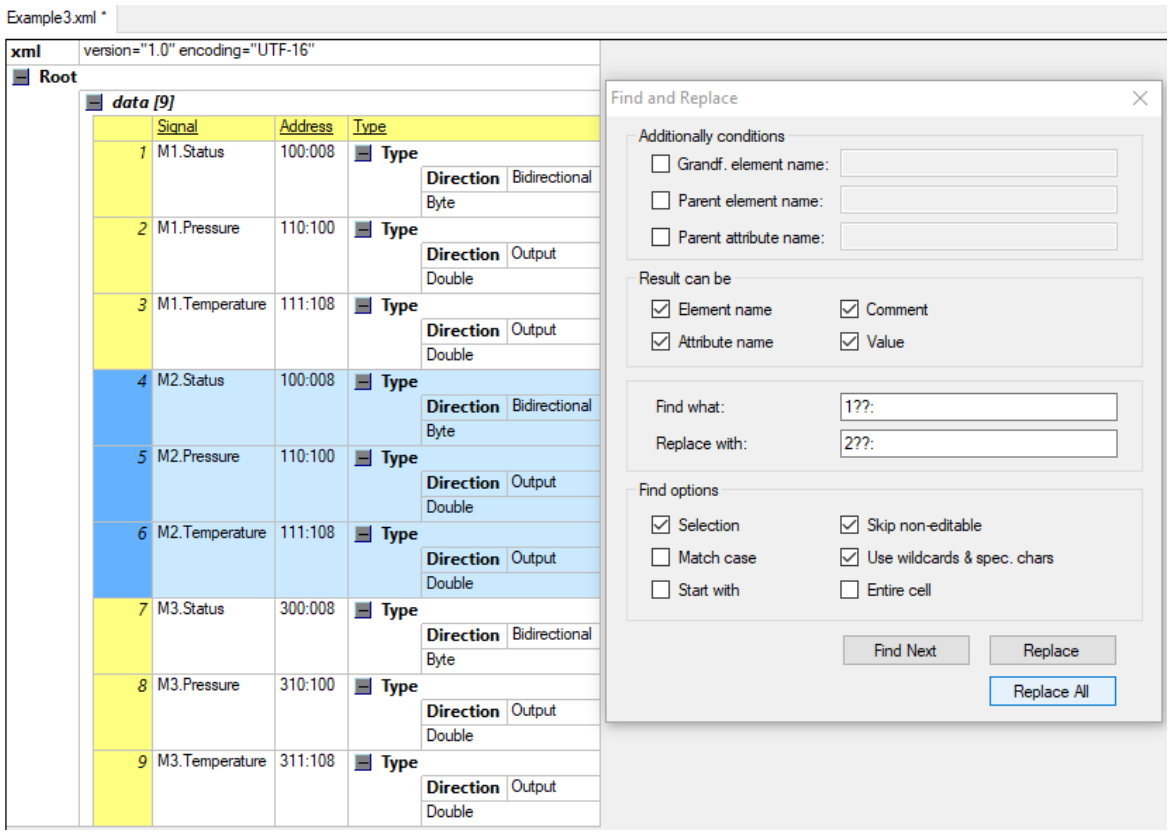
Rows are duplicated and now we modify some values. For this purpose we can use Find/Replace dialog applied on the selected area.

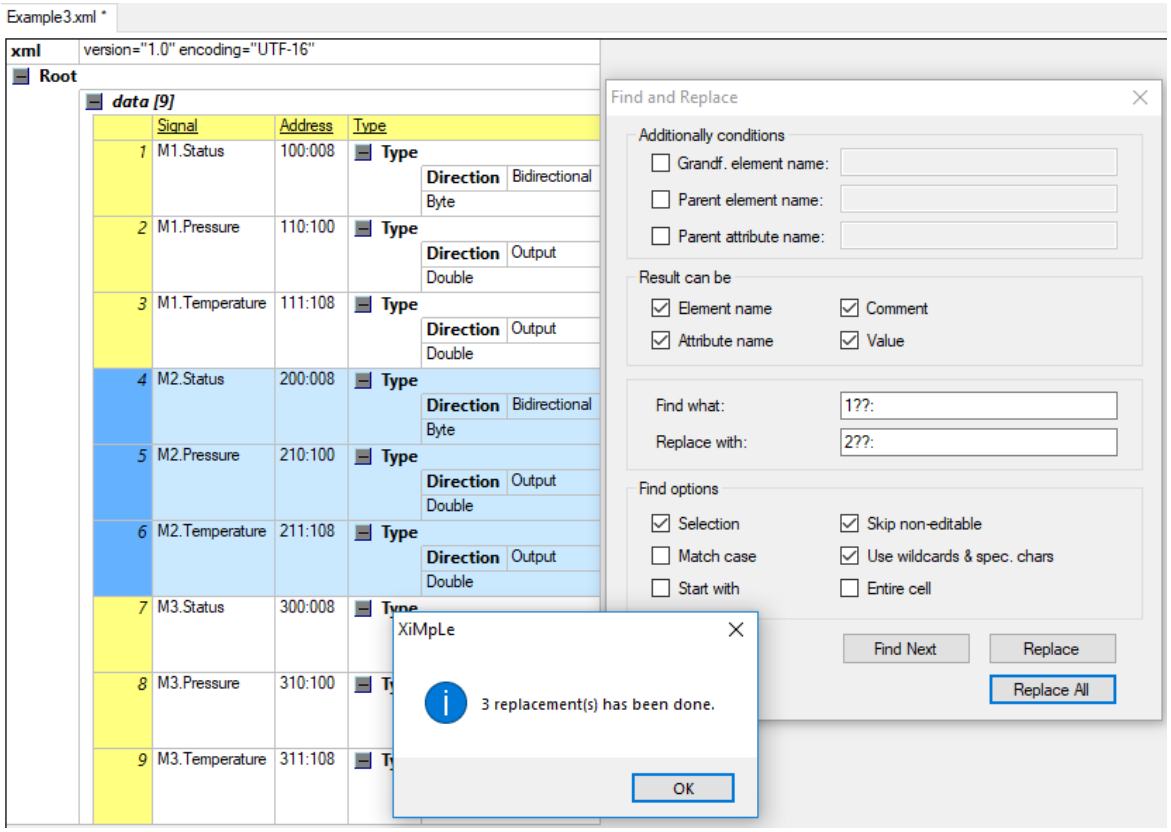


Firstly we replace all texts “M1” for “M2” in the selected area…



For next replacing of "100:" base addresses to "200:" base addresses we can use wildcards - we don't want to change addresses after the colon mark so we should specify only the first part of an address (see a picture).





4 Clipboard operations

All clipboard operations are realized in a "transparent" text mode. That means a user can access and modify data in clipboard without losing a possibility to finish the operation. For data transfer there are used two modes: “Simple text” and “XML text”.

1. *Simple text*

This mode is used for data which don't require any special XML structuring. Data are usually organized in the form of a table - one line for each row and columns separated by the same separator. This mode is especially suited for transferring from/to a spreadsheet editor.

2. *XML text*

This mode is used to handle non-simple structures and it is formed by data and XML tags.
To recognize this mode there must be a special text at the beginning of data on the first line.
Additional requirement is a well-formed xml data structure. You can see the section [11 Copy] for more details.

5 Tables

If XML data contain elements with the same name on the same level of a structure they can be formed to a table. These elements form rows of a table and inner elements form columns of a table. Rows can contain different elements and that mean also different columns.

```
<Person><Name>John</Name><Age>25</Age></Person>
<Person><Name>Paul</Name><Sex>Male</Sex></Person>
```

In this example we can recognize two rows which are formed by the element “Person”. In the first row there is a column “Name” and “Age”. In the second row there is a column “Name” and “Sex”. XiMpLe arranges information to a table like this:

Person [2]

	Name	Age	Sex
1	John	25	
2	Paul		Male

Notice that there is missing an element “Sex” in the first row and in the second row there is missing an element “Age”. If some element is missing in a table then a “no-value” is used in the related cell.

Please note there is a difference between an empty cell and a cell with no-value. Empty cell means that the element exists and its value is an empty string. No-value cell means that the element doesn’t exist at all.

Note: After the name of a table “Person” there is in square brackets number of table rows “[2]”.

The column order of the table is determined:

a) If a table contains a row which contains all elements, this row will define the order of columns in the created table.

b) Otherwise, program tries to find correct order according the order of elements in the source file. For instance, if element <A> is ahead of element <B> then column A should be ahead of column B.

Generally, there can be more solutions or even no solution.

For example, let's have this part of xml file:

```
<Table><A>a1</A><B>b1</B></Table>
<Table><C>c2</C><B>b2</B></Table>
```

For this table the columns can be arranged in the sequence A-C-B or C-A-B. There exist more solutions.

In another example the fragment defines two rows of a table and there is no optimal ordering of columns A and B.

```
<Table><A>a1</A><B>b1</B></Table>
<Table><B>b2</B><A>a2</A></Table>
```

The rows which have different internal ordering are marked with a sign "x" on the left side of the table row number.

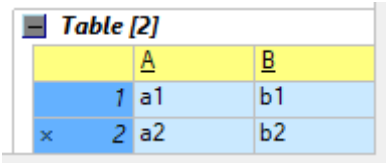


Table [2]

	A	B
1	a1	b1
x 2	a2	b2

Applying the "order" command (step 6 in section 5.1) we will make this internal change:

```
<Table><A>a1</A><B>b1</B></Table>
<Table><A>a2</A><B>b2</B></Table>
```

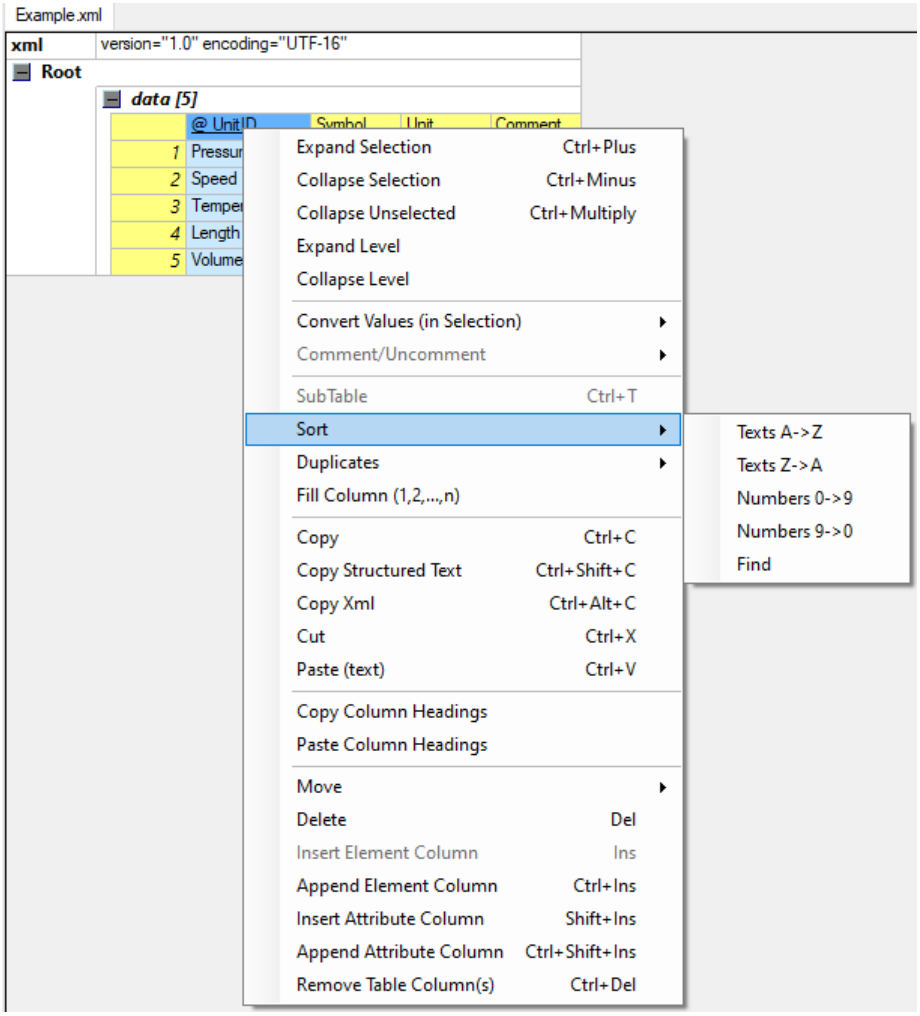
5.1 Special commands related to a table

A table is a widely used object in XiMpLe and that's why there exist some special rows and columns oriented commands which can be invoked from a context menu if some row/column is selected.

1. Inserting an empty row to a table before the first selected row
2. Appending an empty row to a table behind the last selected row
3. Deleting a row or multiple rows
4. Moving selected rows (for moving rows it's probably handier to use the keyboard shortcuts – see [27 `Shortcuts`])
5. Copy and paste row(s)
6. Order Elements according Columns – in case there is no optimal columns ordering, some of table rows can have different internal ordering than table columns ordering - in such case this command is enabled in context menu and we have a possibility to arrange internal order of elements according the table columns order
7. Inserting an empty element or attribute column to a table before the first selected column (please note that attributes are always arranged before elements)
8. Appending an empty element or attribute column to a table behind the last selected column
9. Removing a column or removing multiple columns
10. Moving selected columns (for moving columns it's probably handier to use the keyboard shortcuts – see [27 `Shortcuts`] or drag & drop)
11. Copy and paste table headings

12. Filling one selected column by ordinal numbers if the column contains only simple text (if not we can simply delete all values in the column and then fill it). This command overwrites the values in a selected column.
13. Sorting a table by a selected column – only one column can be selected for sorting

    We can use ascending or descending sorting for texts or for numbers. All rows and columns of a table are affected by sorting and the change is physical, not only virtual.



The selection “Numbers 0->9” (ascending) or “Numbers 9->0” (descending) will try to convert all texts to numbers before sorting.
The commands “Texts A->Z” and “Texts Z->A” are sorting by an alphabet.
Information for advanced users: text sorting is done as “culture invariant” and all sorting should be stable (entries with the same values are not swapped).

The command "Find" tries to find in each cell of the selected column a text which is entered in the Find dialog [see 20.3]. If the text is found in the column's cell (or inside its children) such row is moved up. At the end all table rows, in which the used column cell fits the searched criteria, are at the top of the table.

Tip: This feature could be combined with the exposing table function [see 5.2] to get more complex sorting results.

Example:
Let's some column contains the values: "10", "NoValue", "020", "5.3", "Dummy", "0.05e6".

Results of different sorting commands will be:
"Numbers 0->9": "NoValue", "Dummy", "5.3", "10", "020", "0.05e6"
"Numbers 9->0": "0.05e6", "020", "10", "5.3", "NoValue", "Dummy"
"Texts A->Z": "0.05e6", "020", "10", "5.3", "Dummy", "NoValue"
"Texts Z->A": "NoValue", "Dummy", "5.3", "10", "020", "0.05e6"

Remark: Originally it was possible to sort only column which contains simple values. Now it's possible to sort also by a column containing structures. In such case the sorting is done by the name of the first element (or attribute) in the structure, then by the name (or value) of the first child of that element and then by the column's text value if it exists.
In many cases such sorting has no practical meaning - the main purpose of this feature is to group empty or simple column values and the complex structures together.

14. Find duplicates in a table by selected columns

Duplicates are found according the selected columns. Values and elements must be the same in all selected columns to be counted as duplicate. If the whole table is selected, all columns are considered as selected.

If some column contains complex element with the children, the setting *Ignore Order* from the context submenu can affect the result (see also example below).

For duplicates we can do these actions:

a) *Count* number of duplicates and the result copy to clipboard
b) *Group* duplicates together and move them to the end of the table
c) *Remove* duplicates from the table

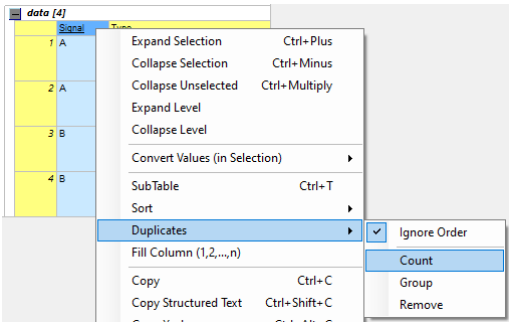
Example:

Let's have a table:

data [4]

	Signal	Type
1	A	**Type**: **B** Input; **C** Output
2	A	**Type**: **C** Output; **B** Input
3	B	**Type**: **B** Input; **C** Output
4	B	**Type**: **C** Output; **B** Input

i) If only the first column is selected, there are found two duplicates – row 1 with row 2 and row 3 with row 4. The *Ignore Order* option has no effect in this case because there are no complex elements in the first column.



ii) If the second column is selected and *Ignore Order* is set, there are found 3 duplicates (row 1 with rows 2, 3, 4).

iii) If the second column is selected and *Ignore Order* is not set, there are found 2 duplicates (row 1 with row 3 and row 2 with row 4).

iv) If both columns are set, we get 2 duplicates (row 1 with row 2 and row 3 with row 4) if we ignore order; or get no duplicates if we don't ignore order.

5.2 Expose table row(s)

By this function we can virtually separate one or more table rows from the table. One selected row will be displayed as an isolated element; more selected rows will create a new table inside existing table.

data [6]

	Signal	Address	Type
1	M1.Status	100:008	Type
2	M1.Pressure	110:100	Type
3	M1.Temperature	111:108	Type
4	M3.Status	300:008	Type
5	M3.Pressure	310:100	Type
6	M3.Temperature	311:108	Type

data [2]

	Signal	Address	Type
1	M1.Status	100:008	Type
2	M1.Pressure	110:100	Type

data [2]

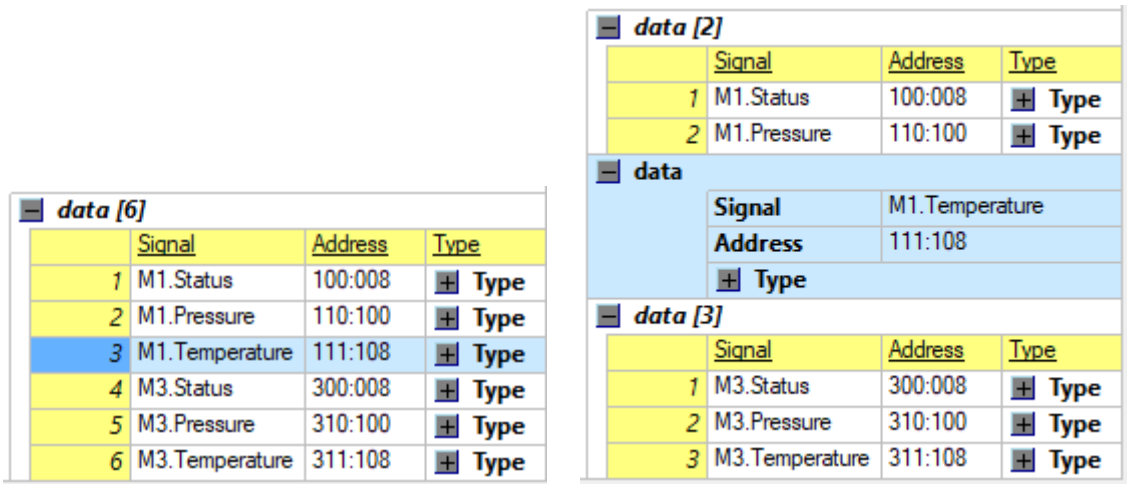
	Signal	Address	Type
1	M1.Temperature	111:108	Type
2	M3.Status	300:008	Type

data [2]

	Signal	Address	Type
1	M3.Pressure	310:100	Type
2	M3.Temperature	311:108	Type

Third and fourth row are exposed and they form a new table (on the right)

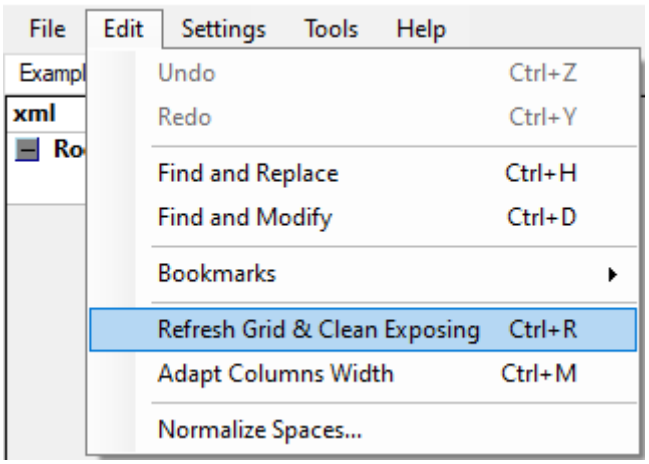
In principle this function can divide the table up to three tables all with the same name. Each table can be manipulated separately, renamed or we can insert a new element in between.



Third row is exposed and it's displayed as an isolated element (on the right)

At one moment there can be only one exposed area. New exposing will cancel the previous one.

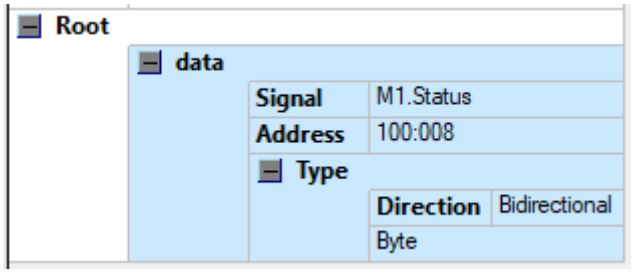
Exposing can be revoked by a refresh function from menu Edit->Refresh Grid & Clean Exposing or by a shortcut Ctrl + R.



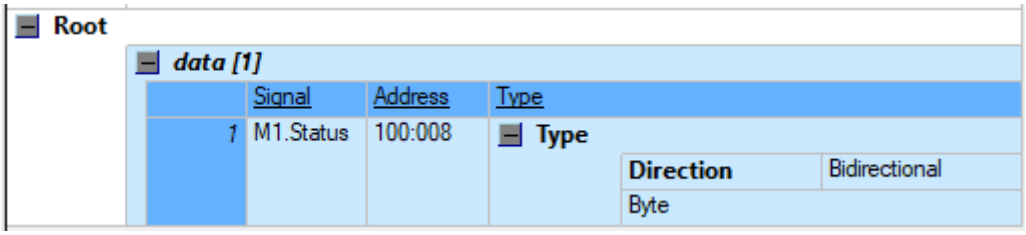
Note: When the Compare mode is in use and both compared files are displayed in the table grid view the exposing is cleaned and the function itself is not allowed while comparing two xml files. To make it ready again just switch one file to text view or return to standard mode.

5.3 Expose element as table row

This command can show a single element as a table with one table row.



Element data will be displayed as one row table.



Such table can be manipulated in the same way like any regular table. Exposing can be revoked by refreshing (Edit->Refresh Grid & Clean Exposing / Ctrl + R) or by a function Expose table row [5.2].

If some other row is added it becomes normal “permanent” table.

This command is not available in Compare mode while comparing two table grid views (same conditions as a function Expose table row described in a previous section).

6 Delete

Selected cell or a whole area can be deleted. If an element is selected and deleted then this element and everything what it contains is removed. If a value is selected and deleted then this value is set to an empty string.

For table elements/values we can set the cell to “no-value” which means that not only a value is empty but also an element should be removed. A “no-value” is indicating by a different color and in such case an element is not saved to the result file.

If a used command is “delete” then empty string is set to a cell value. If a used command is “remove” then no-value is set to a cell value. Moreover there is a possibility to replace all empty cells in a selected area by no-values or vice versa.

Note: For values which are not a part of a table there is no difference between delete and remove.

The whole row or a range of rows can be selected and deleted. It deletes selected row(s) completely. If all rows of a table are deleted a whole table is removed as well. If only one row remains a table is switched to an element.

The whole column or a range of columns can be selected and deleted. In this case only the values are deleted. To remove selected column(s) completely there is a special command “Remove Table Column(s)”. Columns can’t be removed all – at the end at least one special column “<Text>” will remain and it keeps the values of row elements.

7 Sub-tables

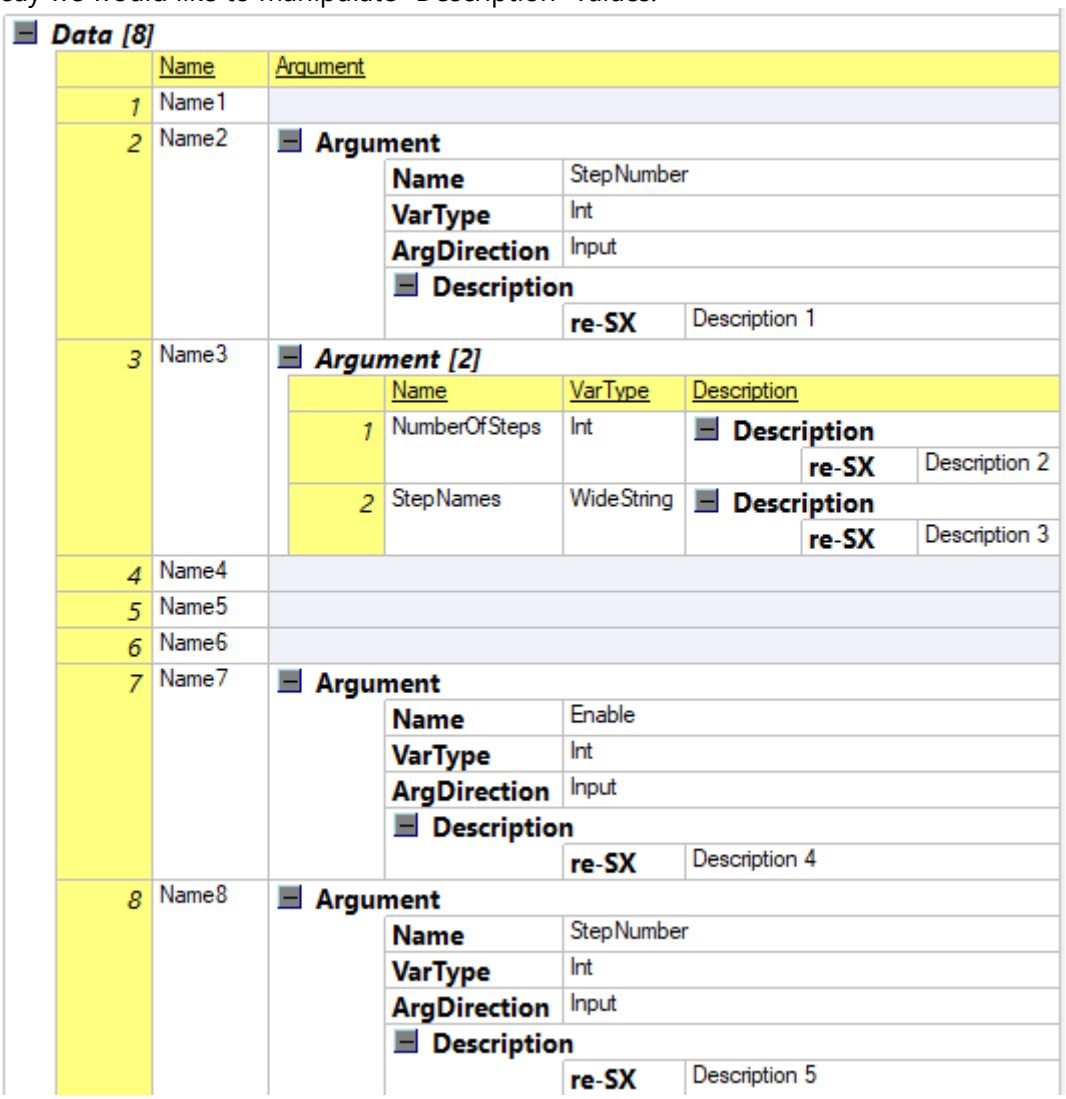
For data which are nested deeper inside a table and can't be accessed in a simple way a user can use a special functionality – sub-table – which can regroup data and make manipulations easier.

The sub-table function can be applied on one non-trivial table column. It opens a new modal window and displays the column data in a new form. Technically it removes one level of data from the view. A function can be repeated to open a new sub-table from an already opened sub-table.

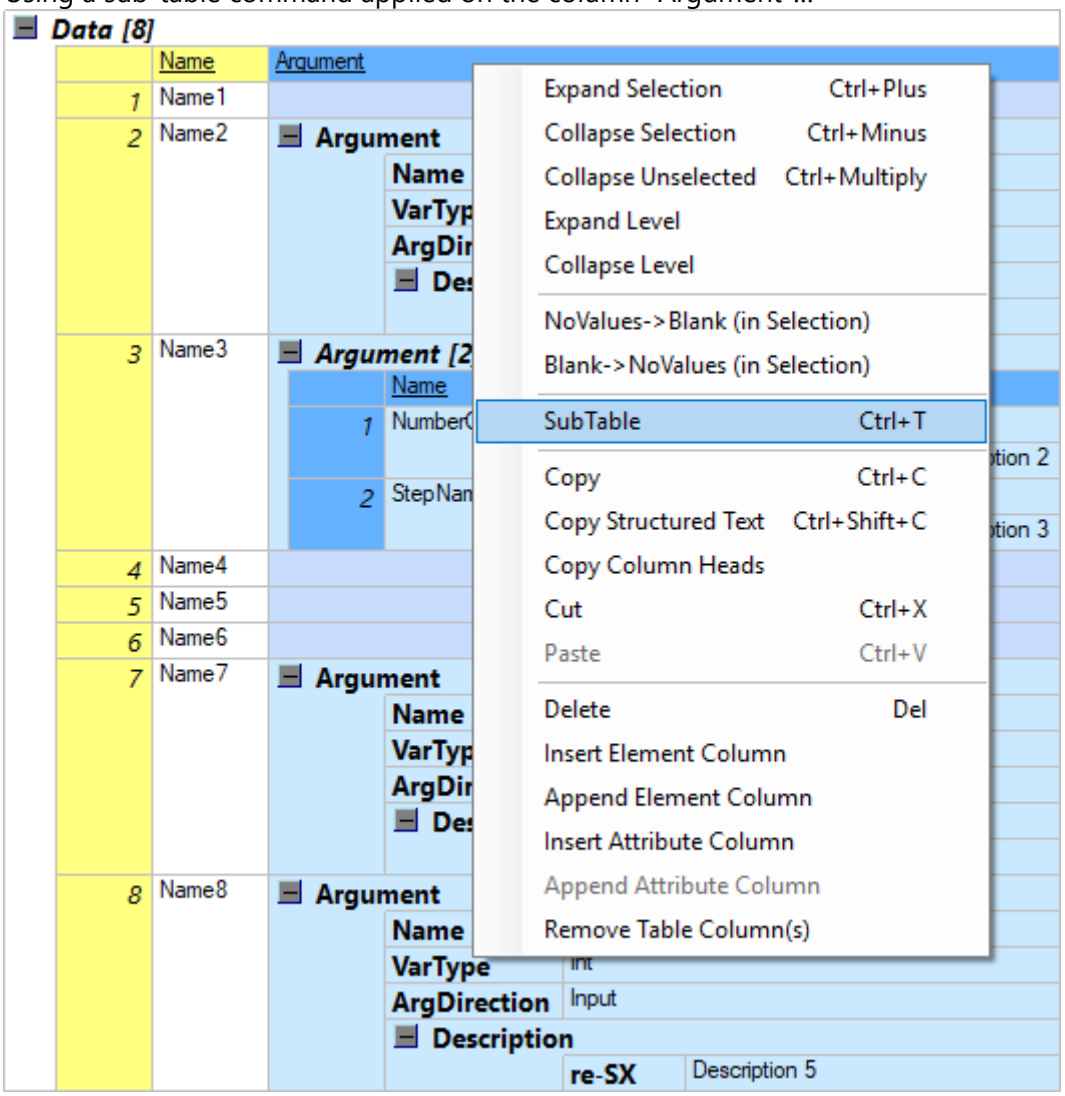
Data in a sub-table can be edited as usual but no new lines can be added or deleted because data must be keeping consistent considering the parent table. Adding or removing columns is possible. Adding elements inside the sub-table, copying or pasting data are allowed as well.

In a sub-table there isn't possible to do “undo” or “redo” function but “undo” can be done at the end when all sub-tables are closed and a focus is returned to the main window.

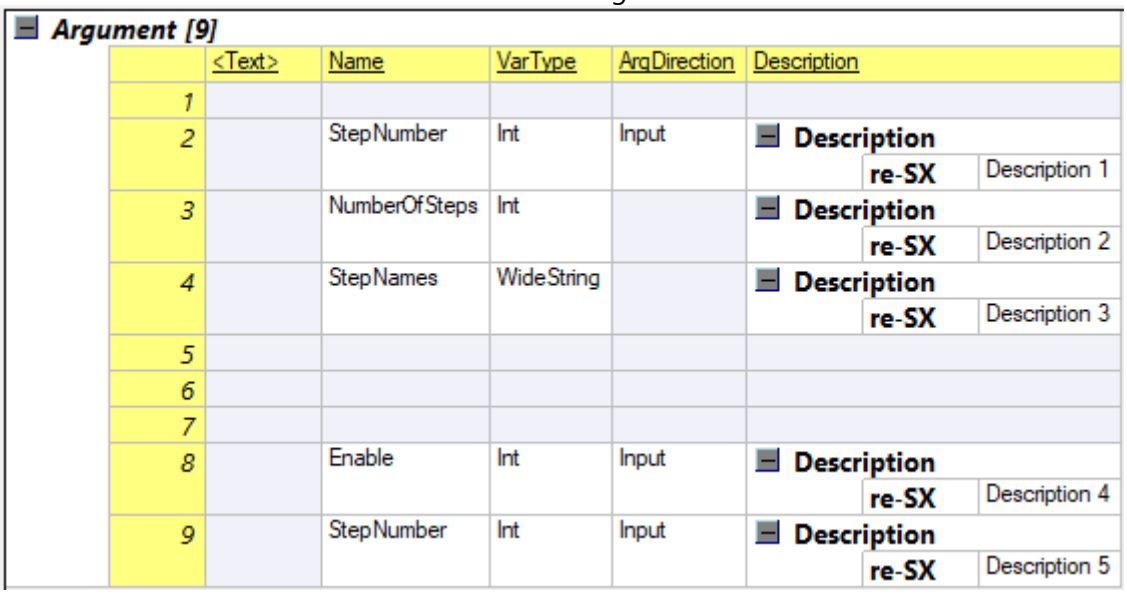
To demonstrate the sub-table functionality let’s take some more complex data structure and let’s say we would like to manipulate “Description” values.



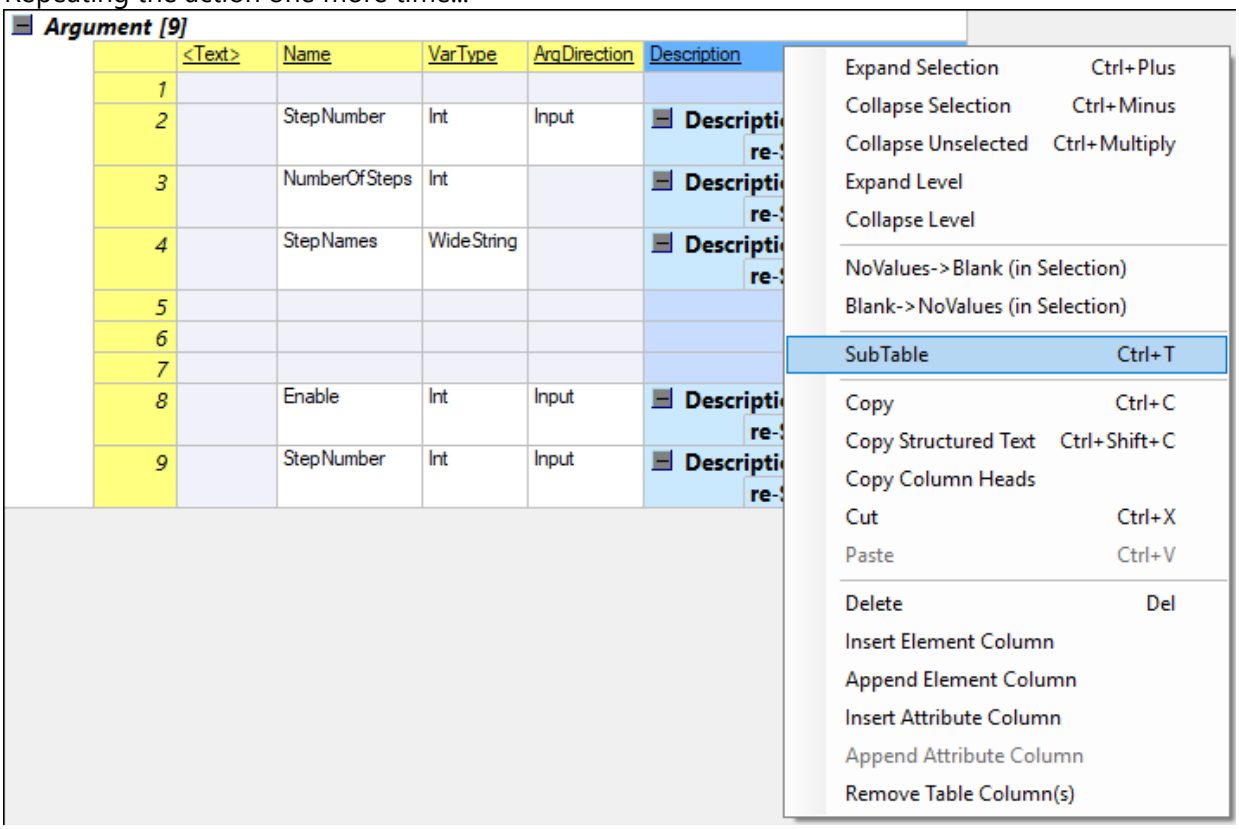
Using a sub-table command applied on the column “Argument”…



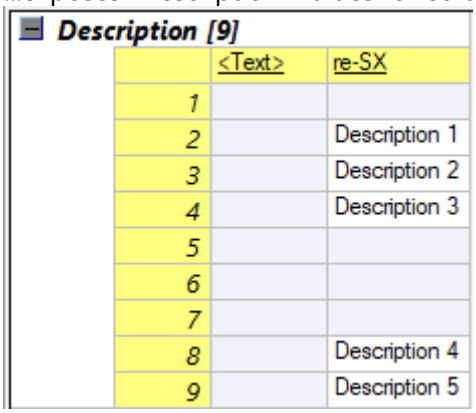
…we can observe a little better table to reach our goal.



Repeating the action one more time...



...exposes "Description" values for editing.



Now we can simply edit, copy or paste values from/to the column “re-SX”. Another useful possibility is renaming a column or adding a new column if desired.

Description [9]

	<Text>	re-SX	NewColumn
1			
2		Description 1	New Value1
3		Description Modified	
4		Description 3	
5			New Value 2
6		New Description	
7			
8		Description 4	
9		Description 5	

After modifications when closing all sub-table windows and return back we can overlook the result.

Function [8]

	Name	Argument
1	Name1	
2	Name 2	**Argument** **Name**: StepNumber **VarType**: Int **ArgDirection**: Input **Description** **re-SX**: Description 1 **NewColumn**: New Value1
3	Name 3	***Argument [2]*** 1 — Name: NumberOfSteps; VarType: Int; Description → re-SX: Description Modified 2 — Name: StepNames; VarType: WideString; Description → re-SX: Description 3
4	Name 4	**Argument** **Description** **NewColumn**: New Value 2
5	Name 5	**Argument** **Description** **re-SX**: New Description
6	Name 6	
7	Name 7	**Argument** **Name**: Enable **VarType**: Int **ArgDirection**: Input **Description** **re-SX**: Description 4
8	Name 8	**Argument** **Name**: StepNumber **VarType**: Int **ArgDirection**: Input **Description** **re-SX**: Description 5

8 Editing values

The most of data shown in the editor can be modified and changed. To change some value we can double click the value (it can be also a name of an attribute or a column name). Another possibility is to press F2 to start edit selected cell or just start a typing. An edit box opens and we can edit the value or type a new one.

Editing can be ended by clicking somewhere outside the edit box or by pressing Enter or Control + Enter or, and it should work always, Alt + Enter.

If we are editing a cell which has had only one line or we have started by typing (and the cell will be overwritten by the typing) then Enter ends the editing and Control + Enter add a new line.

If we are editing (by F2 or double clicking) a cell which has had more than one line then Enter adds a new line and Control + Enter ends the editing.

Escape will cancel editing and all changes will be lost.

Note: Some values like elements' or attributes' names can't have multiple lines.
Note: It can happen that entered text is not a valid value (e.g. element name can't start with a number or it can't contain some characters and so on) and can't be accepted. In such case an info message box appears.

9 Expand and collapse

For a better orientation between data we have a possibility to "hide" some information currently not needed. Elements which contain some complex information and tables are displayed with a small button in a cell's corner. Using this button we can collapse (hide) or expand (show) interior data. This functionality can increase clarity of data we are focused on.

There are several commands which are able to collapse or expand multiple cells at once based on a selected area. Also in between general settings [21.1 `General`] there are some options related to this topic.

1. Expand the selected area - expands all cells inside the selected region including all internal data.
2. Collapse the selected area - collapse all cells inside the selected region including all internal data.
3. Collapse the unselected area - collapse all cells outside the selected region including all internal data.
4. Expand level – expands all cells which are on the same level of nesting as the place where a context menu was invoked. Internal data collapse/expand information are unaffected. For this command is important on which place we have invoked context menu. Selected area is irrelevant in this case!
5. Collapse level – collapses all cells which are on the same level of nesting as the place where a context menu was invoked. Internal data collapse/expand information are unaffected. For this command is important on which place we have invoked context menu. Selected area is irrelevant in this case!

10 Selection

Usually a user is not working with all data at the same time and it's necessary to select only a range of data which will be affected by some operation.

To select some cell(s) we can use a left mouse clicking together with a mouse button holding and mouse moving along. All focused cells are gradually selected and put together to the selected range if possible (on the top structure level there are some restrictions).

There are other options how to select cells: holding a shift key with a left mouse clicking or with cursor keys. A combination with a shift key pressed adds a currently selected cell to the same range with already selected cells by forming a rectangular area's shape.

There are some special cases of a selection:

- selecting a whole table by left clicking on the top left corner cell of a table
- selecting a whole table row by left clicking on the row number cell of a table
- selecting a whole table column by left clicking on the column head name of a table

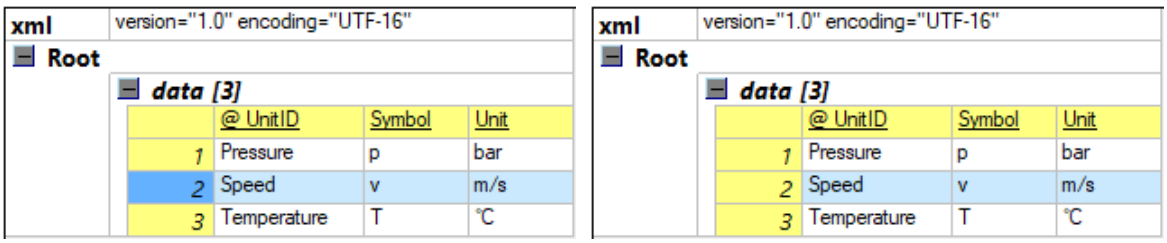
To select multiple rows or columns we can use the previously described combination of a shift key and cursor keys as well.

The control key with cursor keys can be used inside a table to move the cursor on the edge of a table in the cursor direction.

The alt key with cursor keys can be used inside a table to move the cursor on the first different value type. For this functionality a program recognizes four value types: no-value, empty string, non-empty string and element structure. It tries to find the first different value in the cursor direction – if all values are of the same type nothing happens. This can be used e.g. to find missing empty values in a table's column.

Tip: We can select a row of a table and then hold down the shift key and press a control key and a cursor down: all rows from the selection to the end of a table should be selected.

Note: Some actions require a selection of a whole row and not only a selection of all cells in a row.



10.1 Convert values (in Selection)

Values in a selection can be modified by some special functions which appear in the context menu under the group "Convert values (in Selection)". For these functions there is a common rule - they change only element values (not element names) and only values which are exposed and not inside collapsed branches. This could give a user freedom to "hide" values which shouldn't be affected by the conversion.

The available functions are:

- *NoValues->Blank*
  Changes all “no-value” to “empty string” (see also section [5. Tables])

- *Blank->NoValues*
  Changes all “empty string” to “no-value” (see also section [5. Tables])

- *Base64->Text*
  Values in xml might be encoded in the Base64 and this function can be handy if we want to decode such value(s). For this function is important the encoding scheme, which has been used for encoding from text to Base64. In most cases the “utf-8” is used, that’s why utf-8 is used by default.
  In case a different encoding is required there is a possibility to define new encoding in the Settings.xml file on the XPath: /XiMpLe/Schema[#]/Data/Base64Encoding=“utf-8”.
  Some of the allowed values: "utf-8", "utf-16" or "Unicode", "utf-16BE" (big endian Unicode), "utf-7", "utf-32".
  Each schema can have its own decoding and switching the schema could change the encoding for Base64 conversion values. The used encoding is displayed in the tooltip for this function and also in the name of the function in the context menu.

  Please note that this function can fail if the input text is not a valid Base64 string. In such case the text is not changed (decoded) and the message box appears with the information how many values couldn’t be converted.

- *Text->Base64*
  This is the inverse function to the Base64->Text function. Also, for this function is used encoding scheme described in the previous function.

  Please note that encoding of the xml itself has no relation to the encoding scheme used for these two functions. For example, if we have xml in utf-16 we can convert (and it will convert by default) texts to Base64 based on utf-8 encoding scheme.

- *Text->GZip*
  Compress non-empty texts in selected cells by GZip algorithm. Result of compression is binary, so it's converted to base64 text to be able to use it in xml files.
  The result of this function is depended on the defined encoding (see also command *Base64->Text* above).

- *GZip->Text*
  Decompress text compressed by GZip algorithm from the base64 text.

  Please note that this function can fail if the input text is not a valid Base64 string or if it's not a valid GZip compression. In such case the text is not changed (decompressed) and the message box appears with the information how many values couldn’t be unzipped.

- *Xml->Beautify*
  Normalize spaces (beautify) in the value if it contains correct xml structure. Otherwise doesn't change the value.
  For instance Gunzip extracts the xml content or by a "comment" function [10.1 Convert values (in Selection)] the xml structure is created in the comment's value.
  Such contents can be normalized. The parameters for the normalized spaces definition of the current schema is used for this function. See also section [21.2.5 Normalize spaces].

- *Blank->GUID*
  Fill selected blank cells (empty string values) with the globally unique identifiers. We can choose lower or upper case and encapsulating in brackets with hyphens or pure format. The “no-value” cells and cells with some values are skipped.

- *Json Text->Value*
  This command is available only for JSON file editing. Converts valid numerical values and predefined constants (false, true, null) which are marked as text to values. Texts and values are distinguished by a leading apostrophe character. So, this conversion will remove leading apostrophe for all valid values. Texts which can’t be converted to valid values are not changed. See [section 18.2] for additional details.

- *Json Value->Text*
  This command is available only for JSON file editing. Converts valid numerical values and constants to text. That means that all valid values are prefixed by apostrophe character and in the file such value will be stored in between quotation marks. For instance, ’23 will be stored as “23”. See [section 18.2] for details.

10.2 Comment / Uncomment

Selected area can be put into a comment node by "comment" command. Or commented value can be brought back to the structure by "uncomment" command.

Comment works with elements, comments or values and tries to group them into one comment block if it's possible and selection is continuous block. Otherwise more comments are created, e.g. if table's column is selected and commented. Comment command ignores selected attributes because they can't be separately commented, they must be commented together with the owner's element. If the whole content of table's column cell is selected, the command includes also the column element.

Uncomment does a reverse action to comment and it tries to recreate xml structure according the comment text. If comment doesn't contain valid xml structure text, the comment's value is assigned to the parent's element value if such value doesn't exist or it's empty. Otherwise the comment is unchanged.

Remark: The xml specification doesn't allow two minus characters "--" inside the comment. That's why whenever in the commented value appears "--" it's replaced by "⌐⌐". When the "uncomment" command is used, these special characters are reversed back to "--" again.
In principal if you do n-times "comment" and then you do n-times "uncomment", the result should be like on the beginning.

11 Copy

An area we have selected can be copied to the clipboard. There are several options of copy.

1. If the selected area contains only simple texts – that means it contains only values and there are no element or attribute names (simplified) – data are copied in a form of text values separated by a tabulator (default) or by a separator defined in the general settings [21.1 `General`].

In this example if we copy a selected area (which is simple)

data [5]

	@ UnitID	Symbol	UnitName	Unit
1	Pressure	p	**UnitName**	bar
2	Speed	v	**UnitName**	m/s
3	Temperature	T	**UnitName**	°C
4	Length	l	**UnitName**	mm
5	Volume	V	**UnitName**	cm³

And then try to paste the clipboard in a text editor we can see this content. Between the first and second column is the Tab character by default:

Speed	v
Temperature	T
Length	l

Result 1

2. If the selected area contains some non-simple cell(s) data are copied as xml text.

In this example if we copy a selected area (which is not simple)...

data [5]

	@ UnitID	Symbol	UnitName	Unit
1	Pressure	p	**UnitName**	bar
2	Speed	v	**UnitName**	m/s
3	Temperature	T	**UnitName**	°C
4	Length	l	**UnitName**	mm
5	Volume	V	**UnitName**	cm³

...and then we paste the clipboard into some text editor we will see this content (please note the special mark at the beginning of a text which identifies the xml text for the internal purposes of XiMpLe editor):

```
>XiMpLe<>

  <data>
   <Symbol>v</Symbol>
   <UnitName>
    <en-US>Speed</en-US>
    <de-DE>Geschwindigkeit</de-DE>
   </UnitName>
  </data>
  <data>
   <Symbol>T</Symbol>
   <UnitName>
    <en-US>Temperature</en-US>
    <de-DE>Temperatur</de-DE>
   </UnitName>
  </data>
  <data>
   <Symbol>l</Symbol>
   <UnitName>
    <en-US>Length</en-US>
    <de-DE>Länge</de-DE>
   </UnitName>
  </data>
```

Result 2

11.1 Copy Structured Text

If the selected area contains some non-simple cell(s) we can force the editor to copy data as structured text. In the previous example if we copy data as a structured text we can review this content:

```
v       UnitName
T       UnitName
l       UnitName
```

Result 3

Another example of the same data but this time displayed differently (one element is expanded):

data [5]

	@ UnitID	Symbol	UnitName		Unit
1	Pressure	p	**UnitName**		bar
2	Speed	v	**UnitName**		m/s
3	Temperature	T	**UnitName**		°C
				en-US Temperature	
				de-DE Temperatur	
4	Length	l	**UnitName**		mm
5	Volume	V	**UnitName**		cm³

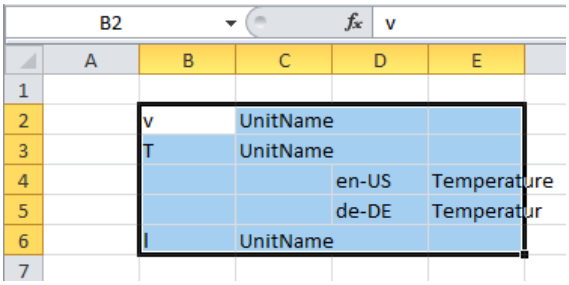
Copy the selected area in the way described in point 2 (= copy as xml data) will get the same result as the *Result 2*. Copying xml data is not affecting by displaying.

But for copying as a structured text it is important how elements are collapsed or expanded in the grid! For this example the result of a copy should look like this:

```
v       UnitName
T       UnitName
                en-US   Temperature
                de-DE   Temperatur
l       UnitName
```

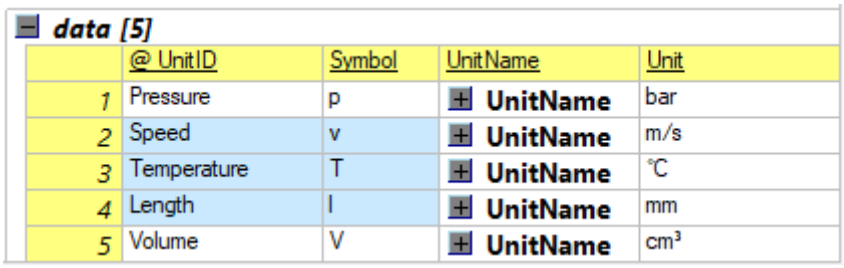
Result 4

Between texts are arranged separators – they are organized in the way to enable putting these data to Excel (for instance) and keep the displayed structure from the XiMpLe.



11.2 Copy Xml

If the selected area contains only simple cell(s) we can force the editor to copy data as xml data. In the previous example



if we copy data as xml we can review the clipboard content:

```
>XiMpLe<>

  <data UnitID="Speed">
    <Symbol>v</Symbol>
  </data>
  <data UnitID="Temperature">
    <Symbol>T</Symbol>
  </data>
  <data UnitID="Length">
    <Symbol>l</Symbol>
  </data>
```

12 Cut

Cutting combines the copy command and the delete command. Data are copied to the clipboard and then they are deleted.

13 Paste

Data from the clipboard can be pasted to a selected place in a grid. Clipboard data are used always as text that's why these data can be prepared or modified almost anywhere in some text editor or as a result of copy from a spreadsheet editor and so on.

According to the clipboard content and a selecting destination area for data we can distinguish several possibilities of data pasting.

1. *Simple text is in the clipboard* (=there is no sign at the beginning of text indicating the xml structure)

a) A whole row is selected or more rows are selected

Pasting will add new row(s) (one row for any one line of text in the clipboard) and fill the row by column values from left to right. A line of text usually contains several values separated by a tabulator (default separator) or by a separator defined in the general settings [21.1 `General`].

A new row is created above the first selected row with one exception. If the last row is selected (single or as a last row of a block) then a new row is append at the end of a table.

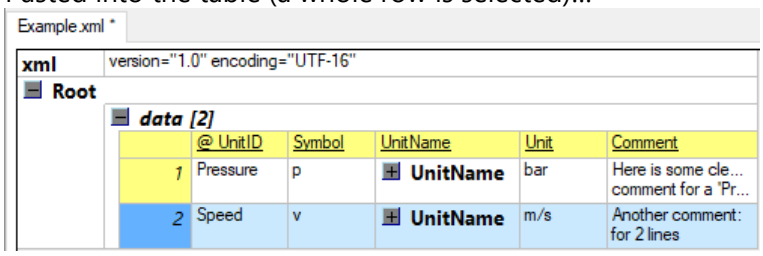
The consequence is if we want to paste data to the last but one row position a workaround must be done – insert a new empty row above the last row then paste data above this new row and then remove it.

Clipboard data:

1 2 3
4 5 6

Example 2

Pasted into the table (a whole row is selected)...



...and the result

Example.xml *

xml	version="1.0" encoding="UTF-16"				
Root					
data [4]					
	@ UnitID	Symbol	UnitName	Unit	Comment
1	Pressure	p	UnitName	bar	Here is some cle... comment for a 'Pr...
2	Speed	v	UnitName	m/s	Another comment: for 2 lines
3	1		2	3	
4	4	5	6		

Cells for which there are not enough data in the clipboard are set to no-value.

b) Selecting area is simple and a whole row is not selected

Pasting will overwrite values in the selected area (if there is enough data in clipboard).

Let's pasting data from *Example 2* to this selected area:

xml	version="1.0" encoding="UTF-16"			
Root				
data [5]				
	@ UnitID	Symbol	UnitName	Unit
1	Pressure	p	Pressure	bar
2	Speed	v	Speed	m/s
3	Temperature	T	Temperature	°C
4	Length	l	Length	mm
5	Volume	V	Volume	cm³

And the result:

xml	version="1.0" encoding="UTF-16"			
Root				
data [5]				
	@ UnitID	Symbol	UnitName	Unit
1	Pressure	p	Pressure	bar
2	Speed	v	Speed	m/s
3	1		2	3
4	4	5	6	mm
5	Volume	V	Volume	cm³

Cells for which there are not enough data in the clipboard are not changed.

c) Selected area is non-simple

Let's pasting data from *Example 2* to this selected area:

xml	version="1.0" encoding="UTF-16"			
Root				
data [5]				
	@ UnitID	Symbol	UnitName	Unit
1	Pressure	p	UnitName	bar
2	Speed	v	UnitName	m/s
3	Temperature	T	UnitName	°C
4	Length	l	UnitName	mm
5	Volume	V	UnitName	cm³

When pasting a warning message appears with a question if we want to continue. We should be careful using this operation because the result strongly depends on how the cells are expanded or collapsed. See the section [11 Copy] for additional information.

Result can be sometimes a little bit confusing:

data [5]

	@ UnitID	Symbol	UnitName	Unit
1	Pressure	p	UnitName	bar
2	Speed	v	UnitName	m/s
3	1		UnitName	3
4	4	5	UnitName	mm
5	Volume	V	UnitName	cm³

In this case the element names were not replaced – the reason is that the element's name can't start with a number. You can try to edit the name of some element and verify that enter a number "2" for element's name is not possible and will invoke an error dialog.

Well, let's try another text in the clipboard – instead of numbers we paste letters:

a b c
d e f

Result should like this (the first impression could be again confusing):

data [5]

	@ UnitID	Symbol	UnitName	Unit	b	f
1	Pressure	p	UnitName	bar		
2	Speed	v	UnitName	m/s		
3	a			c	b	
4	d	e		mm		f
5	Volume	V	UnitName	cm³		

In this case elements were renamed as expected. Changing the element name from "UnitName" to "b" (or "f" respectively) implies an extension of the table. A new columns "b" and "f" are added. The values nested inside the element "b" (or "f") are unaffected because they were not visible during pasting.

Note: For pasting in the point 1b) and 1c) is applied next rule: if the number of rows in the selected area can be divided by number of rows in the clipboard without the rest pasting is repeated again until end of area is reached. Otherwise only the first one pasting is done.

For example if the selected area is 8 rows height and we have copied 2 rows into the clipboard then pasting will be repeated 4 times to fill all selected area.

If the selected area is 7 rows height then only the first two rows of the selected area will be filled.

A special case is to have just 1 row of data in the clipboard. Then all selected rows will be filled by this value during the pasting.

2. *XML text is in the clipboard* (=there is a sign at the beginning of text indicating the xml structure)
   Remark: If the text after the sign is not well-formed XML then an error appears when pasting such data and the action can't be completed until the problem is fixed.

a) A whole row is selected or more rows are selected

Pasting will add new row(s) and paste data defined by XML clipboard data. If some element doesn't exist in the table then a new column name is added to the table. A new row is created before the first selected row with one exception: if the last row is selected (single or as a last row of a block) then a new row is appended at the end of a table.

Let's have these data in clipboard which define two elements named "R" and they contain other elements (Symbol, UnitName, Unit2 and Symbol, UnitName2, Unit).

```
>XiMpLe<>

  <R>
   <Symbol>U</Symbol>
   <UnitName>
    <en-US>Voltage</en-US>
    <de-DE>Voltzahl</de-DE>
   </UnitName>
   <Unit2>V</Unit2>
  </R>
  <R>
   <Symbol>F</Symbol>
   <UnitName2>
    <en-US>Force</en-US>
    <de-DE>Kraft</de-DE>
   </UnitName2>
   <Unit>kg.m/s²</Unit>
  </R>
```

Example 3

Paste this data into this table as new rows...

xml	version="1.0" encoding="UTF-16"			
Root				
data [3]				
	@ UnitID	Symbol	UnitName	Unit
1	Pressure	p	UnitName	bar
2	Speed	v	UnitName	m/s
3	Temperature	T	UnitName	℃

And the result should be:

data [5]						
	@ UnitID	Symbol	UnitName	Unit	Unit2	UnitName2
1	Pressure	p	UnitName	bar		
2	Speed	v	UnitName	m/s		
3	Temperature	T	UnitName	℃		
4		U	UnitName: en-US = Voltage; de-DE = Voltzahl		V	
5		F		kg.m/s²		UnitName2: en-US = Force; de-DE = Kraft

We can see how the data was handled – if some element (Unit2, UnitName2) didn't exist in the original table a new column was created for such element. If some element (UnitName in 2nd row, Unit in the 1st row) didn't exist in clipboard data then no-value is set in a table.

b) A whole row is not selected

Pasting will overwrite values in the selected area.

- In this mode names of elements and attributes are relevant only if we are pasting to the table and for all names there exist the same name in the selected area. In such case data are pasted to appropriate positions and the order in xml data is not important. This mode is used mainly if we are copy & paste data in the same table and over the same columns.
- Otherwise (=if there exists at least one element or attribute name in clipboard data for which doesn't exist a column with the same name in the *selected* area) names of elements are not relevant and data are stored one by one to target cells in the input order.

Let's paste data from *Example 3* to this selected area

data [5]

	@ UnitID	Symbol	UnitName		Unit
1	Pressure	p	**UnitName**		bar
			en-US	Pressure	
			de-DE	Druck	
2	Speed	v	**UnitName**		m/s
			en-US	Speed	
			de-DE	Geschwi...	
3	Temperature	T	**UnitName**		°C
			en-US	Temper...	
			de-DE	Temper...	
4	Length	l	**UnitName**		mm
			en-US	Length	
			de-DE	Länge	
5	Volume	V	**UnitName**		cm³
			en-US	Volume	
			de-DE	Volumen	

And the result (for instance 'Unit2' doesn't exist in the selected area so data are pasted in the order they are formed in clipboard data):

data [5]

	@ UnitID	Symbol	UnitName	Unit
1	Pressure	p	**UnitName** **en-US** Pressure **de-DE** Druck	bar
2	U	**Symbol** **en-US** Voltage **de-DE** Voltzahl	V	m/s
3	F	**Symbol** **en-US** Force **de-DE** Kraft	kg.m/s²	°C
4	Length	l	**UnitName** **en-US** Length **de-DE** Länge	mm
5	Volume	V	**UnitName** **en-US** Volume **de-DE** Volumen	cm³

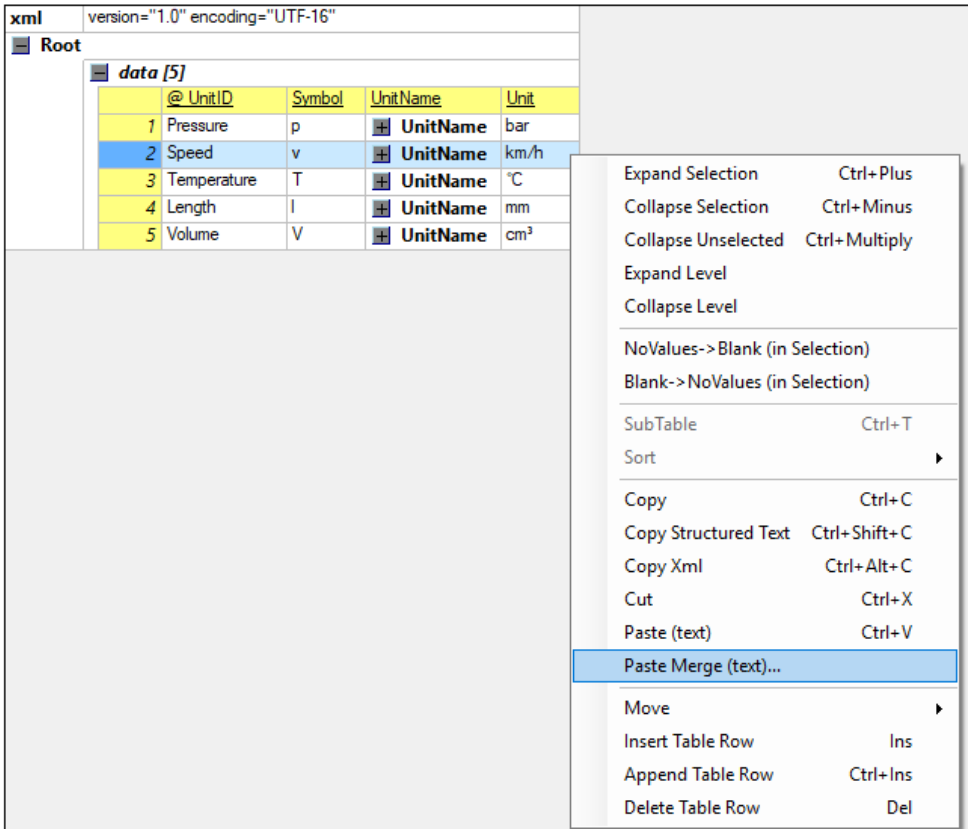
Note: For pasting in the point 2b) is applied this rule: if the number of rows in the selected area can be divided by number of rows in the clipboard without the rest pasting is repeated again until end of area is reached. Otherwise only the first one pasting is done.

For example if the selected area is 4 rows height and we have 2 “rows” in the clipboard then pasting will be repeated 2 times to fill all selected area.

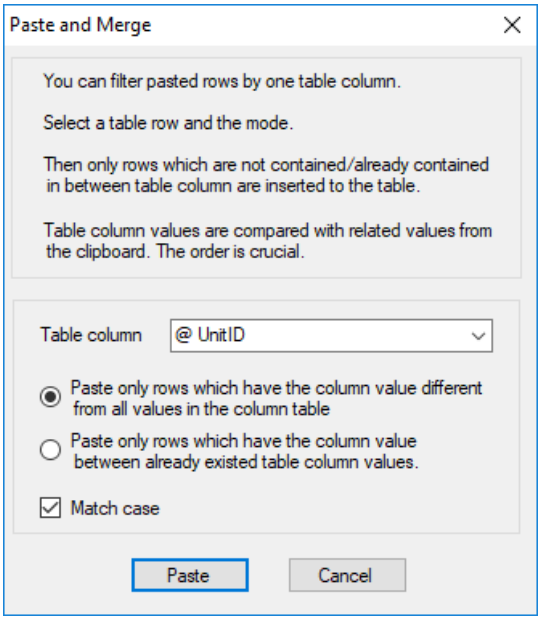
A special case is one cell of xml data in the clipboard (it can be a complex structure which is inside one cell on the top level for example). Then all selected area will be filled by this structure during the pasting. It can be used with an advantage if we want to create a new column and we want to fill it by some nested elements and values which will be the same for all cells in a column. Or if we want to combine it with a sub-table command.

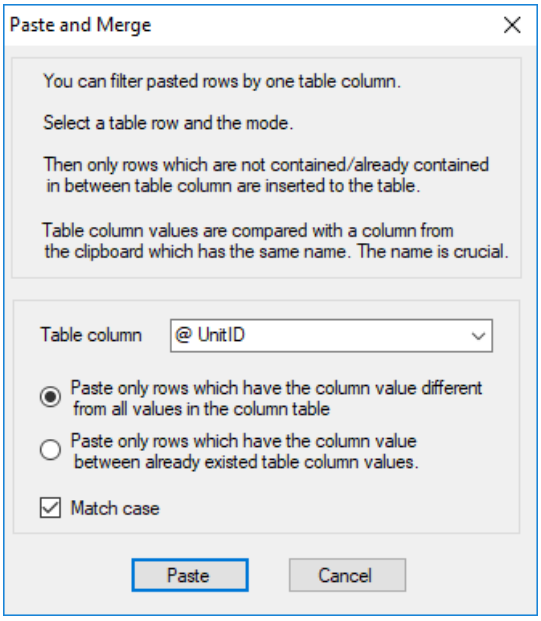
14 Paste Merge...

Sometimes we might need to merge two tables together, but we want to paste only rows which have some new column key value which is not yet contained in the first table. For such case we can use this function which gives us the possibility to select one column as a key and filter pasted data based on this key value. This command is available when we manipulate with the whole table row(s) only.



A new dialog window is opened (there are two slightly different version based on the clipboard content – text or xml)





If we are pasting texts to a table, the order of these values are used to determine the related table column (e.g. if the Table column = "@UnitID" and this column is in the original table the first column then the first values from the clipboard text data are used for comparing. So we must be aware of the right order in the second table.

If we are pasting xml to a table then we can use any order because the program will find the correct column by the column name which is a part of data structure. If the column name (e.g. "@UnitID") doesn't exist in the clipboard data all rows will be inserted.

- *Table column*
  Here is the list of all table columns in the table where we want to paste new data rows. This selection determines the values to be compared with. Only simple column values can be used - it has no effect if we select a column which contains some xml structure (nested element). Usually is possible to extract important values from the structure place them as a new temporary column and then do paste merge.

- *Radio buttons*
  We can select to add all rows which are unique/non-unique in the selected column value. For instance: Let's the table have the column "@UnitID" with two rows and values "Text1" and "Value1". New rows will be added only in case the appropriate column value is different than "Text1" and "Value1" (this is the meaning of the first radio button).
  The second radio button will add new rows to the table if and only if the appropriate column value is equal "Text1" or "Value1".

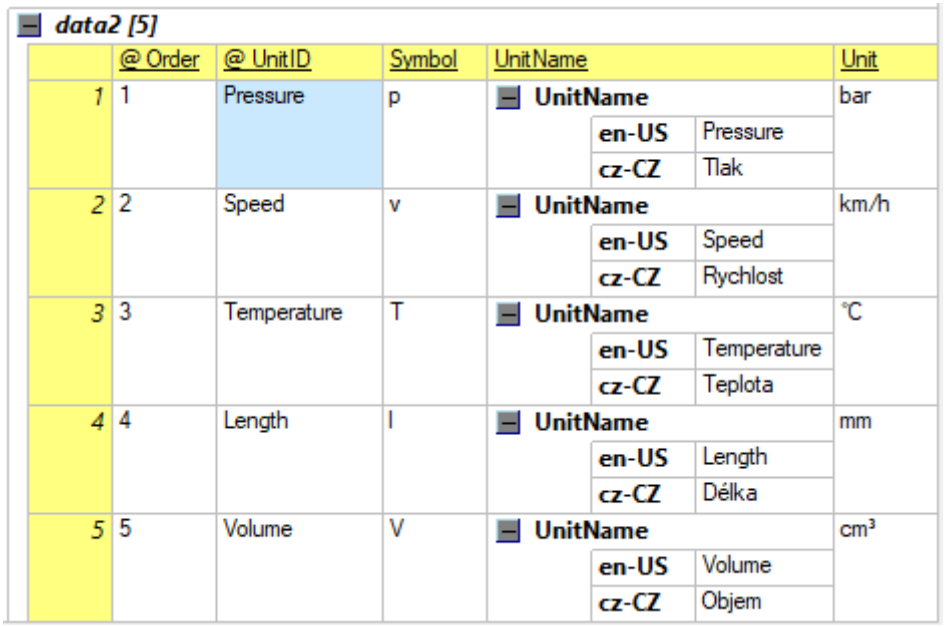
- *Match case*
  Compare the column values case sensitive ("text1" is not equal "Text1") or insensitive ("text1" is equal "Text1")

A remark: This functionality doesn't remove duplicate rows in the clipboard data.

Example:
Let's have two data tables (they can be in different files or both in the same xml file).

data [3]

	@ UnitID	Symbol	Unit	Comment
1	Pressure	p	bar	Comment1
2	Speed	v	m/s	
3	Volume	V	cm³	Comment2



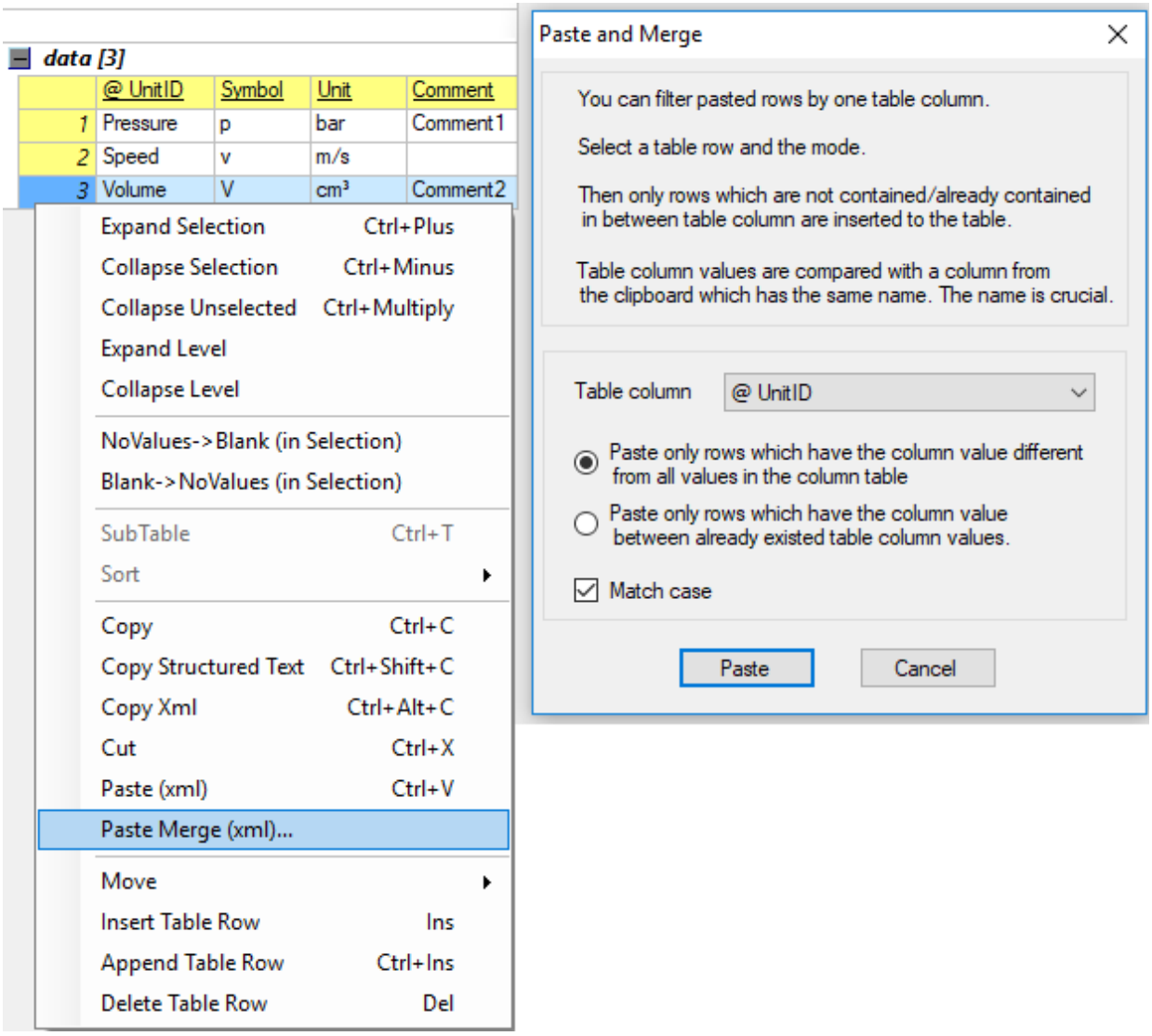
data2 [5]

	@ Order	@ UnitID	Symbol	UnitName	Unit
1	1	Pressure	p	**UnitName** **en-US** Pressure **cz-CZ** Tlak	bar
2	2	Speed	v	**UnitName** **en-US** Speed **cz-CZ** Rychlost	km/h
3	3	Temperature	T	**UnitName** **en-US** Temperature **cz-CZ** Teplota	°C
4	4	Length	l	**UnitName** **en-US** Length **cz-CZ** Délka	mm
5	5	Volume	V	**UnitName** **en-US** Volume **cz-CZ** Objem	cm^3

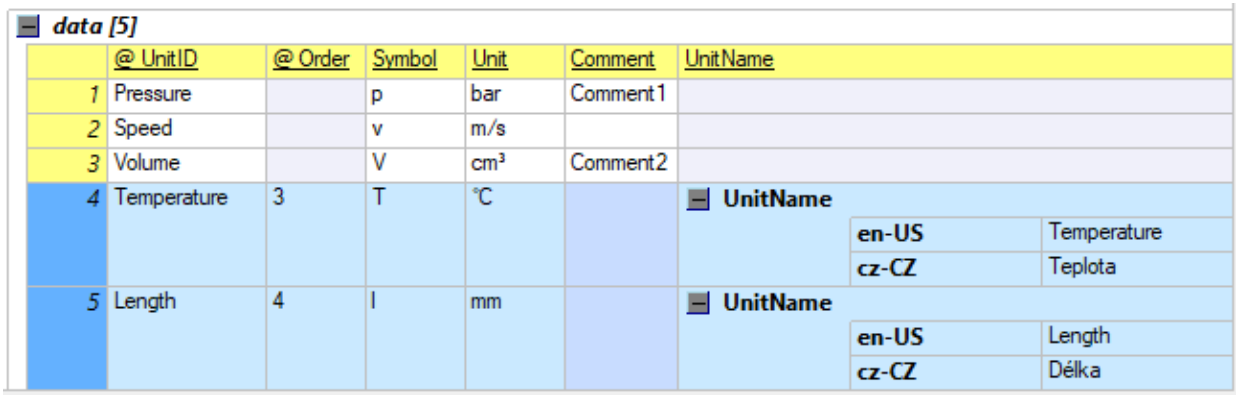
Now we would like to add all rows from the table “data2” to the table “data” except the rows which has the “@UnitID” value already in the first table (in fact for this case we want to add only rows where “@UnitID” = “Temperature” or “Length”).

Firstly, we select all rows of a table “data2” and copy them to the clipboard. There are some structured elements so the xml format will be used. That is important for our case because tables have different columns and different order of columns and we need to pair same columns in the result. If there are only simple elements we can force (by a command “Copy Xml”) to copy table data as xml structure.

Then we will call a command “Paste Merge...” from the first table “data”. It will open a new dialog...



Using the appropriate settings, we will get the result:



You can notice that the columns “@Order” and “UnitName” which were not in the original table were added to the table during the pasting as well.

15 Table Heading

15.1 Copy / Paste Column Headings

When we select one or more table columns (or a whole table by clicking to the top left corner) we can copy names of selected columns (headings) to the clipboard or reversely paste from the clipboard new names for table columns. In the context menu appears two commands: copy column headings and paste column headings.

Column names are xml elements and there are some restrictions for characters we can use in the names. For instance, we can't use spaces or brackets. For these reasons when we are pasting new column names from the clipboard XiMpLe is checking these names and all characters which are not allowed are replaced by the underscore character '_'. If some name is repeating then an index is added to the end of the name (e.g. "value", "value1", "value2").

15.2 Freeze / Unfreeze

For a table it's possible to freeze a table heading to have a better overview if table is exceeding one page. In such case a table heading remains on top of the view. Only one table heading can be frozen at the same moment. The freeze command removes any other previously used freeze command.

The command can be found in a context menu for any cell, but it's disabled if the cell is not inside a table. Please note that for the selected cell the nearest table heading is frozen.

16 Insert and Append

We have already seen that an element has its name and its value which can be changed.
An element can contain another elements and/or attributes. When a single element is selected we can add a new element inside this element or add a new attribute to this element.

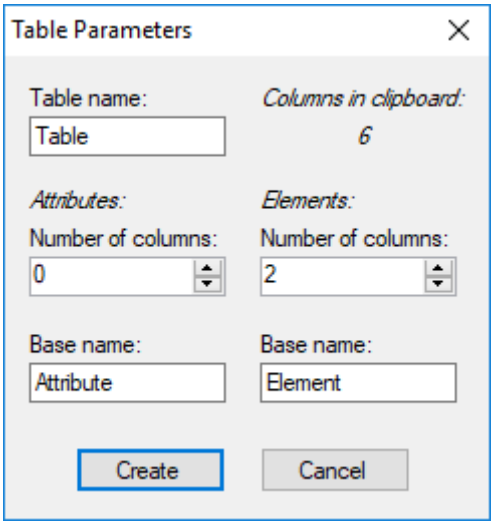
We have a possibility to insert a new element ahead of the selected element or append a new element behind the selected element.

For attributes we have similar options – insert or append an attribute.

There could be some other objects which can be inserted or appended (currently a comment) but the functionality is always the same.

16.1 Insert / Append Table

A special option which deserves a separate comment is the function for adding a table. Generally, we are able to create a new table by adding elements only but tables are so frequently used that there is a special command available. When we use this command (we can find it in the context menu only) a dialog appears and we can specify parameters of the table:



- *Table name*

  The name of the table - in fact this name will be used for table row elements.

- *Number of columns*

  This is the number of columns used for attributes (the first number) and for elements (the second number). As an additional hint there is information about number of columns formed by current clipboard data in the top right corner. This can be used when we copy some data, and we don't know exactly how many columns we need for a new table.

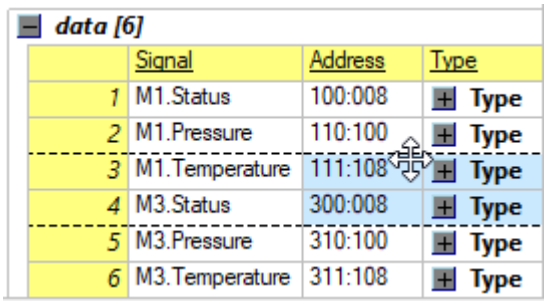
- *Base name*

  Column names are generated from the base name by adding ordinal numbers as index. We can overwrite the table names later by pasting new names from clipboard or by editing them.

A newly created table will have two empty rows.

17 Drag & drop

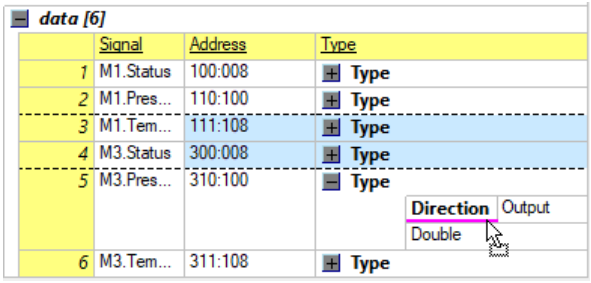
It's possible to move or copy (if holding Control key) the selected area to a new place in the xml structure by a drag and drop functionality. Moving a mouse pointer near the border of the selected area, a cursor for moving into four directions appears and indicates a possibility to begin the drag and drop function.

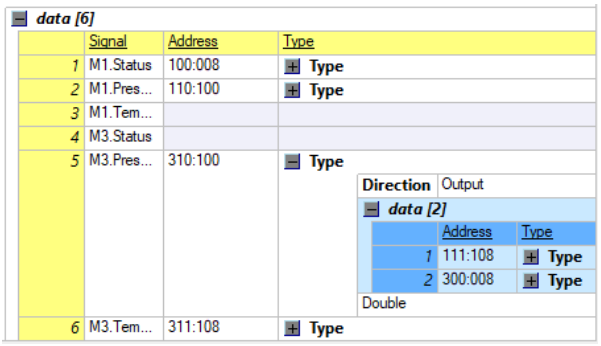


By drag and drop is possible to exchange attributes or elements, moving table rows or moving a column in the table, copy selected area and so on.

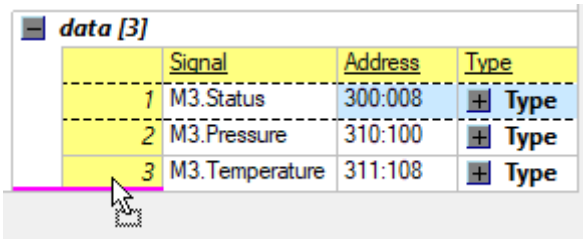
Some actions are not supported and in such case a cursor for drop is changed to not allow icon. For instance moving selected area into itself (only a copy inside itself is possible), moving element to the attribute etc.

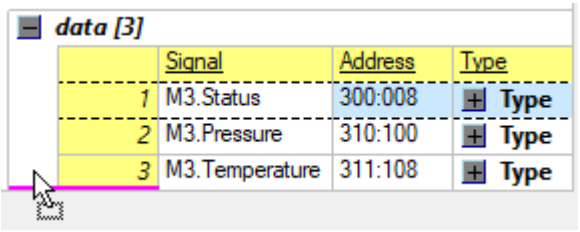
While holding left mouse button we can select the target of the drop which is indicating graphically.





Please note that if we want to select last row of the table as a target we must point by the cursor at the last row inside the row heading area of the table. If we want to target a whole table we must stay with a cursor on the last row of the table outside the row heading.





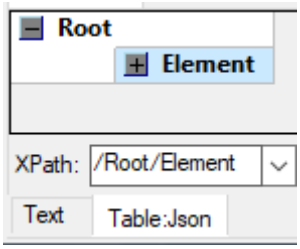
In the left picture we will move selected area to form a new last row, in the right picture we move elements next to a table and add new elements Address and Type behind the table.

The drag and drop action can be interrupted by the Escape key.

18 JSON files

From version 1.4 editor supports JSON files – they can be loaded and edited in a table view with all benefits which are available for XML files. For errors in JSON file the similar behaviour like for XML file is applied – it switches to text editor and a message appears indicating the position in the file where the error was found.
The file can be anytime switched from table view to text view and check the json source. In fact, the json source is converted to xml format but XiMpLe will preserve whitespaces of the original file and if you switch back you should see no differences.

Json format is indicated by text “Table:Json” on the bottom tab (see picture).



If you copy some complex structure in the table's view the internal xml structure is transformed back to json and in the clipboard you will see json structure (see also section [11 Copy]).

There is no explicit way how to convert json file to xml file or vice versa. If you need such conversion you have to select and copy from one format and paste it to another.
For instance you can select the root element in json table, copy it and then create a new xml file and overwrite root element.

18.1 Special attributes

To be able to protect all information from the json file the convertor (to xml) uses for some cases special attributes which contain some information which can't be stored in the xml structure naturally.

There are three special attributes which can appear in the xml structure:

- "json.name.--"

  This attribute is used for json names which are not valid as xml element names. The original name is converted to xml valid name (the underscores and a suffix index are used to generate the valid name) and the original name is stored as a value to this special attribute. When it's converted back from xml to json information from this attribute is reused.

- "json.array.--"

  This attribute indicates that the alone standing element builds an array. A typical example could be this json fragment:

  ```
  "single array": [
     { .. here can be some nested complex structure .. }
  ]
  ```

  The "single array" will be represented with an xml element but the information about array must be stored separately. In this case the value of attribute is irrelevant and important is only the presence of the attribute of this name.

- "json.empty.--"

  This attribute is used to distinguish between empty string value and empty object as a value. For instance: *"Block" : { }* versus *"Block" : ""*

  Also, for this attribute the attribute's value is not used and just the existence of this attribute is tested.

By this mechanism the json can be converted to xml and back without losing information.

But be aware that opposite conversion can lose information. If we convert xml to json and back we don't have to get the same xml document. All xml attributes will change to elements prefixed by '**@**', all comments, style-sheets directives and others non-elements will be lost.

18.2 JSON values

JSON structure is quite simple and straightforward but there is one important thing which must be cared about. Values in json can be written as text values in quotation marks or without quotation marks if the value is a number or a special constant. There are three constants: false, true, null.

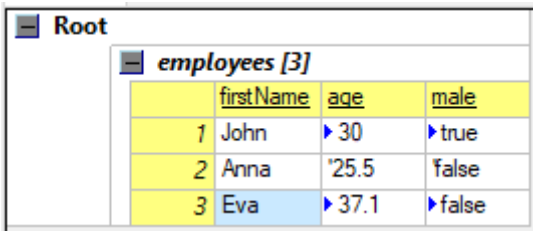
Examples of text json values:
{ “name” : “Adam” }, { “age” : “33” }, { “married” : “yes” }, { “male” : “true” }

Examples of numbers or constants:
{ “age” : 33 }, { “married” : false }, { “male” : true }

As we can see, numbers and constants can be used by both ways - as a text and also as a number. When XiMpLe parses the json file, the numbers and constants which are used as texts are prefixed by an apostrophe character (e.g. ’23 or ‘false). It’s a similar like in Excel if you want to use a number as a text.

The consequence is that if some text value starts with an apostrophe it will be prefixed by another one apostrophe. When they are converting back all values starting with an apostrophe is taken as texts and the leading apostrophe is removed.

In tables the numbers and constants are prefixed by a small triangle in front of the value to get a better orientation in the structure.



Numbers can be converted to texts and vice versa (see section [10.1] for details). Numbers can be also specified in the search function (see section [20.3 `Find and Replace`]).

19 File Menu

19.1 New

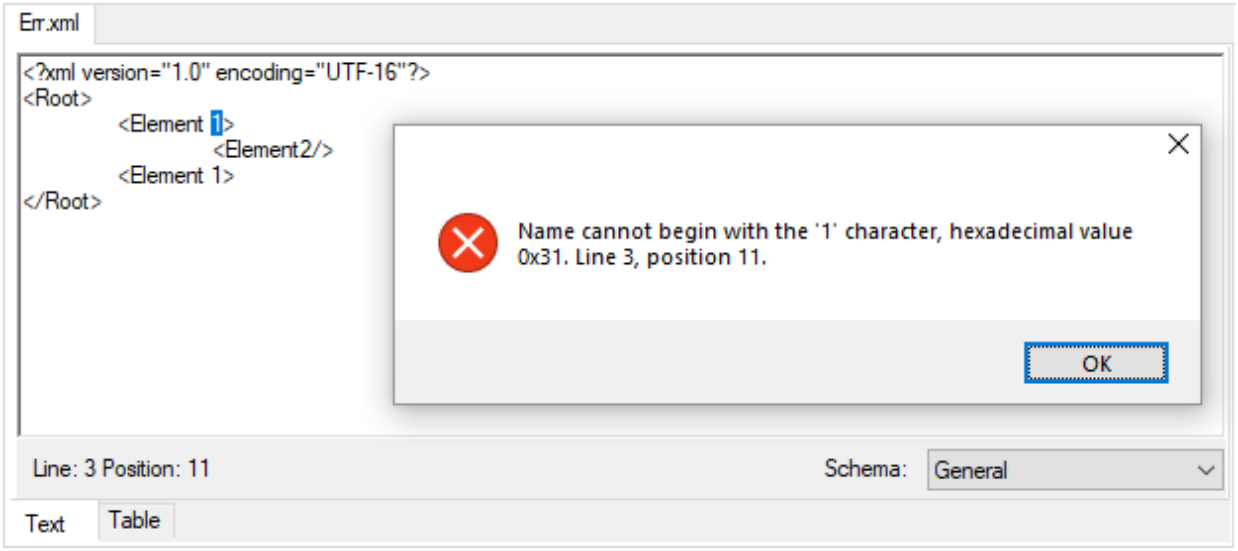
Create a new empty xml file which contains xml declaration and the root element with one child. A user can select a file encoding for XML file (utf-8, utf-16 or utf-32). Additionally, it’s possible to create a new empty text file (utf-8) or a new JSON file with one child element (utf-8).

19.2 Open...

Open a file from a selected folder. Another possibility to open a file is to drag and drop a file to the editor. It’s possible to select for open or drag and drop multiple files. In such case all these files will be gradually opened. The process can be interrupted by pressing Escape key.

A well-formed XML document or a correct JSON document is expected. It’s possible to have several files opened.

If a file is not a well-formed XML (or JSON) document then it can’t be displayed in the table editor and the file is opened in the internal text editor instead. An error message appears (there are some exceptions when error message doesn’t appear – see below) describing the place of the error (a line and a position) and a short info about the error. A cursor position is set to the indicating place. Then we have a possibility to repair the issue.



There are three exceptions when no error appears, and the text editor is opened instead:

1. If the file is evaluated as non-xml file, then text editor opens directly. If we want to know the reason why it can't be edited as an xml (or json) just try to change the text editor to table editor tab on the bottom of the window to see the error message.
2. The file is too big. Currently the limit is approximately 100MB. If the file is bigger then it's loaded into the text editor directly.
3. If the current file is displayed in the text editor, the general option 'Hide text tab if xml is well-formed' is not active and some other file is opened by drag & drop (doesn't matter if it's valid xml or not) then a text editor will be opened directly instead of an xml editor. If the file is correct xml file just switch the tab back to the xml editor.
   The reason for this behaviour is to be able to use XiMpLe also as a text editor without additional clicking or error handling.

   Note: This rule is not valid when the editor is in the comparison mode.

See also [22 `Text Editor`].

19.3 Open recent file

Select a file from the list of lastly opened files. Items in the list can be removed by clicking the cross icon "x" on the right side. On the left side there is an icon '*' to add the item to the favourite list on top.

19.4 Open favourite file

Select a file from the list of favourite files to open it. Any file from the recent file list is possible to add to the favourite list by clicking the stars icon '*' on the left side. Items in the list can be removed by clicking the cross icon "x" on the right side.
List of favourite files is should be managed by the user if there are already too many items.

19.5 Close

Close a currently selected opened file. If the file was modified a user is asking for saving.

19.6 Close all

Close all opened files. For modified files a user is asking for saving.

19.7 Save

Save the currently selected opened file.

19.8 Save as...

Save the currently selected opened file to a new destination and/or with a new name.

19.9 Save all

Save all opened files.

19.10 Run script...

Open a dialog for selecting the script file, which is then executed. This option is available only for registered users. See also [23.6 `Script mode`].

19.11 Start / Stop recording script

By this command we can record our actions and create a new script. When the recording is stopped (be aware that it can't be done while any sub-table is opened) the new text file is created and the recorded script is put inside. Such script can be run (see also [23.6 `Script mode`]) and with the original file it should mimic your actions. The main support is in the standard table's view actions; in the text editor the support is currently only basic.

The idea is to re-use such script on other files with the same structure - see also [23.6 `Script mode` - batching]. In such case some modifications of the generated script is necessary:

- We should care for all Open() commands and have on mind that for batching we need to open always different file. Also is a good practice to close the file at the end if it's not required for other actions.
- Some actions can be improved. For instance if we insert a new element and then set the new name for such element, automatic generation will create a sequence: select cell, insert new element, select this element and set the new value for that element.
  A better solution for a human would be to get the created cell from the Insert() function and then set the new name of that cell directly without any other selection.

The point is that if we want to use the script more generally for more files, it will need some additional changes from our side. The recording script is only the pattern to help us to start.

Note: No actions are recorded while the script is running.

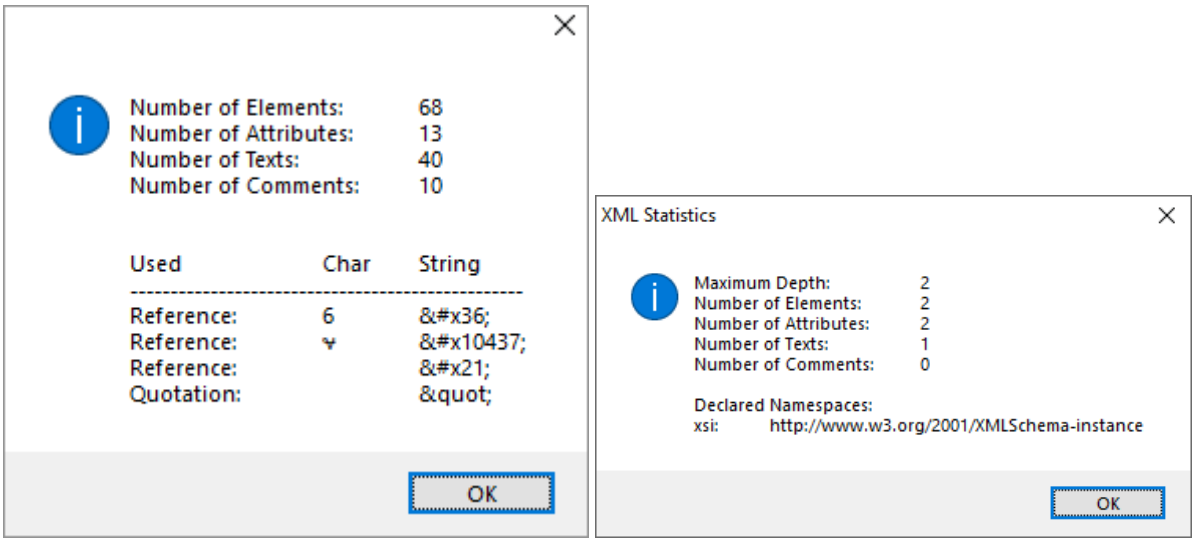
19.12 Open Containing Folder

For the currently opened file it opens external windows explorer with the folder where the file is located and select him in the list.

19.13 XML Information

Short information about the currently selected file: Number of elements, attributes, texts and comments and information about declared entities and special characters used in the file. For additional details please check the section [21.2].

This information is read when the file is loaded and it's are related to this state. So adding an element in an editor doesn't affect this information.
From version 1.5 there is information about declared namespaces. This information is updated while editing and might be helpful in [23.4 `Validation Mode`].



In the example on the left side we can see that in a file is used character '6' also by a reference "6". An exclamation mark character is used only as a reference "!". The same is true for a quotation character which is always used as """ in this example.
The example on the right side demonstrates declared namespace in the file – in this case it's a namespace xsi.

19.14 Exit

Exit the program. If there is some modified and not saved file(s) a user is asking for saving.

20 Edit Menu

20.1 Undo

Get back the state before the last operation. Undo can be done repeatedly. Note that the undo function can be disabled or restricted to some maximum number of remembered undo states (see [21.1]). If you disable undo functions you can increase a performance a little bit. Undo disabling or restricting can decrease an amount of used memory.

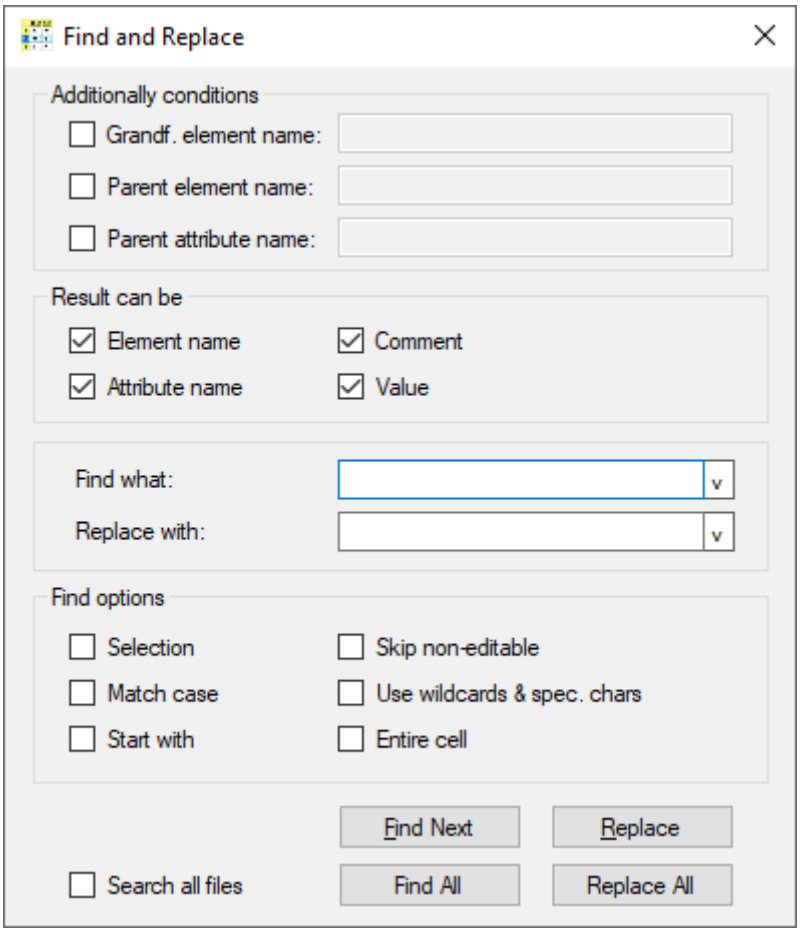
Note: Undo doesn't remember the information about expanding or collapsing states of elements which don't form a table.

20.2 Redo

Revert back the state after the last undo. Redo can be done repeatedly for each previous undo.
Note: Redo doesn't remember the information about expanding or collapsing states of elements which don't form a table.

20.3 Find and Replace

Find and/or replace some value or a name.



For searching can be used wildcards characters.

There are five wildcards (and one reserved character '\') available:

- Question mark '**?**' represents any but just one character (e.g. "a?c" will find "abc" but not "ac" or "abbc")
- Asterisk '*****' can be used for any group of characters including empty set and a new line character (e.g. "a*c" will find "ac", "abc", "abbc", "ab123c")
- Percentage '%' can be used for any non-empty (at least one character) group of characters (a new line character is excluded which means that a search will not exceed the current line); when used for a replacing this character has a special meaning – replacing by a lookup table (see [20.3.2])
- Number sign '**#**' can be used for a numeral or a group of numerals – it will always get all numerals in the group (e.g. "a#b" will find "a1c", "a120c" but not "ac" or "a1.2c" or "a-2c"). The exception is joining more number signs together – in such case the all but the last one represents one numeral (e.g. number 12345 searched by '###' will be find and the first '#' is '1', second '#' is '2' and the third '#' is the rest of the group: '345').
  Moreover the numbers found by this wildcard can be additionally modified (see [20.3.3] for details).

- Commercial At '**@**' can be used for any white space (a space character, a tabulator character, a hard space) or a group of white spaces but at least one (e.g. "a@c" will find "a c" or "a c")

Wildcards can be combined arbitrary.
A percentage character '%' has a special meaning when used in a replace field (see [20.3.2]).

If we want to use some of the reserved characters ? * % # @ \ as normal characters (in case the option "Use wildcards & spec. chars" is checked) we must prefix it by a backslash character '\'. For instance if we want to find an asterisk we need to enter "*".

Additionally we can use a special string to represent a tabulator: '\t', a line feed: '\n' and a carriage return: '\r'. These strings are valid only if wildcards are allowed.
If we want to use them for finding and/or replacing "Use Wildcards & spec. chars" option must be checked.

Main fields are:

- *Find what*

  A string used for a finding. Wildcards can be used in a combination with a "Find options/Use wildcards & spec. chars" option. Finding starts from the current position and when the end of a file is reached it starts from the beginning until the starting position is reached again (a full cycle is done).

- *Replace with*

  A string used for a replacing. Wildcards can be used in a combination with the "Use wildcards & spec. chars" option only.

  Wildcards for *Find* and *Replace* fields can be used arbitrary and in any order. These rules will be applied:

  - If some wildcard used in *Replace* field is not specified in *Search* field then an empty string will be used instead

  - If more wildcards of the same type are used in *Replace* field than in *Search* field the lastly used string will be used again

  - Example: Let's "*Find what*" = "*o ??" and "*Replace with*" = "?* *#".
    Then applied on the string "Hello world" the result will be: "wHell Hell"
    (explanation: '*' = "Hell", the first '?' = "w", the second '?' = "o";
    the second '?' is not used in the replace, the second '*' in replace will repeat the value for the first '*' because in *Find* field it's defined only once, the '#' will be replaced by empty string because it's not used in searching pattern)

A command "Find Next" is going through the file (from the current position) and tries to find a string entered in the field "Find what". There can be specified additional conditions as well.

For finding (and replacing) can be used these options and input fields:

- *Grandfather element name*

  The name of a grandfather element can be specified in this field (when a related checkbox is checked). In such case an additional condition is added which requires that a found object has the parent's parent element of the name which was entered in this option.

  Wildcards can be used for this field but the name must fit exactly. For example if we want to specify a parent element name "Data17" we can use strings like "Data17", "Data??", "Da*", "Data#", "Data1#" but not only "Data".

- *Parent element name*

  The name of an element can be specified in this field and then an additional condition is added for a parent element name. It means that only objects which have the parent element of the specified name can be found. Wildcards can be used for this field but the name must fit exactly.

- *Parent attribute name*

  The name of an attribute can be specified in this field and then an additional condition is added. In this case only the values of a specified attribute can be found. Wildcards can be used for this field but the name must fit exactly.

We can restrict results by settings in the group "Result can be" as well:

- *Element name*

  The result can be an element name. If it's not checked then element names are skipped.

- *Attribute name*

  The result can be an attribute name. If it's not checked then attribute names are skipped.

- *Comment*

  The result can be a comment. If it's not checked then comments are excluded.

- *Value*

  The result can be a value. If it's not checked then values are skipped.

- *Json text*

  The result can be json text value (it can be perceived as an xml value which is stored in json as text). If it's not checked json text values are skipped. If *Value* checkbox is unchecked this option is disabled. See also section [18.2 `JSON values`].

- *Json number/constant*

  The result can be json number or a constant (true, false, null) value (it can be perceived as an xml value which is stored in json as a number). If it's not checked json number or constant values are skipped. If *Value* checkbox is unchecked this option is disabled. See also section [18.2 `JSON values`].

Find options:

- *Selection*

  Only in the selecting area the searching or replacing is done.

- *Match case*

  Searching will be case sensitive if this option is checked.

- *Start with*

  Text must start with the entered text.

- *Skip non-editable*

  Some cells can't be edited at all. For example comment element name (#comment) or xml declaration. We can exclude these non-editable cells from the searching.

- *Use wildcards & spec. chars*

  Enable or disable the possibility to use wildcards and special characters for the searching and replacing.

- *Entire cell*

  Entered text must match the whole cell.

Special option:

- *Search all files*

  This option is related to commands '*Find All*' and '*Replace All*' and can expand the area of activity from current file to all other opened files which are in table's view mode. Files opened in text mode are excluded even they are valid xml files.
  If the lookup table is defined and used, then also the currently used lookup table file will be skipped and excluded from searching.
  If we are searching in a subtable, this option is hidden and can't be used.

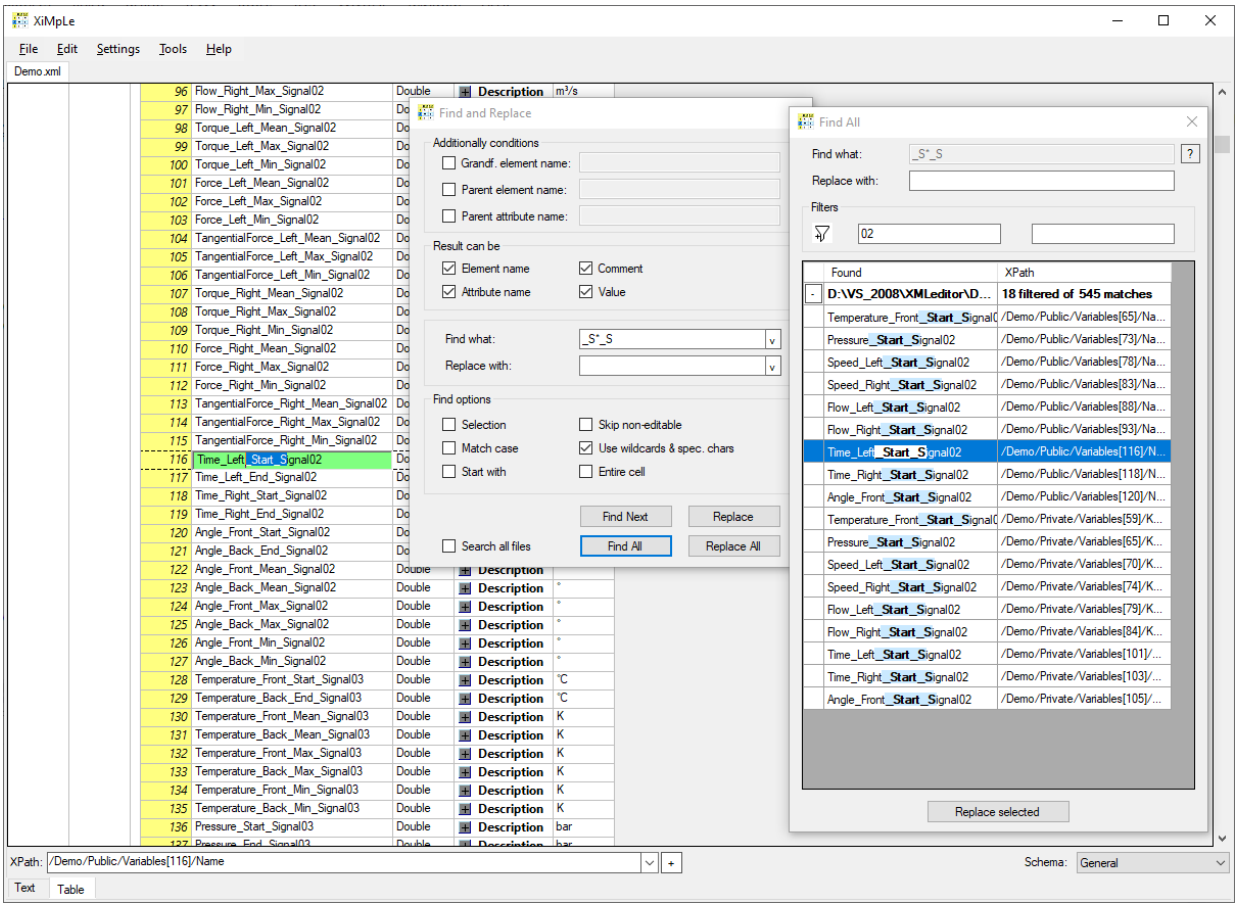
20.3.1 Find all

By this command we can collect searching results from one or more files and display them in a separate dialog, which is opened. The list contains found texts and the XPathes of the results.

Each file has its own records, which can be collapsed or expand in the list.

By selecting a row in the list, the appropriate cell is selected and a found text is highlighted.

An example of such dialog we can see in the next picture.



Here we have used the Demo.xml file with the 'Find All' using a wildcards pattern "_S*_S". Then in the dialog we have used a filter ("02") to display only a subset of results from all found records.

If we select some row of the table, the appropriate cell is selected as we can see on the left side of the picture.

Results can be filtered by a found text (left column) or/and by an XPath (right column). We can enter text which must be contained in the results.

It's possible to select multiple rows of the table (with a Shift or Control key, they can form a non-continuous area). By a button '*Replace selected*' (or by a key 'R') the replacing can be applied on the selected records. The found texts will be then replaced by a new string which is defined in the '*Replace with*' text box in this dialog.

It's possible to select all lines by a key shortcut Ctrl+A and copy selected rows to clipboard by a key shortcut Ctrl+C.

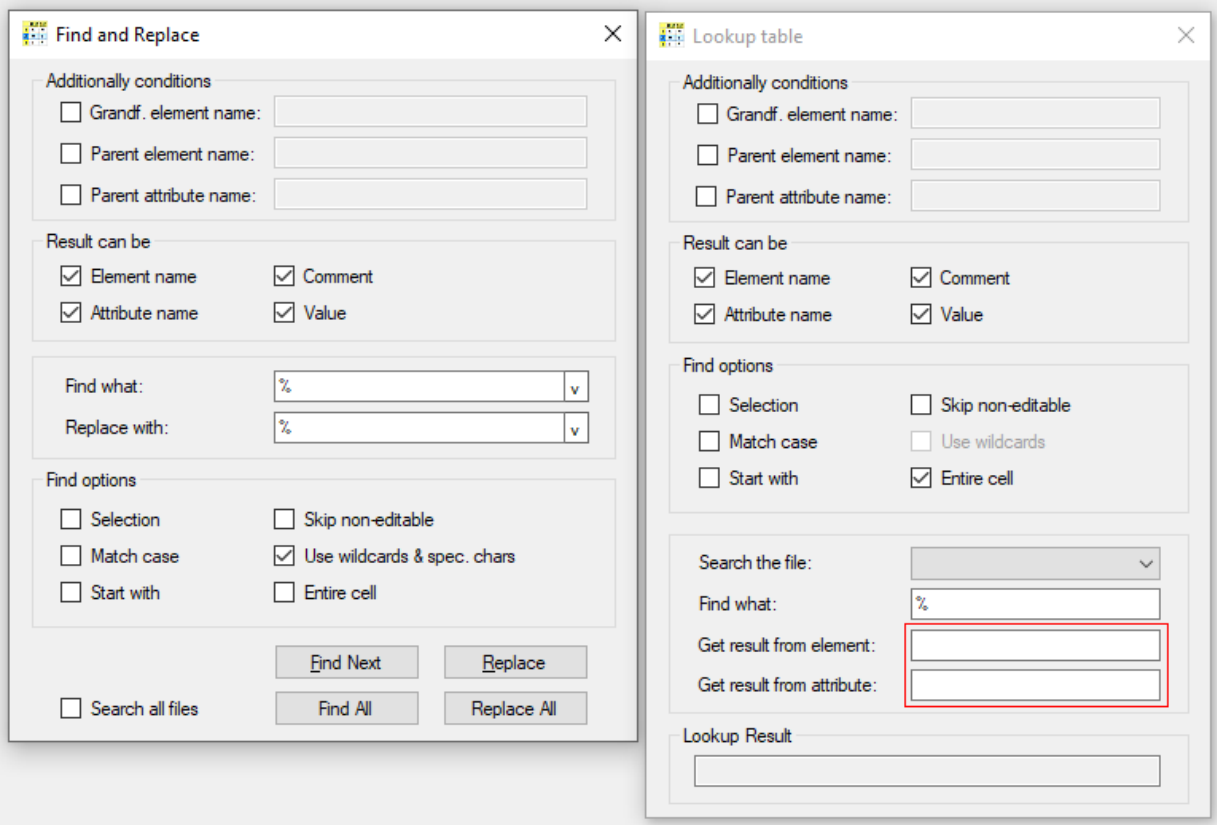
- Note: In this dialog it is irrelevant what is entered later in the parent 'Find and Replace' dialog. The options are taken at the moment of opening the 'Find All' window. Used search options can be checked by moving mouse cursor over the question mark in the top right corner. Also text for replace is taken from 'Find All' window and not from 'Find and Replace' one.
- Important: While this dialog is opened, every change we do in the file(s) is reflected and updated in the dialog. It will take some time to update results and for longer file(s) it might be time consuming. So always consider if it is worth to have the dialog opened permanently or only on a request.

- Exception: If the dialog is opened with the option 'Selection', which requires to search only selected area, the question mark in the top right corner is displayed with an exclamation mark "?!" to highlight this fact. If we do any change in the file which changes the selected area, instead of dialog results update, the dialog is closed because the original area is no longer valid. That means we should do changes from the 'Find All' dialog in this case, because changes done by a 'Replace selected' button shouldn't change selected area.

20.3.2 Replacing using a lookup table

For replacing there exists a special possibility to use a lookup table. This table can be defined anywhere in an external xml file. The table should have at least two columns – one column contains values to find, and the second column contains related values to obtain. It doesn't matter if these columns are elements or attributes. The value is found in the table and a new related value is taken as a result.

When we use a character '%' in the replace edit box as a wildcard then a second dialog appears, and we can specify some details for a lookup table. Some fields are required.



The most of options have been already described in the previous chapter. Only new options are mentioned here.

- *Search the file:*

  In a combo box we can select an xml file we want to use for a lookup table. Only valid xml files already loaded in XiMpLe are shown in the list.

- *Find what:*

Usually there is only a character '%' in an edit box which is related to the character '%' in the replace edit box in a find dialog window. If the percentage character is missing, then it is added automatically at the end of a string.

- *Get result from element:*

  It defines element's name from which the result will be taken. The name is case sensitive. See also example [20.3.2.1]
- *Get result from attribute:*

  It defines attribute's name from which the result will be taken. The name is case sensitive. See also example [20.3.2.1]

One of these two names must be defined if we want to use this feature.

- *Lookup Result*

  It displays a result of finding in a lookup table. If no record is found a background's field is coloured slightly to red. In such case the 'Replace' command doesn't change the cell value.
The 'Replace all' command replaces only values which are found in the lookup table. Others are skipped and not replaced.

This functionality requires several necessary conditions to be met:

1) 'Search the file' field must be defined
2) 'Get result from element' or 'Get result from attribute' field must be defined

Note: We have a possibility to leave Lookup table in "undefined" state and use the wildcard '%' in a similar way like a wildcard '*'. In such case replacement will be done without lookup table. The benefit from this action could be the switching of texts because we can distinguish between two groups of characters. For instance, searching "(*, %)" and replacing "(%, *)" will change the text "(hello, world)" to "(world, hello)".

20.3.2.1 Example

The best way to demonstrate Lookup table functionality would be an example. Let's have these two xml files:

Example.xml | ExampleID.xml

xml — version="1.0" encoding="UTF-16"

Root

data [5]

	@ UnitID	Symbol	Unit	Comment
1	Pressure	p	bar	id=0001
2	Speed	v	m/s	id=0002
3	Temperature	T	°C	id=0004
4	Length	l	mm	id=0006
5	Volume	V	cm³	id=0005

Example.xml | ExampleID.xml

xml — version="1.0" encoding="UTF-16"

Root

key [5]

	@ ID	Value
1	0001	Stupid comment
2	0002	Needless comment
3	0003	Normal comment
4	0004	Clever comment
5	0005	Unbelievable comment

The first one file 'Example.xml' contains values in the 'Comment' column which we want to replace by values defined in a lookup table in the second file 'ExampleID.xml'.

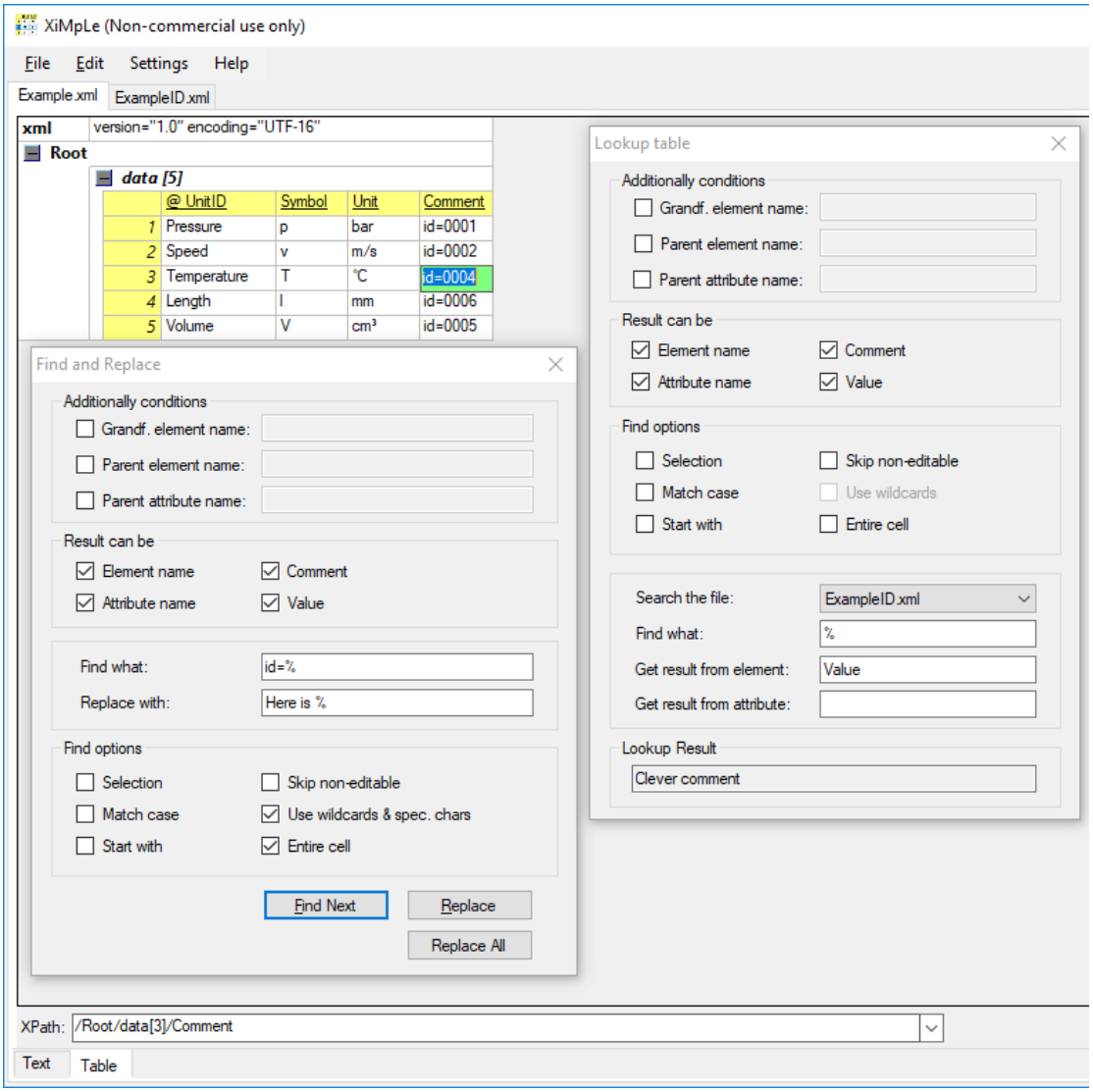
Firstly, we need to find a text 'id=%' (please notice a checkbox 'Entire cell' which expands the wildcard '%' to a whole cell on the next picture). The first record found is the text 'id=0001'. And the character '%' corresponds to the string '0001'.

The substitute string for the '%' character is then transferred to the lookup table (a file 'ExampleID.xml') and the file is searched using criteria defined in the Lookup table dialog. The text '0001' should be found.

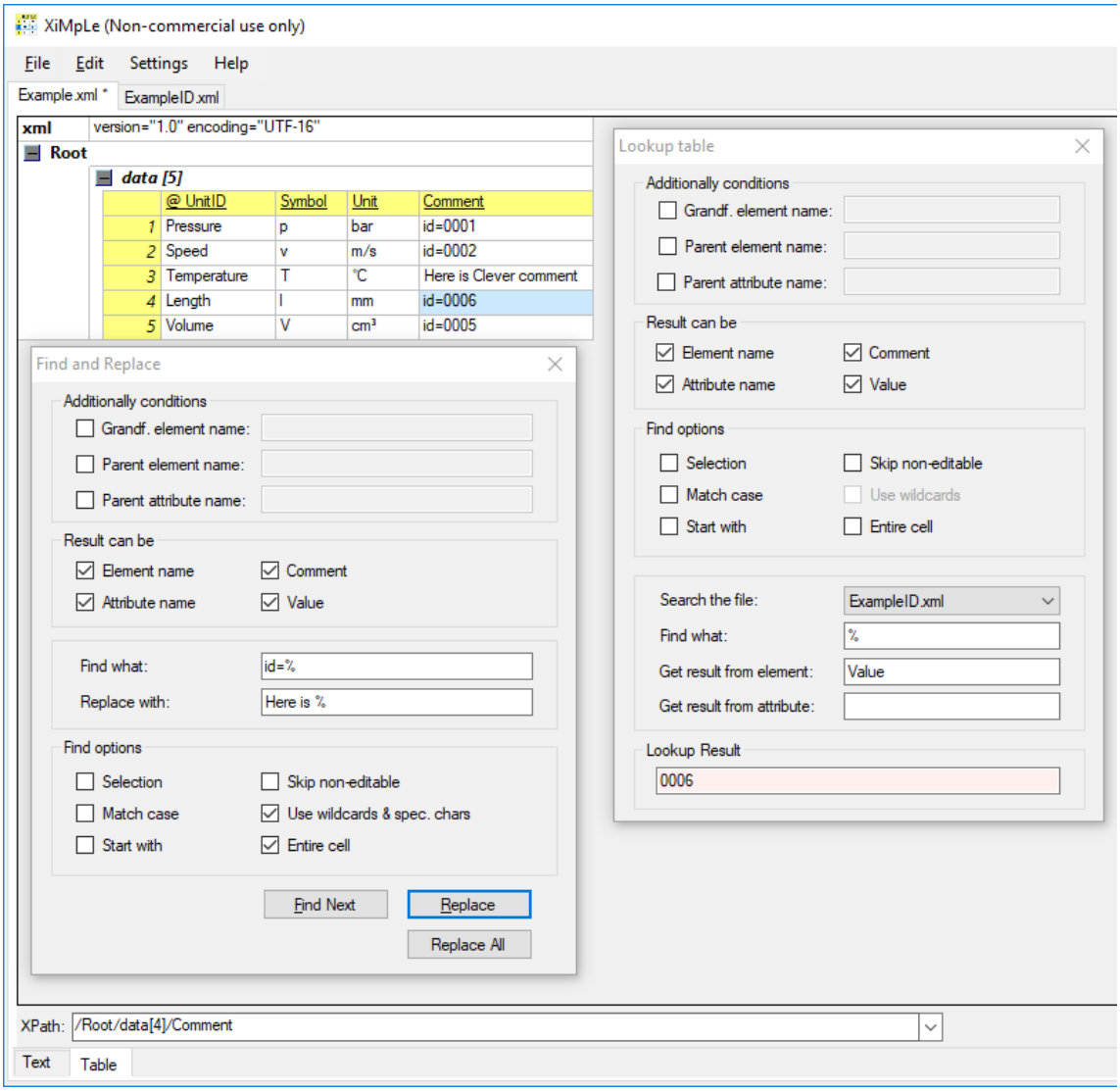
Remark: If there are more records containing a text '0001' in the file 'ExampleID.xml' we have to specify more precisely the parent element for instance to get the correct cell.

If the text '0001' is found, then the result is taken from the column which name is defined by the 'Get result from element' string. In this example the column name is 'Value' (see a picture below) and the result string is 'Stupid comment'. For text '0002' it will be 'Needless comment' and so on.

On the next picture we can see the situation for the string 'id=0004'.



Now '%' = '0004' which gives the result string 'Clever comment' from the lookup table. After 'Replace' command we can observe the result on the next picture.



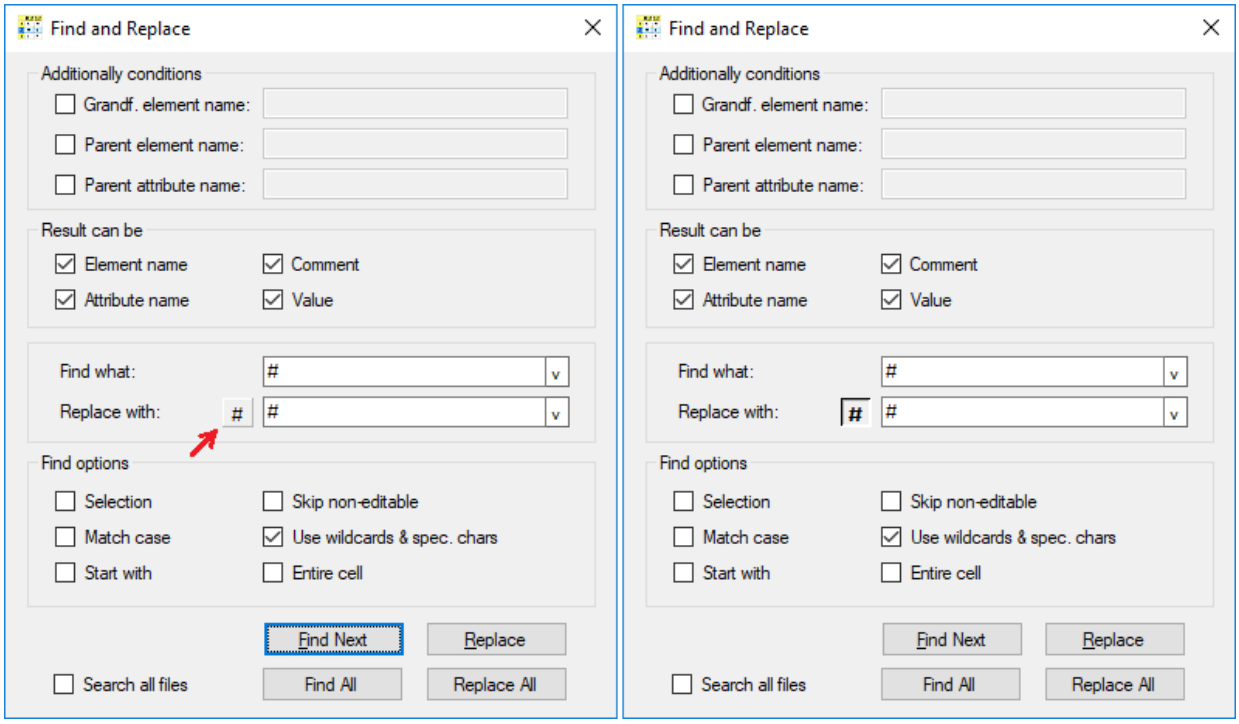
The string 'id=0004' has been replaced by the text 'Here is %' = 'Here is Clever comment'. The character '%' has been substituted from the original value '0004' by a lookup table to the new value 'Clever comment'.

Then the next found string is '0006' but this string is not in the lookup table - that's why we can see a slightly red background on the result field and the value '0006' which corresponds to the searched value '%' which hasn't been found.

20.3.3 Replacing using number modifiers

For a wildcard '#' which finds a group of digits (in fact all these digits are forming a non-negative integer) can be additionally used a special offset modification.

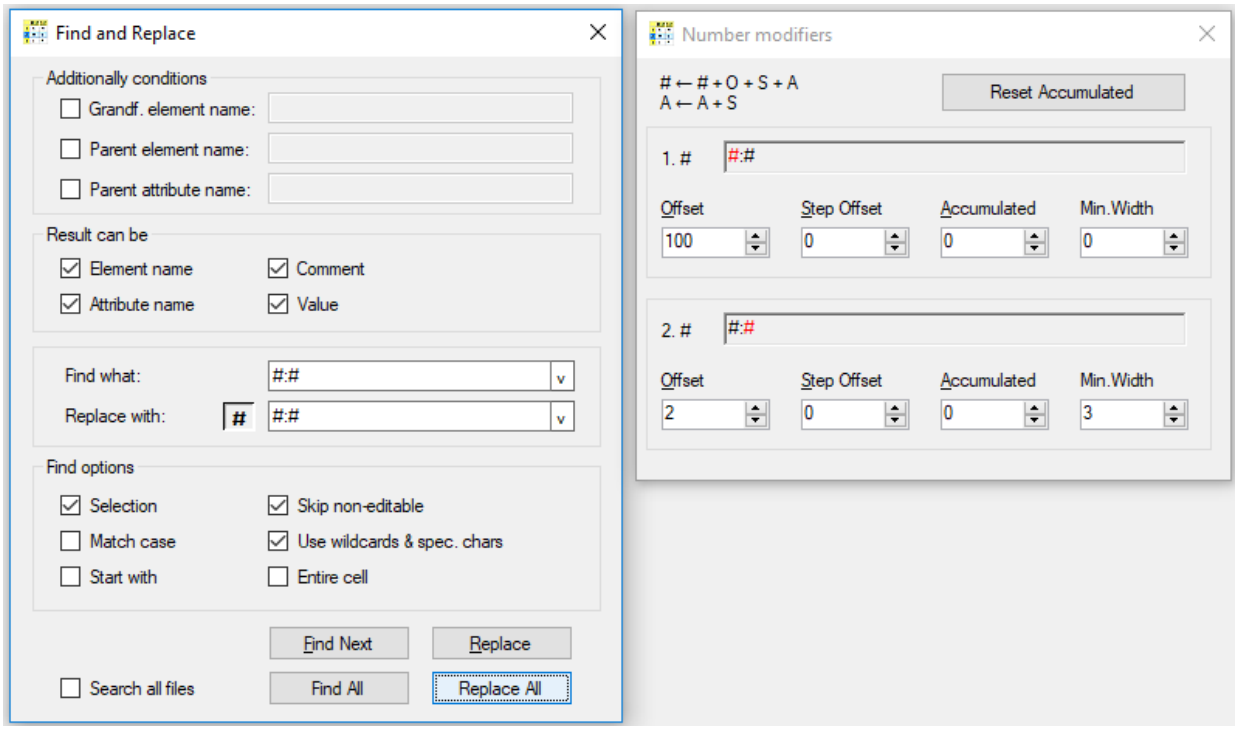
When the option ‘Use wildcards & spec. chars’ is checked and a wildcard ‘#’ is used in the replace field a hidden button for modifiers appears. This button has two states – “not active” or “active” (see a picture below). If a button is not active all wildcards ‘#’ are processed without modifications.



When the button is active then a second dialog appears, and we can specify modifiers for up to four wildcards ‘#’. By default, all modifiers are zero which results in no additional changes.

The modification algorithm is simple, and one replacing can be described like this (see the next picture for references):

1. Get the number which is found by the wildcard ‘#’
2. To this number add the *Offset* value (positive or negative) plus the *Step Offset* value plus the *Accumulated* value. Replace the old number by this new number in the text.
3. Add *Step Offset* value (positive or negative) to *Accumulated* value.



'Number modifiers' dialog contains these elements:

- *Button 'Reset Accumulated'*

Set all Accumulated values to zero. Usually but not always they are reset after one whole replacement cycle. A user should care about this.
Program will reset these values automatically only in a case when the replace pattern is modified and the count of number signs '#' decrease.

- *Label and a text (a copy of 'Replace with') with highlighted number sign '#':*

These items give only information for which number sign we are setting the parameters. There can be up to four groups of parameters.

- *Offset:*

This value will be added as a fixed offset to the found number. It can be positive or negative. If all other parameters are zero, then we can shift replaced values by a constant number.

- *Step Offset:*

This value defines the step between two replacements, and it is added to Accumulated value when the replace is taken. Also, it's added to found number.

- *Accumulated:*

This value is added to the found number as well together with offset and step offset value. It changes during the time by the Step Offset value and is used for creating an increasing or decreasing sequence. Be aware that this value must be reset before new replacement if we don't want to continue in the sequence.

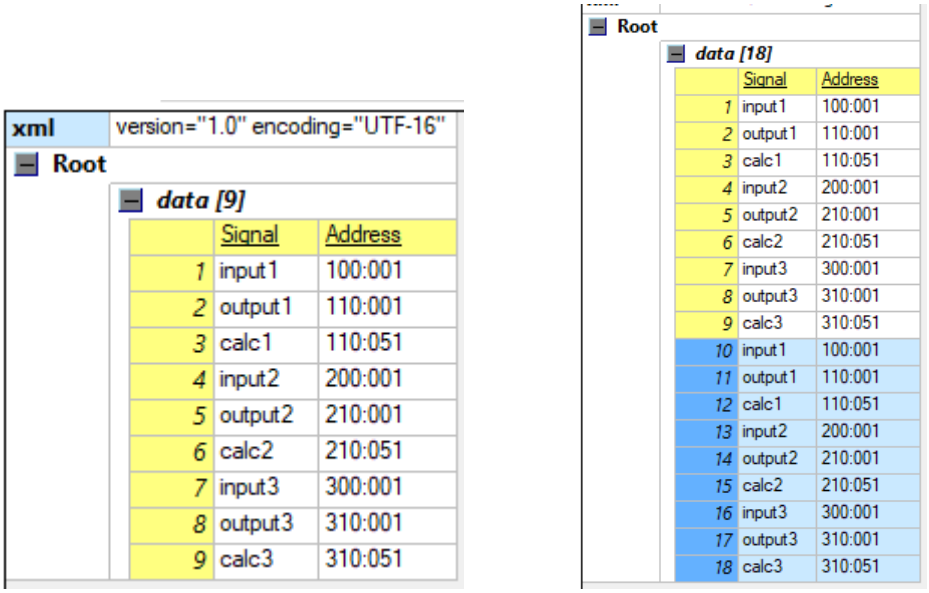
- *Min.Width:*

Here we can enter minimum width size in digits for formatting result number in a replacement text. Sometimes we need it for output which should have trailing zeros. For example, instead of number "2" we want to have number formatted as "002". For such reason we set Min.Width value to 3.
If Min.Width is zero, then no formatting is used. Also, if the number has more digits than this value the result will not change.

Two examples will clarify the description.

20.3.3.1 Example 1

We want to extend input xml file by the second block which should continue after the first block; starting by a copying the values and appending them at the end.



xml | version="1.0" encoding="UTF-16"

Root

data [9]

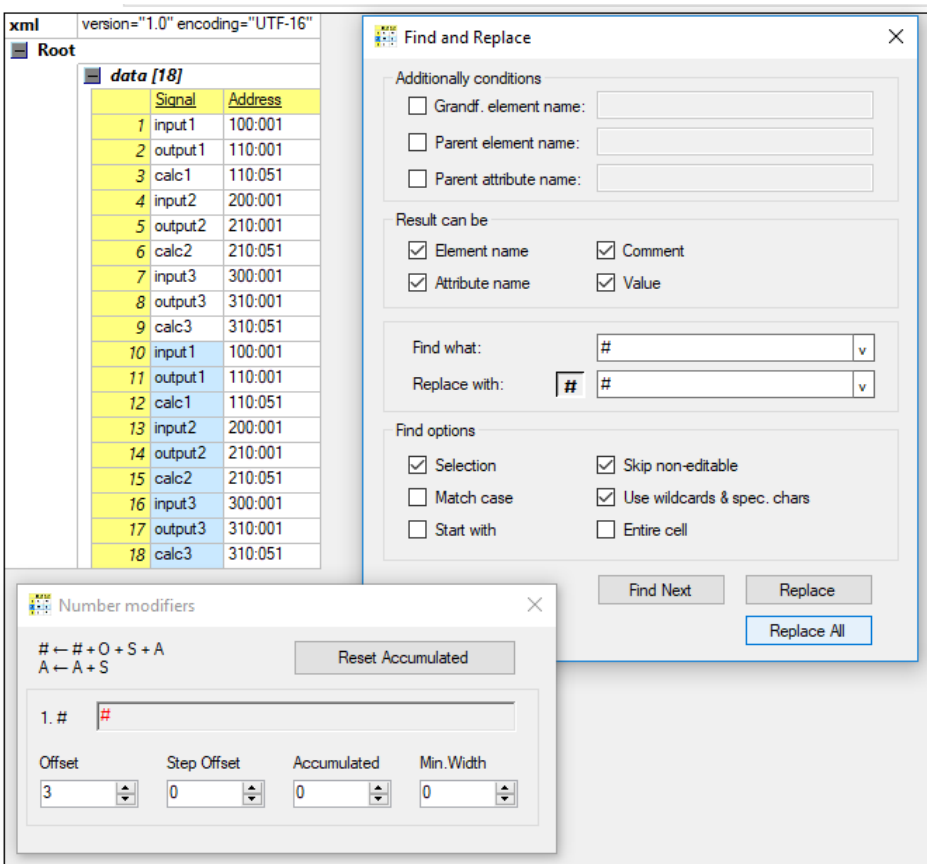
	Signal	Address
1	input1	100:001
2	output1	110:001
3	calc1	110:051
4	input2	200:001
5	output2	210:001
6	calc2	210:051
7	input3	300:001
8	output3	310:001
9	calc3	310:051

Root

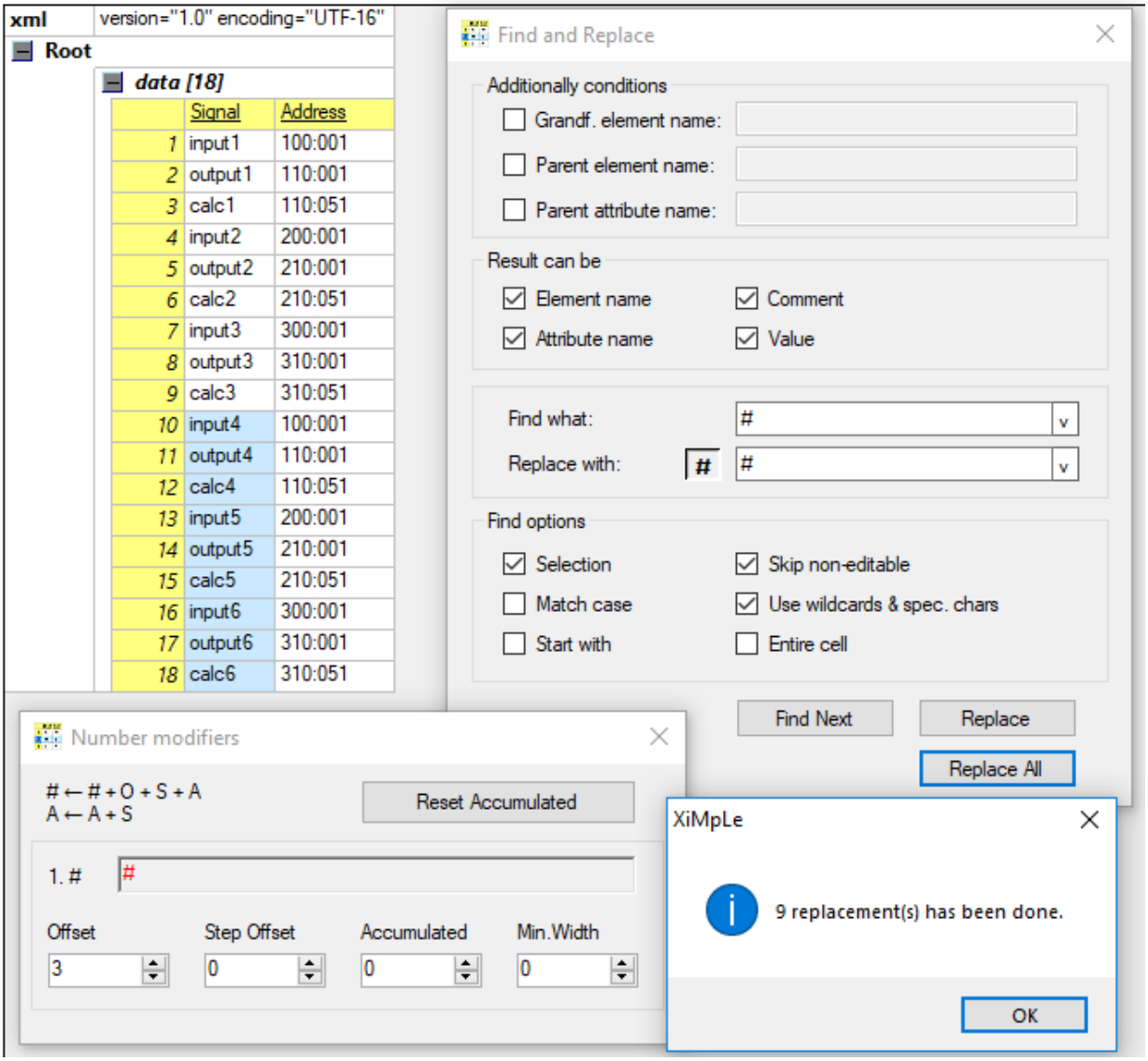
data [18]

	Signal	Address
1	input1	100:001
2	output1	110:001
3	calc1	110:051
4	input2	200:001
5	output2	210:001
6	calc2	210:051
7	input3	300:001
8	output3	310:001
9	calc3	310:051
10	input1	100:001
11	output1	110:001
12	calc1	110:051
13	input2	200:001
14	output2	210:001
15	calc2	210:051
16	input3	300:001
17	output3	310:001
18	calc3	310:051

Now we want to increase all new indexes in the first column by 3. We can use the number modifier where we set Offset = 3 and other parameters are zero. We will do a replacement over selecting area only because we don't want to change the first part.

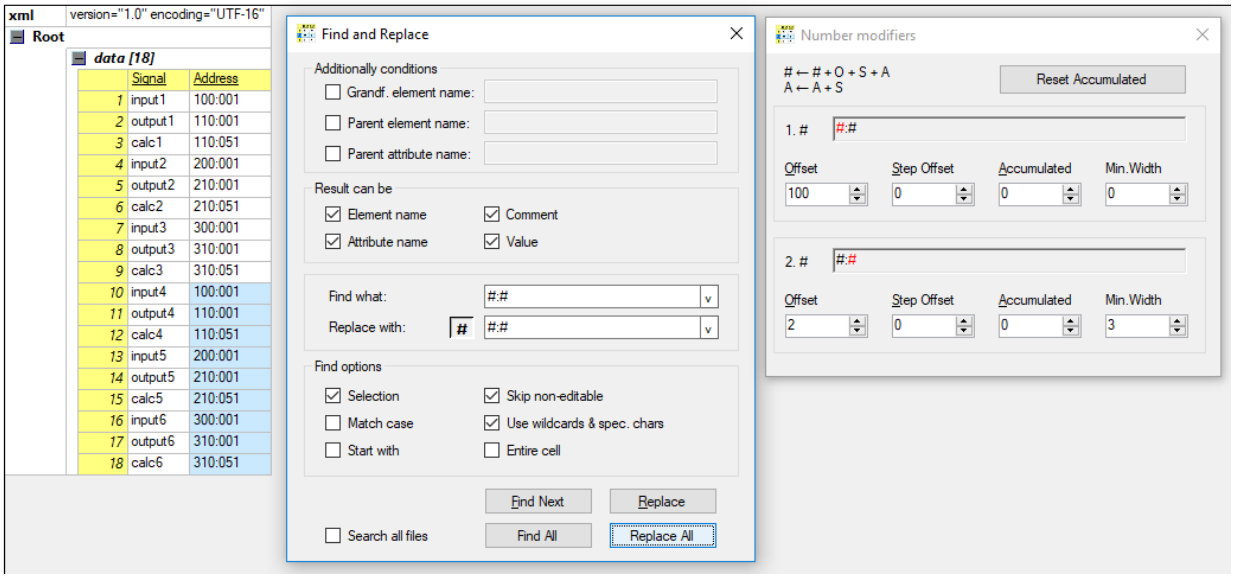


After replacing we can see that 9 replacement were done and indexes are shifted by three now.

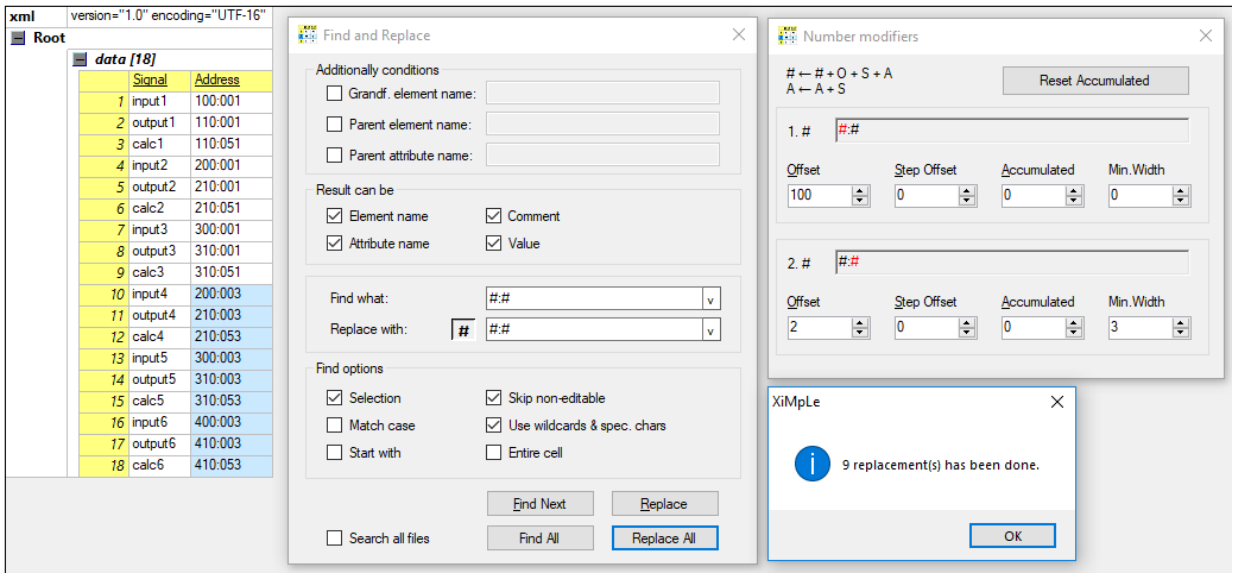


Now we will adapt the Address column values by another modification. We want to modify the first part of the address (before ':') by 100 and the second part of the address by 2 and we want to keep the format of three digits. We can do this replacement together by one command in this case. Usually, the tasks are more complicated and we must do it separately.

We define for finding the pattern '#:#' which will match the address column. For replacement we define the same pattern and set parameters for modifiers – in this case we have two groups of digits – the first found value will be increased by 100 and the second found value will be increased by 2 with the request of format width to three digits.

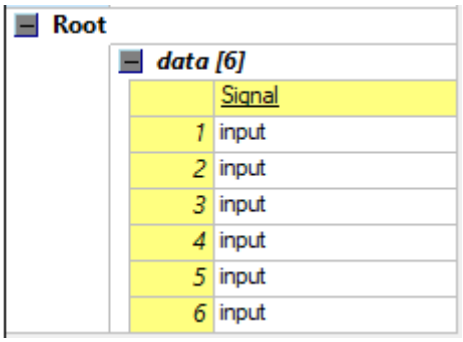


And the result after the replacement in the selected block:

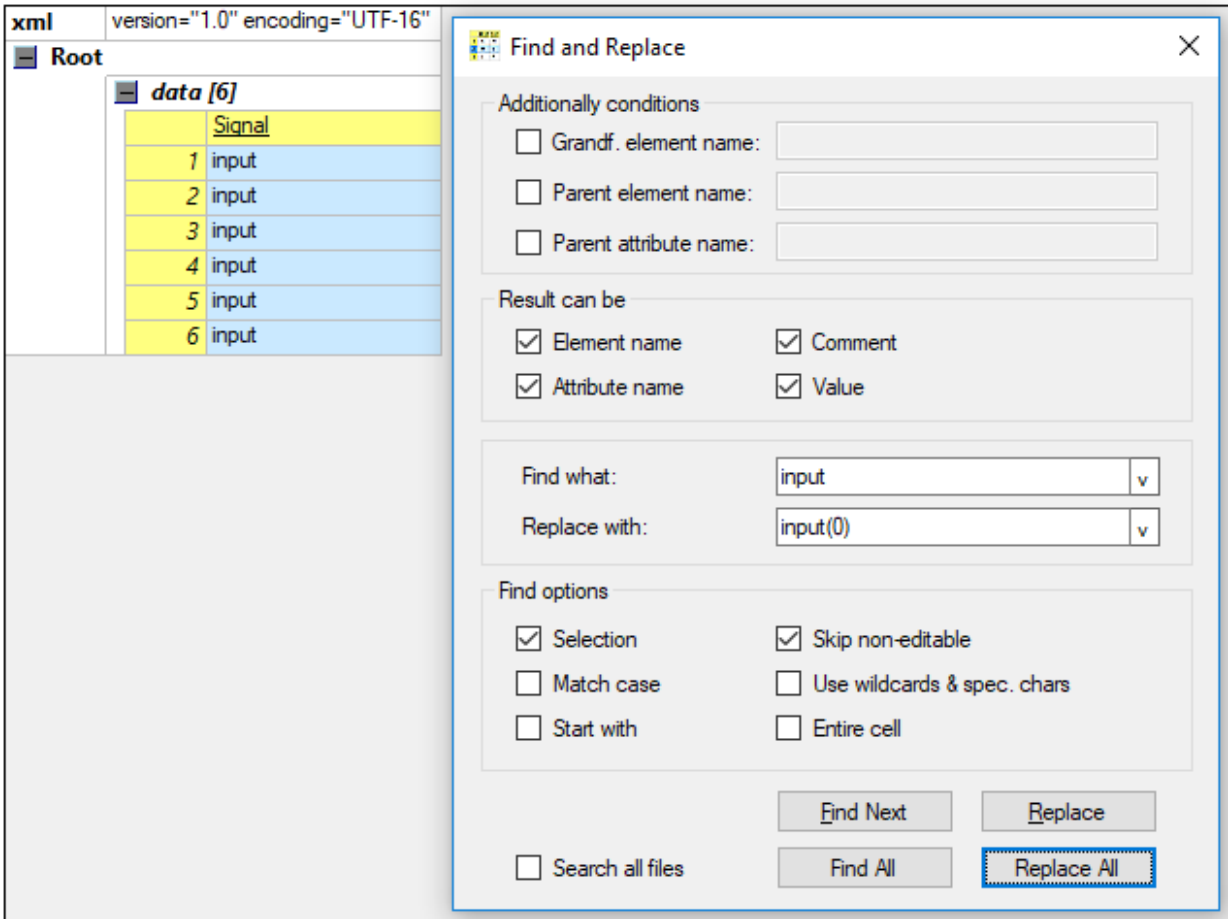


20.3.3.2 Example 2

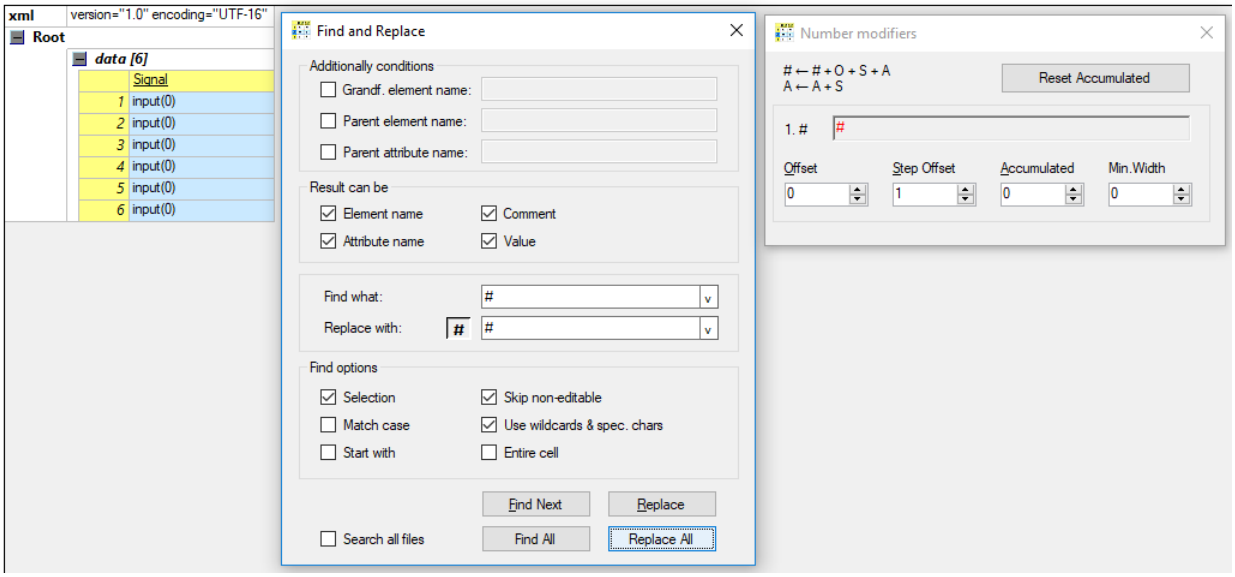
This example will demonstrate increasing sequence. We have very simple xml file and want to add increasing indexes at the end of strings.



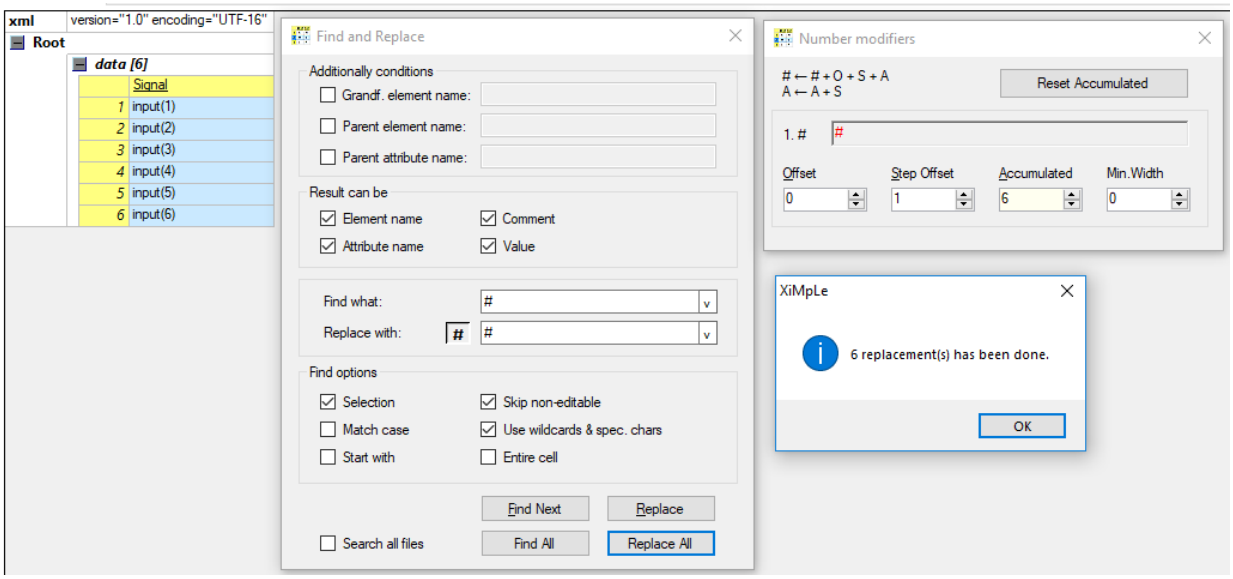
Firstly we insert a “dummy” number at the place where we want to have the sequence number.



Now we set the number modifiers: Step Offset = 1 and other values zeros to get the sequence where each value is increasing by one.



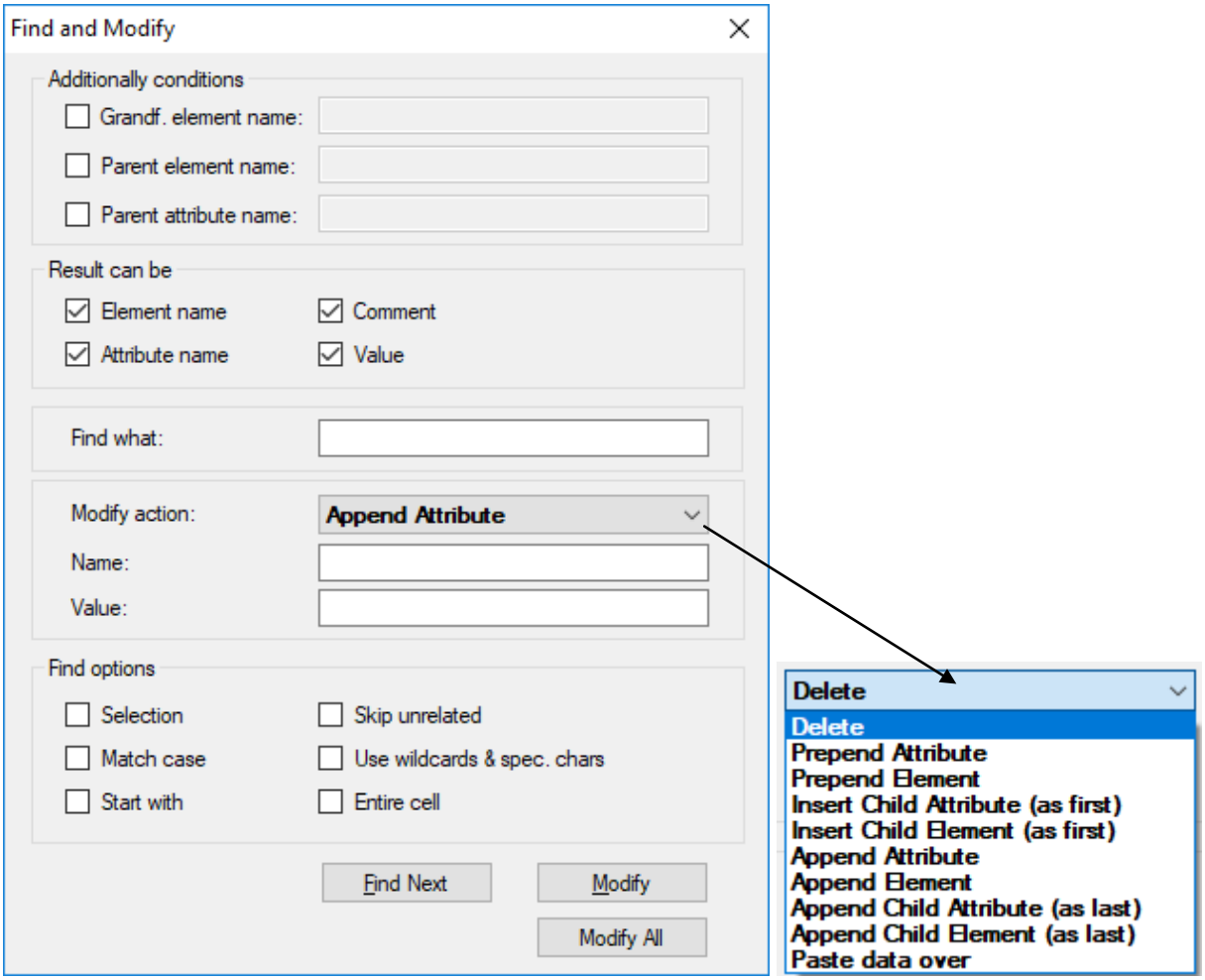
And after replace all we can check result:



Please notice the Accumulated value = 6 at the end.

20.4 Find and Modify

Most of the tasks can be achieved by sub-tables and manipulations with table columns. The find and modify functionality is a supplement for elements modifications.
The dialog is very similar to Find and Replace and the same rules are applied here for wildcards and checkboxes. Please see the section [20.3 `Find and Replace`] for details. The lookup table replacement and number modifiers are not available in this mode, but we can use the wildcards.



The differences are in the fields:

- *Modify action*

  In this combo box we must select an action from the list. This action can be applied then on the find cells. Be aware that some actions shouldn't be applied in some cases. For instance, head column will not be deleted, inside the text value can't be inserted a child element and so on (it will be discussed later). There are three main groups of actions:

  1) Delete the cell which matches the finding conditions
  2) Add some new element or attribute; we can define a name and a value for it
     a) Prepend an attribute or element – if possible, the attribute or element is added before the cell on the same level

b) Insert child attribute or element – if possible, the new attribute or element is created as the first attribute or element inside the found cell

c) Append an attribute or element – if possible, the attribute or element is added next to the cell on the same level

d) Append child attribute or element – if possible, the new attribute or element is created as the last attribute or element inside the found cell

3) Paste xml data from the clipboard over the found cell (you can see [11.2] for details of xml data in the clipboard)

- *Name*

  Here we can set the name of the newly created element or attribute. The wildcards can be used in the same manner as for the replace field in the Find and Replace dialog.

  If the field Name is empty, then a default unique name is generated. If the name contains characters which are not valid for element or attribute name, then these characters will be replaced by an underscore characters.

- *Value*

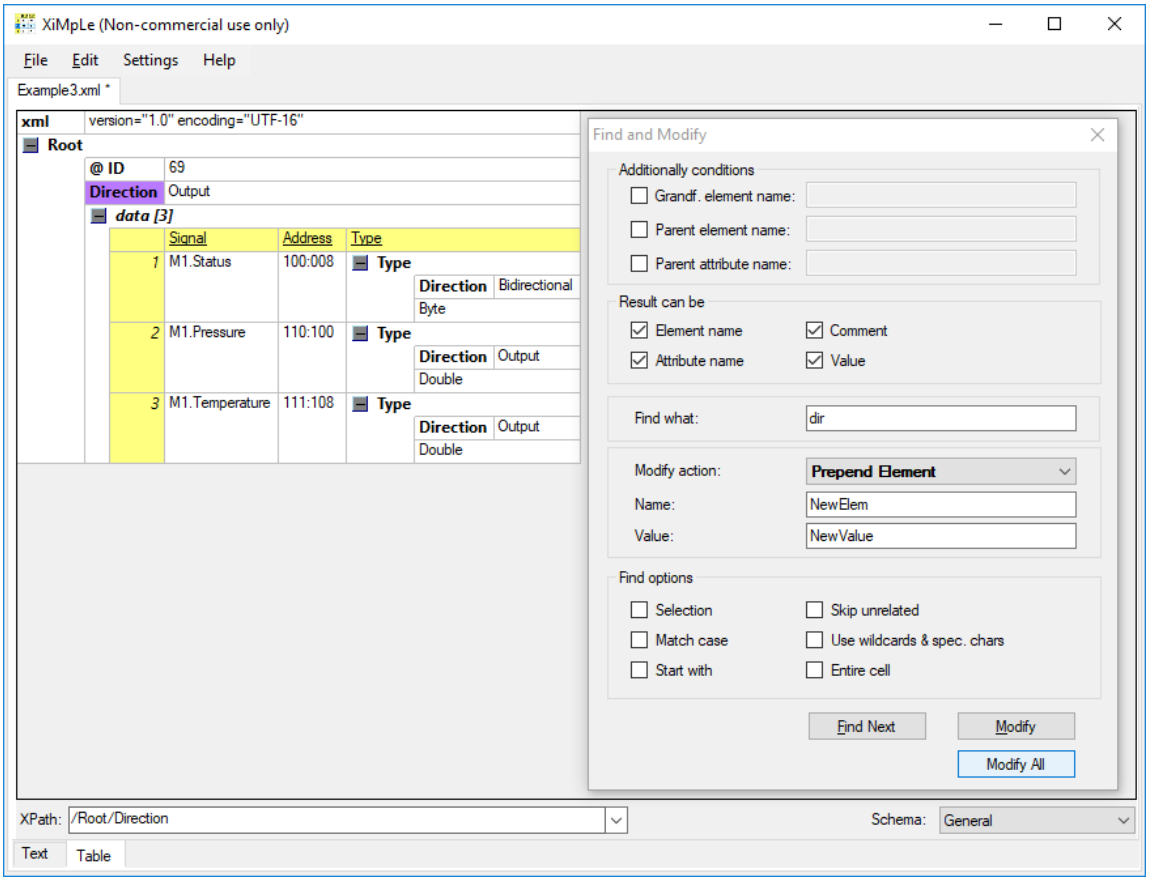
  Here we can set the value of the newly created element or attribute. The wildcards can be used in the same manner as for the replace field in the Find and Replace dialog.

Some actions are restricted and can't be applied on every cell type. The next table describes details - x marks the action which is not possible.

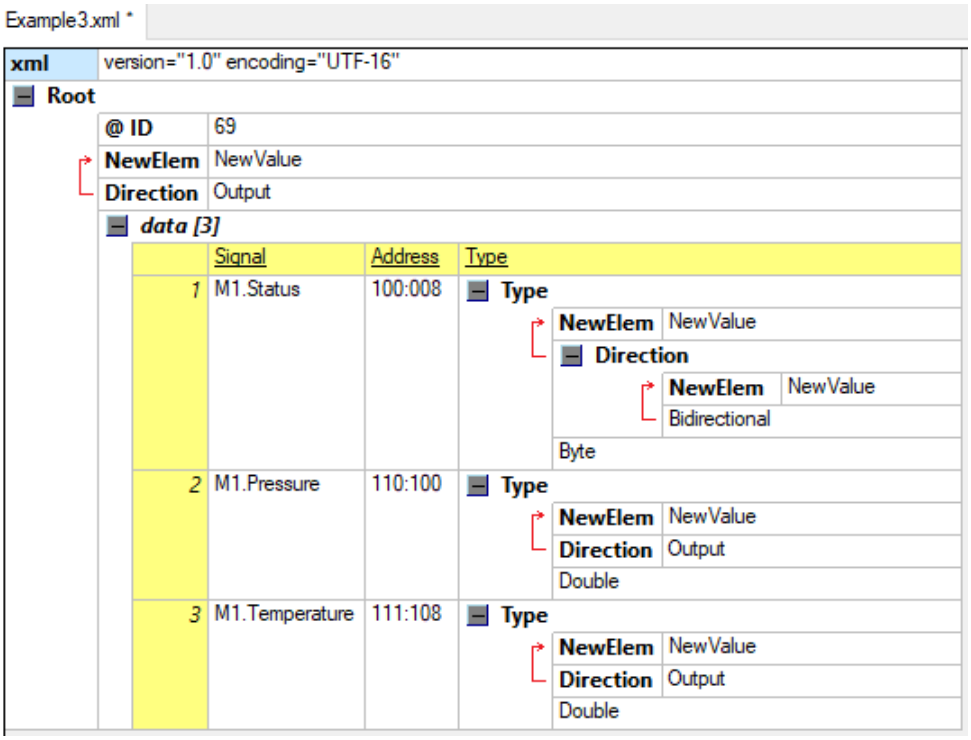
Cell type / Action	Attribute	Attribute Value	Element	Element Value	<Text> Value	Column Head	Table
Delete						x	
Prepend/Append Attribute/Element		x			x	x	x (if table is inside another table)
Insert/Append child Attribute/Element	x	x		x	x	x	x
Paste	x	x			x	x	

Let's do several examples to demonstrate the modify functionality.

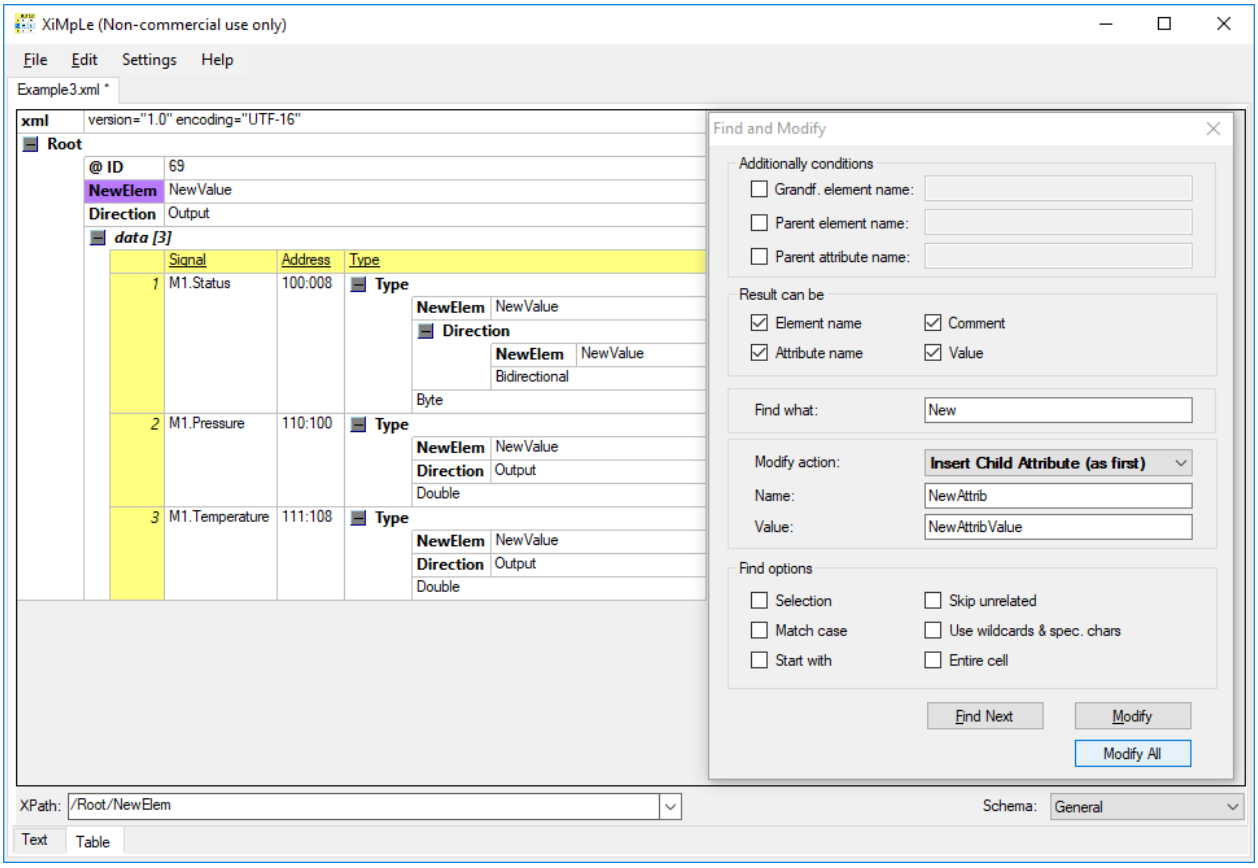
1) Find a string “dir” and prepend a new element (if it’s possible) with a name “NewElem” and the value “NewValue”. A purple colour indicates the active cell for finding (it can be customized [0]).



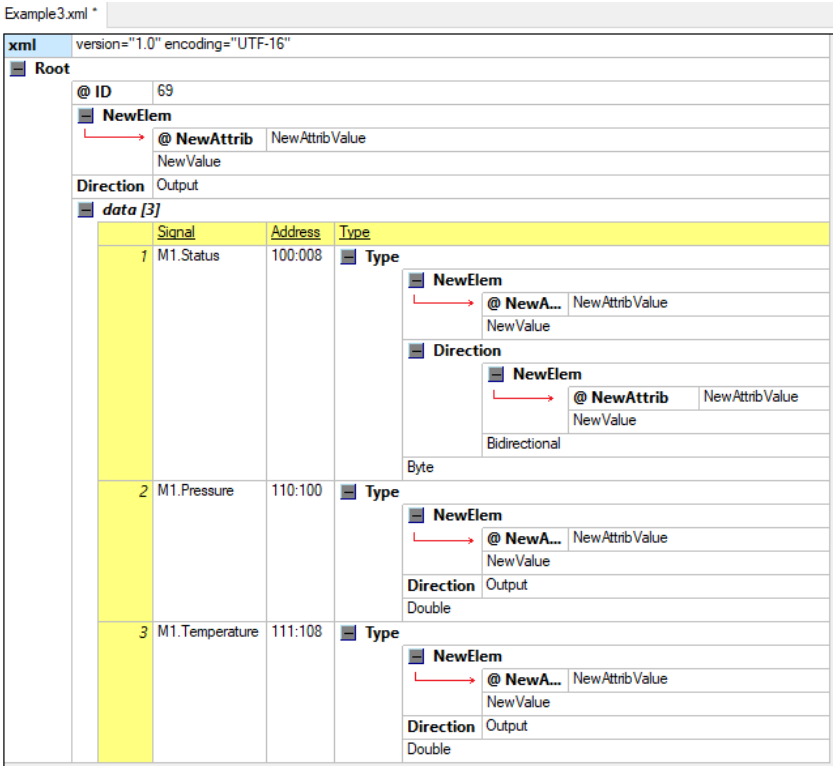
After the command “Modify All” we get this result. The new created elements are marked by red arrows for a better overview.



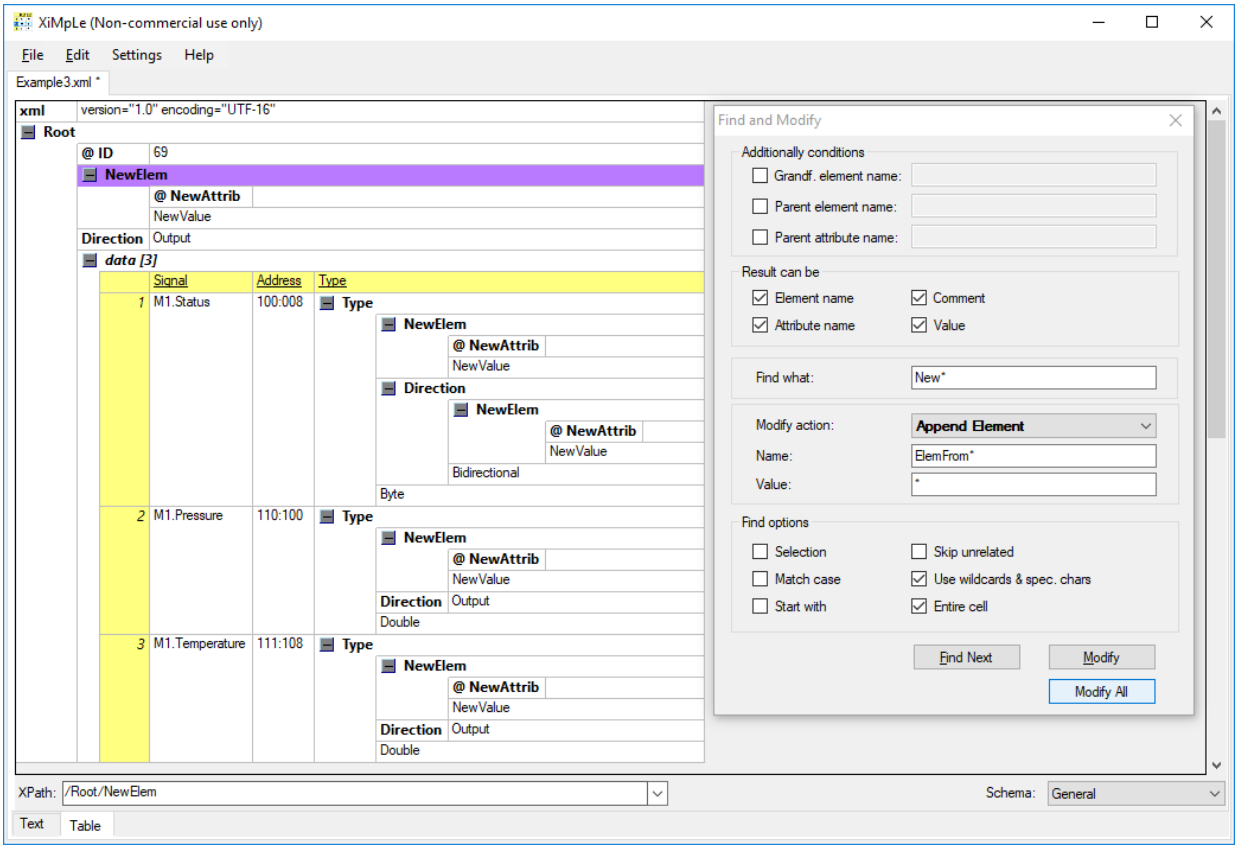
2) Find the string “New” in the file and create a child attribute (= attribute inside the found element) which is named “NewAttrib” with the value “NewAttribValue”.



And the result:



3) Find the string “New*” (using a wildcard ‘*’) and append a new element next to found cell with the name “ElemFrom*” and the value “*”. A wildcard ‘*’ will be derived from the searching field.



For a better orientation the new elements are marked by red arrows in the result.

Example3.xml *

xml	version="1.0" encoding="UTF-16"
Root	
@ ID	69
NewElem	
@ NewAttrib	
ElemFromAttrib	Attrib
NewValue	
ElemFromValue	Value
ElemFromElem	Elem
Direction	Output
data [3]	

	Signal	Address	Type
1	M1.Status	100:008	**Type**

NewElem	
@ NewAttrib	
ElemFromAttrib	Attrib
NewValue	
ElemFromValue	Value
ElemFromElem	Elem
Direction	

NewElem	
@ NewAttrib	
ElemFromAttrib	Attrib
NewValue	
ElemFromValue	Value
ElemFromElem	Elem
Bidirectional	

Byte

	Signal	Address	Type
2	M1.Pressure	110:100	**Type**

NewElem	
@ NewAttrib	
ElemFromAttrib	Attrib
NewValue	
ElemFromValue	Value
ElemFromElem	Elem
Direction	Output

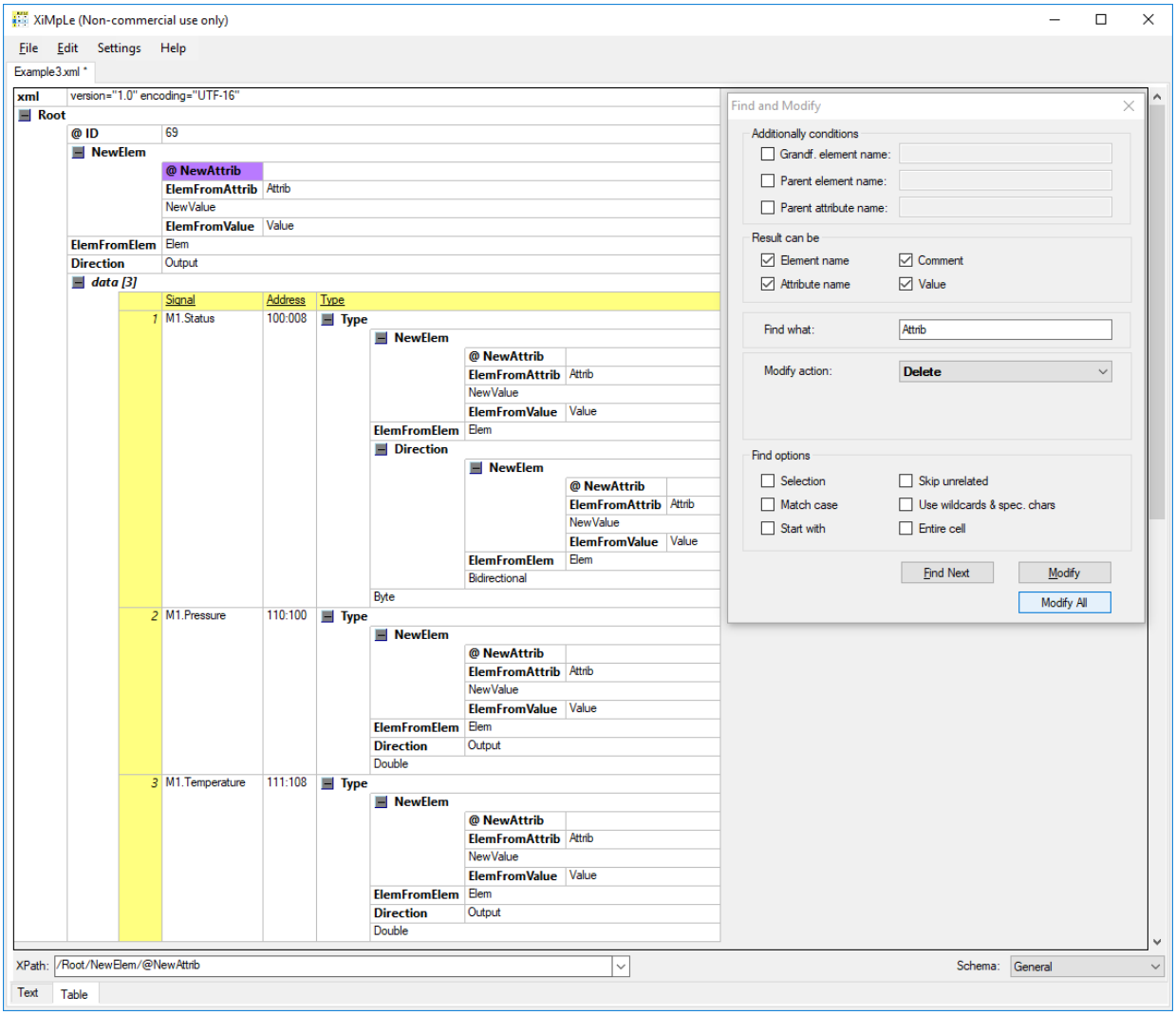
Double

	Signal	Address	Type
3	M1.Temperature	111:108	**Type**

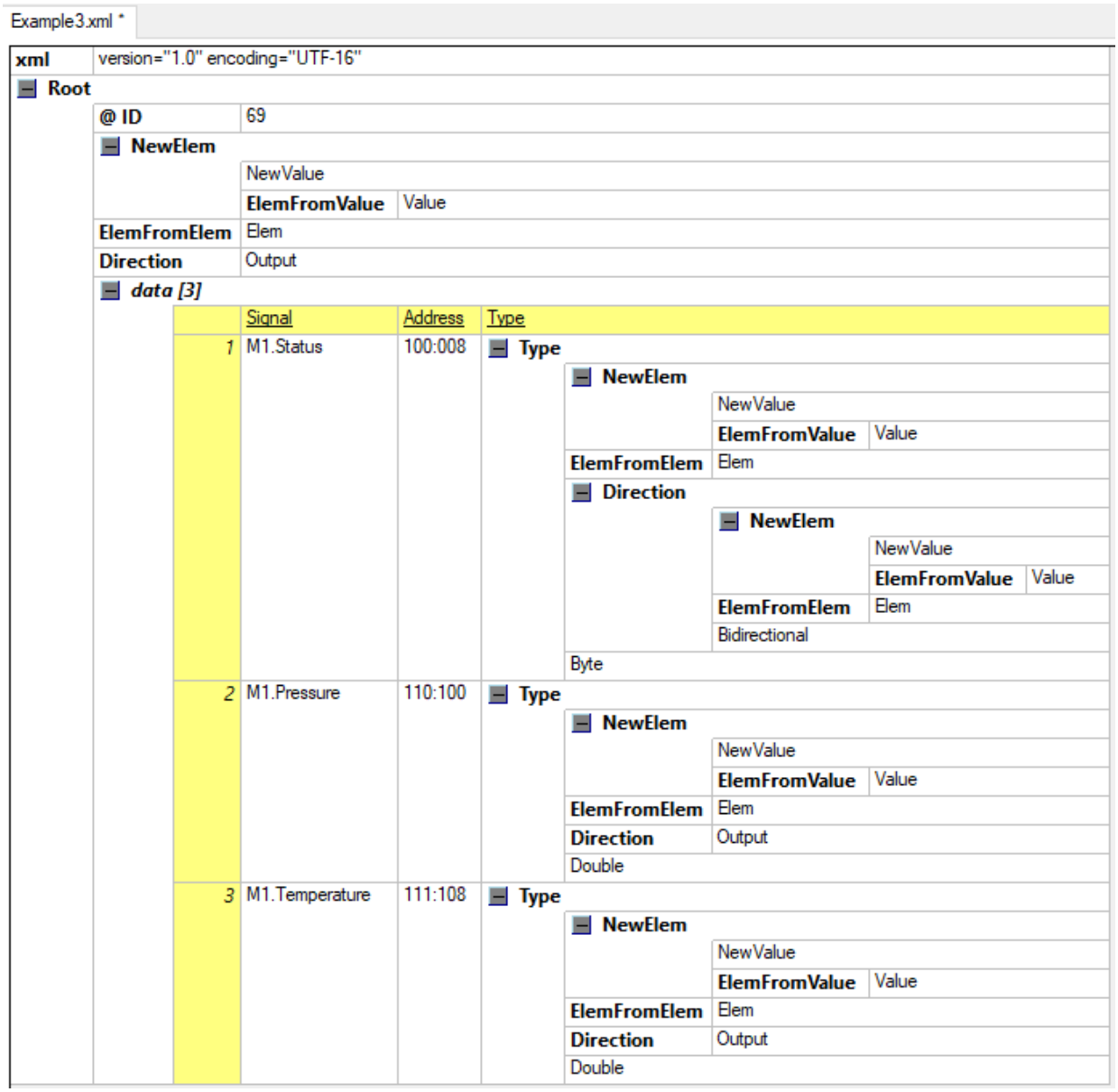
NewElem	
@ NewAttrib	
ElemFromAttrib	Attrib
NewValue	
ElemFromValue	Value
ElemFromElem	Elem
Direction	Output

Double

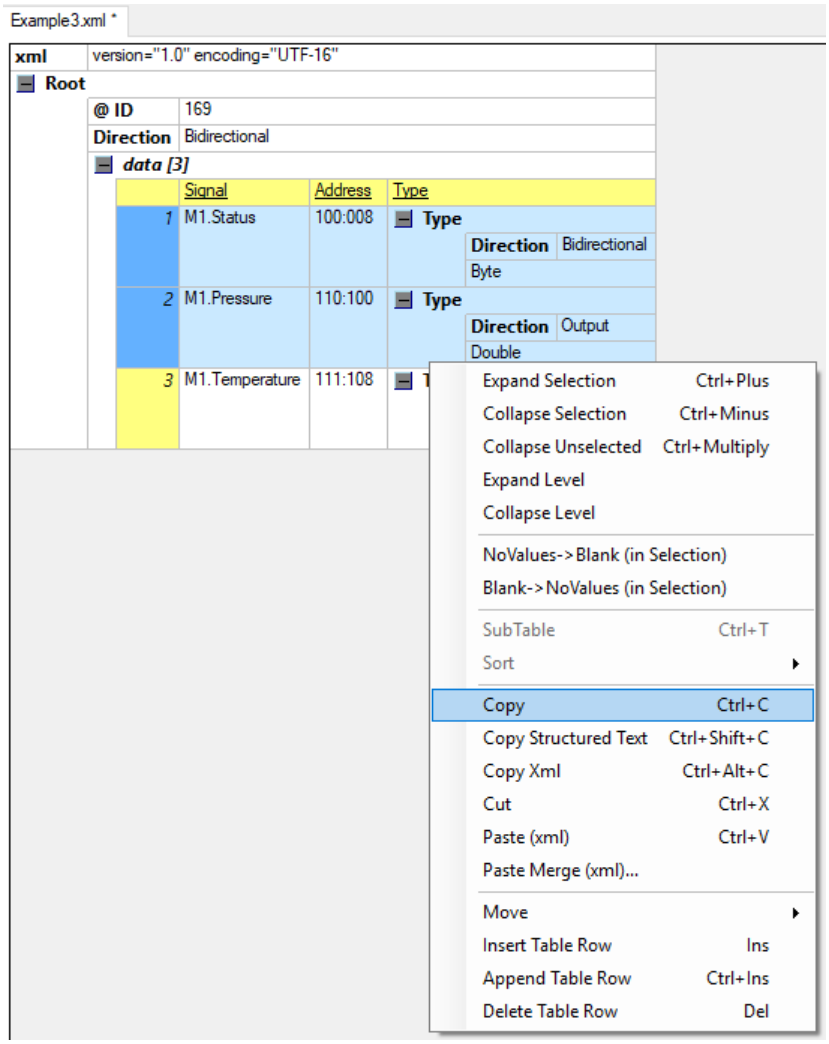
4) Delete cells which contain the word “Attrib”.



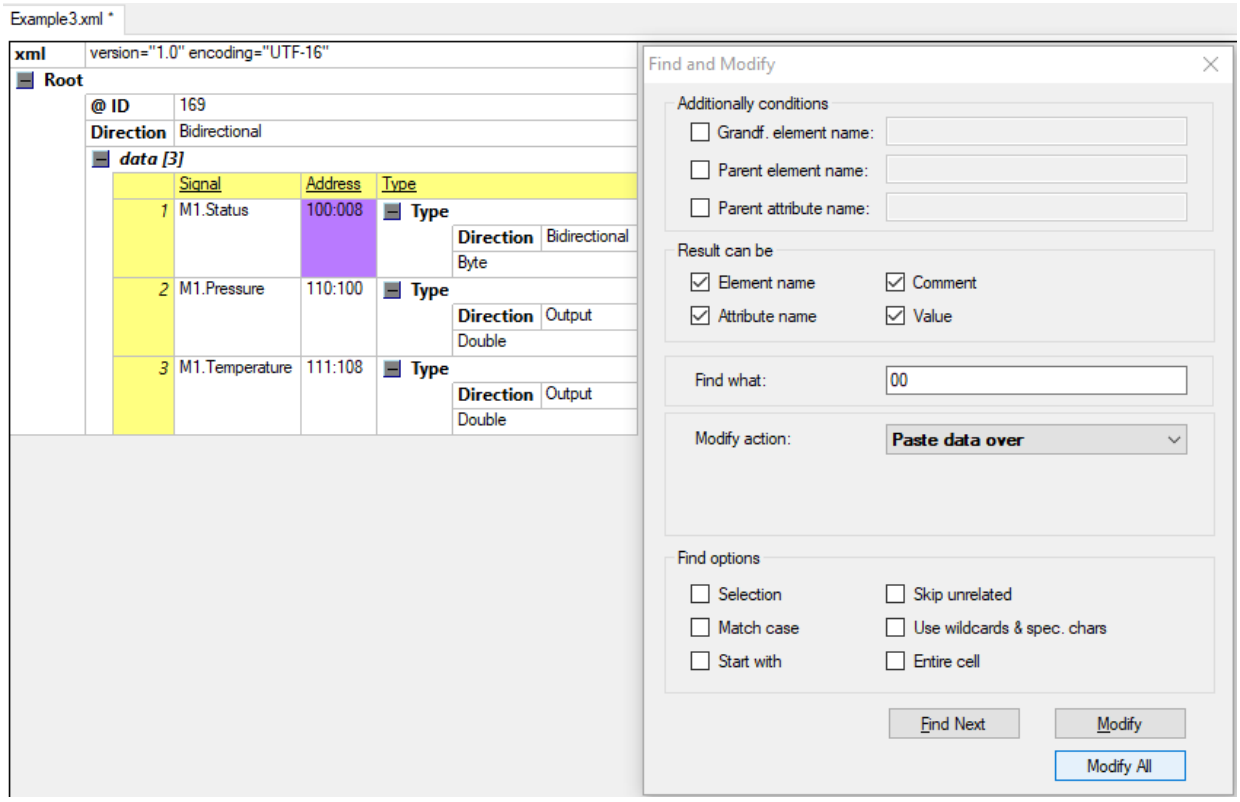
After “modify all” command we should get the result which doesn’t contain any element, attribute or value with the substring “Attrib”.



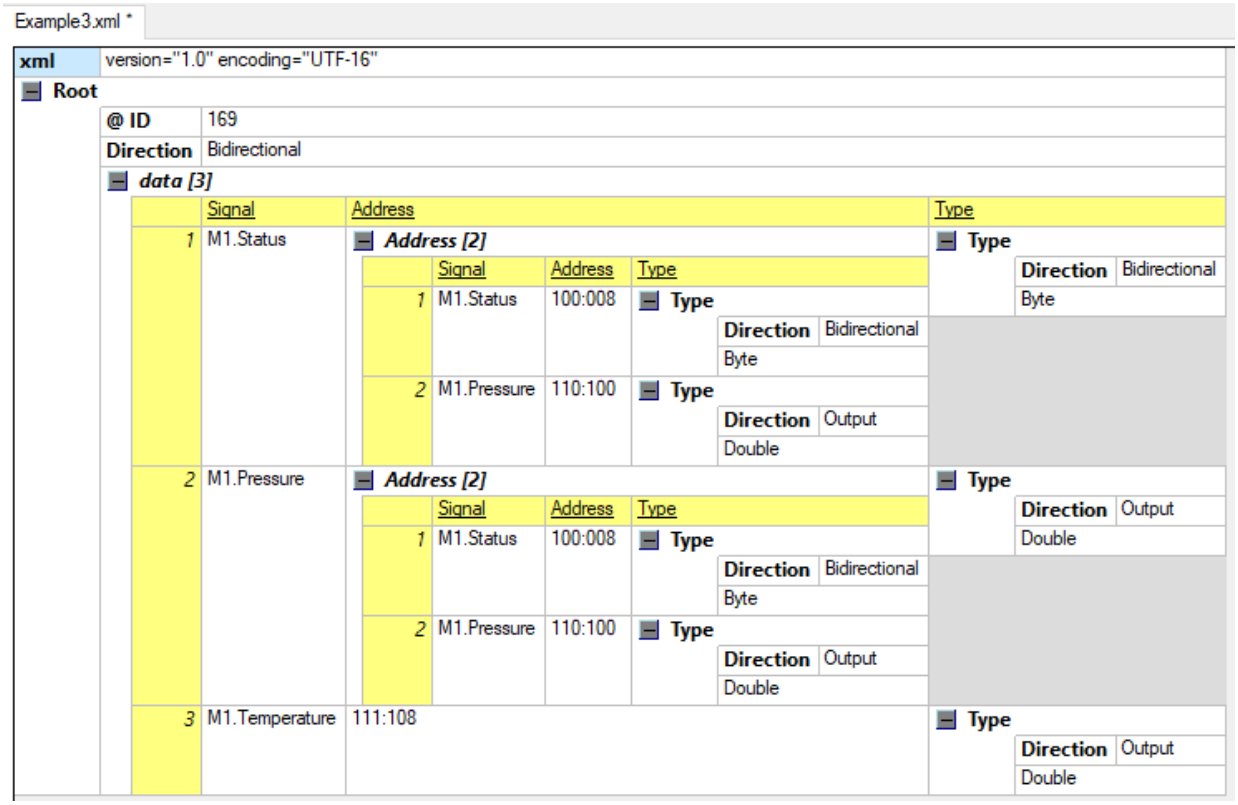
5) In the next example we would like to paste the first two rows of the table “data” over the cells which contain two zeros “00”. Firstly, we prepare the clipboard data by a copy command...



Then we use the “Paste data over” for modify action and search for the text “00”.



We can check the result – two cells were replaced by data from clipboard.



20.5 Bookmarks

This menu item contains four functions which are designated for managing and using XPath bookmarks. They can be used only in the table editor. For more details please see the section [26. `XPath Bookmarks`].

20.5.1 Add / Delete Bookmark

This command adds a current XPath as a bookmark. If this XPath has been already bookmarked it's deleted.

20.5.2 Next Bookmark

Go to the next bookmark from the bookmarks list. This list can be viewed in the combo box (on the bottom of the screen next to XPath edit box).

20.5.3 Previous Bookmark

Go to the previous bookmark from the bookmarks list. This list can be viewed in the combo box (on the bottom of the screen next to XPath edit box).

20.5.4 Delete All Bookmarks

Delete all bookmarks from the list.

20.6 Refresh Grid & Clean Exposing

This is used for refreshing a table grid view and removing exposions which were done by commands [5.2 `Expose table row(s)`] and/or [5.3 `Expose element as table row`].

20.7 Adapt Columns Width

Can be used if we want adapt width of all displayed columns for a current view. If we want to adapt only one column we can double click on the bound line of a column or resize it manually by dragging a column bound and move it horizontally. There is some minimal width of a column applied so a column can't be made invisible.

20.8 Normalize spaces

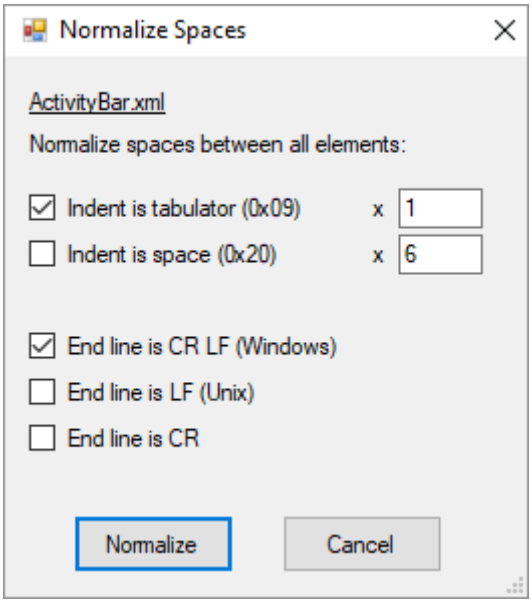
It can happen that spaces between elements can differ through a document. For an xml data processing these spaces are irrelevant but for a human reader it can be disturbing. This function can normalize these insignificant spaces.

A user can choose if spaces will be made by tabulator characters or by spaces and how "wide" these spaces will be (a number of tabulators or a number of spaces used for one nesting level must be entered).

Additionally a user can change the type of line endings.

This functionality can be used also for JSON files.

The initial values for this dialog are taken from the current schema configuration.



For example:

```
<Person>
	<Name>John</Name><Age>25</Age>
</Person>
	<Person>
		<Name>Paul</Name>
		<Sex>Male</Sex>
	</Person>
```

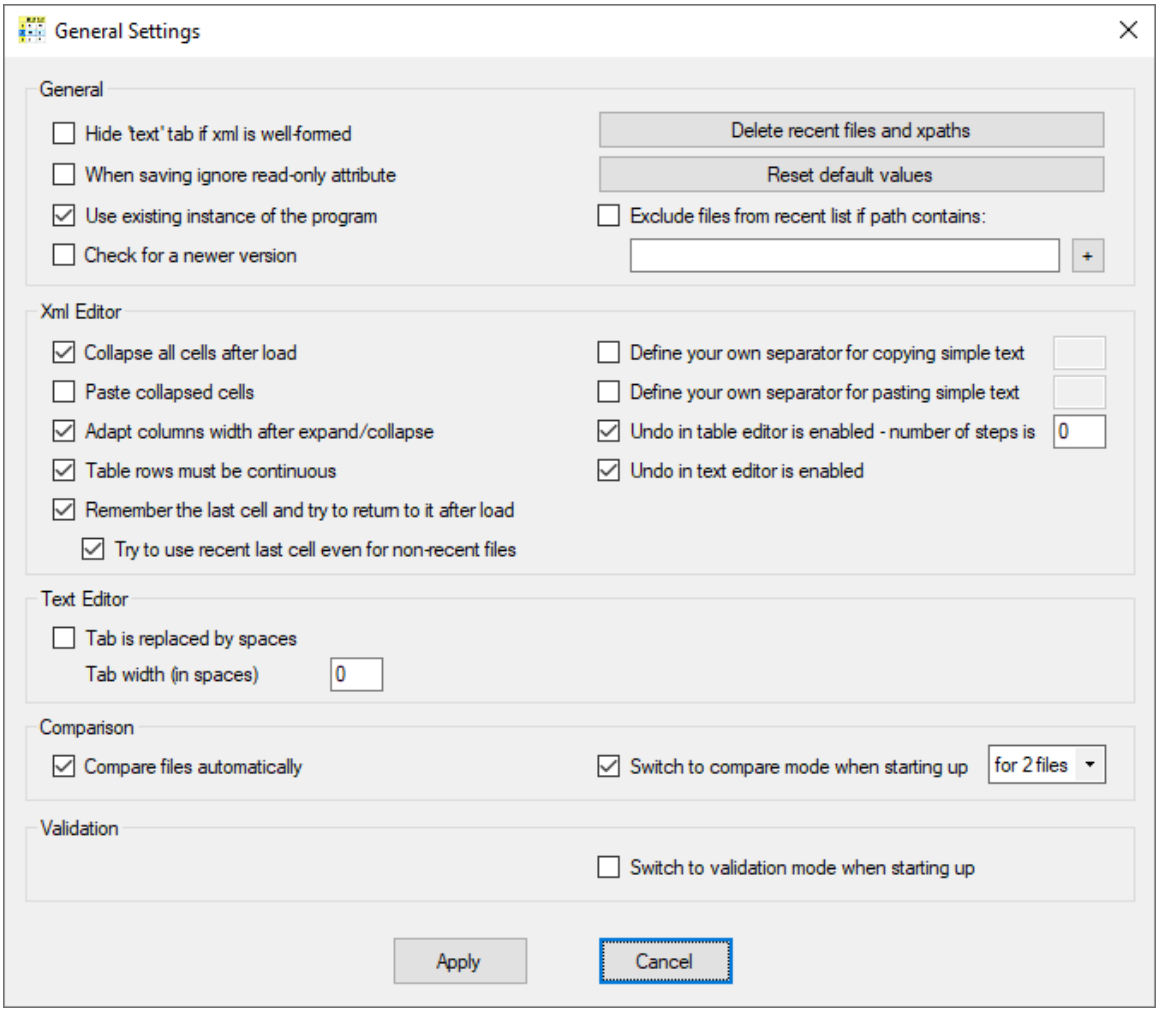
Will be normalized like this:

```
<Person>
	<Name>John</Name>
	<Age>25</Age>
</Person>
<Person>
	<Name>Paul</Name>
	<Sex>Male</Sex>
</Person>
```

21 Settings Menu

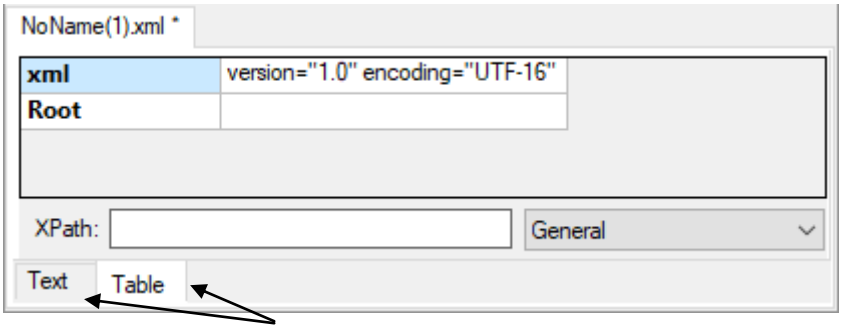
21.1 General...

General settings can be set by this command. A dialog box appears.



- *Hide 'text' tab if xml is well-formed*

If this option is not checked then together with a grid editor also a text editor is accessible by a tab.



If it's checked then a 'Text' tab appears only when an error is found in xml data or the file is not valid xml file.

- *When saving ignore read-only attribute*

If a destination file is read-only we can set our wish to remove this flag and overwrite a file without any asking.

- *Use existing instance of the program*

If this option is checked then for opening files will be used already running instance of the program. There are some exceptions:

  - If running XiMpLe without any file – a new empty instance is opened
  - If program is executed with two files for comparing and the option "*Switch to compare mode when starting up*" is set to "*for 2 files*".
  - If existing instance is running script

- *Check for a newer version*

If this option is selected, once per day the program checks for a newer version. If a newer version exists it's reported once and no more. Next announcement will appear, when the latest version increases again.

- *Delete recent files and XPaths*

By this button we can delete information about lastly opened files and lastly used XPaths.

- *Reset default values*

This will re-set the default settings.

- *Exclude files from recent list if path contains:*

This option enables or disables checking if the currently opened file contains in its path some of text(s) defined in the adjacent edit box bellow. If yes, then such file is not included into the recent file list. More texts are separated by a semicolon character.
For example if we define a text "C:\Temp\; error ;" then all files which are in the folder C:\Temp\ and all subfolders will not be stored in the recent file list. Also any file which contains text " error " as a part of its path will not be stored to this list. Comparing is not case sensitive.

- *Collapse all cells after load*

If checked all cells except root element are collapsed after loading data.

- *Paste collapsed cells*

If this option is checked then pasted data are displayed as collapsed. Otherwise data are fully expanded when pasted.

- *Adapt columns width after expand/collapse*

If this option is checked and you expand or collapse some cell(s) then widths of all columns from the column where you expand or collapse the cell up to the last column are automatically resized.

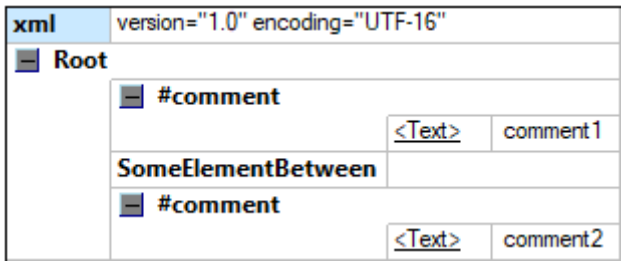
- *Table rows must be continuous*

If this option is not checked then a table will collect all row elements even they are not continuous (for example if some another element stays in between).

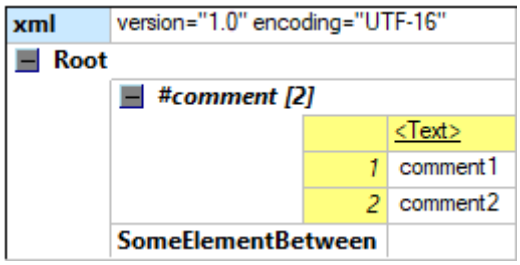
Very simple example (comments can form a table as well):

```
<!-- comment1 -->
<SomeElementBetween/>
<!-- comment2 -->
```

If tables must be continuous a "comment1" is displayed. Then "SomeElementBetween" is displayed and then "comment2" is displayed.



In the other case the "comment1" and "comment2" are displayed in the same table. Then "SomeElementBetween" is displayed.



- *Remember the last cell and try to return to it after load*

When a file is closed the last cell position is remembered. If we check this option XiMpLe tries to jump to the lastly remembered position when this file is opened again.

- *Try to use recent last cell even for non-recent files*

This option checked will try to use some valid recently used path for a loaded file even the file name is not between recent files. It can be useful (for example) in a situation when we want to edit different files with the same structure on the same place.

- *Define your own separator for copying simple text*

By this option we can define and use our own separator which will be used for copying simple text data. Instead of the default tabulator character another one character or a whole string can be used.

- *Define your own separator for pasting simple text*

By this option we can define and use our own separator which will be used for pasting simple text data. Instead of the default tabulator character another one character or a whole string can be used.

- *Undo in table editor is enabled - number of steps is #*

This option enables or disables Undo and Redo functions in the grid editor. Moreover we can restrict the number of Undo steps which will be remembered. Number 0 is used for an unlimited Undo stack.
Disabling Undo functions will increase a performance a little bit. Undo stack number restriction can decrease an amount of memory required by the program. This option is valid only for xml editor.

- *Undo in text editor is enabled*

This option enables or disables Undo and Redo functions in the text editor. Disabling this option is meaningful in the case of editing a big file in combination with a massive 'replace all' function.

- *Display all elements as tables in standard mode*

If this option is selected, all elements are displayed as tables. The option is valid only in standard mode and it's ignored in the comparing and validation mode. While this option is selected, the command "Expose Element as Table Row" is not available.
This option is available for 64bit version only.

- *Tab is replaced by spaces*

If this option is checked then tabulator character is replaced by adequate number of spaces when we are typing in text editor. This affects also inserting tabs in block when more lines are selected. This option doesn't affect the tab characters which do already exist in the file or pasting text or replacement when the tab character '\t' is specified.

- *Tab width (in spaces)*

Here we can specify the width of tab character displayed in the editor. The unit is a space character width. If the option '*Tab is replaced by spaces*' is checked then tab character is replaced by this number of spaces when typing in the editor.
By default, there is a number zero which means that program will determine the tab width automatically from the used font (usually it is 8 spaces for mono-space fonts).

- *Compare files automatically*

This option is used only in comparing mode [see section 23.1]. This option defines if comparing is done automatically for any two files which are displayed (option is set) or if a comparison is done only on a request (option is not set) - with one exception: when the file is loaded on the right side the comparing is invoked automatically.

- *Switch to compare mode when starting up*

It's possible to switch to compare mode immediately after XiMpLe starts. When this option is active there are two possibilities for selecting:

a) *for 2 files*
   If just two files are used as input parameters (e.g. drag and drop two files on the shortcut or run from a command line with two input files) they are loaded in compare mode and comparing is performed.

b) always

The program starts in comparing mode.

If this option is unchecked program starts in the standard (or validation) mode and to comparison mode is switched only by a request from the command in the menu Tools.

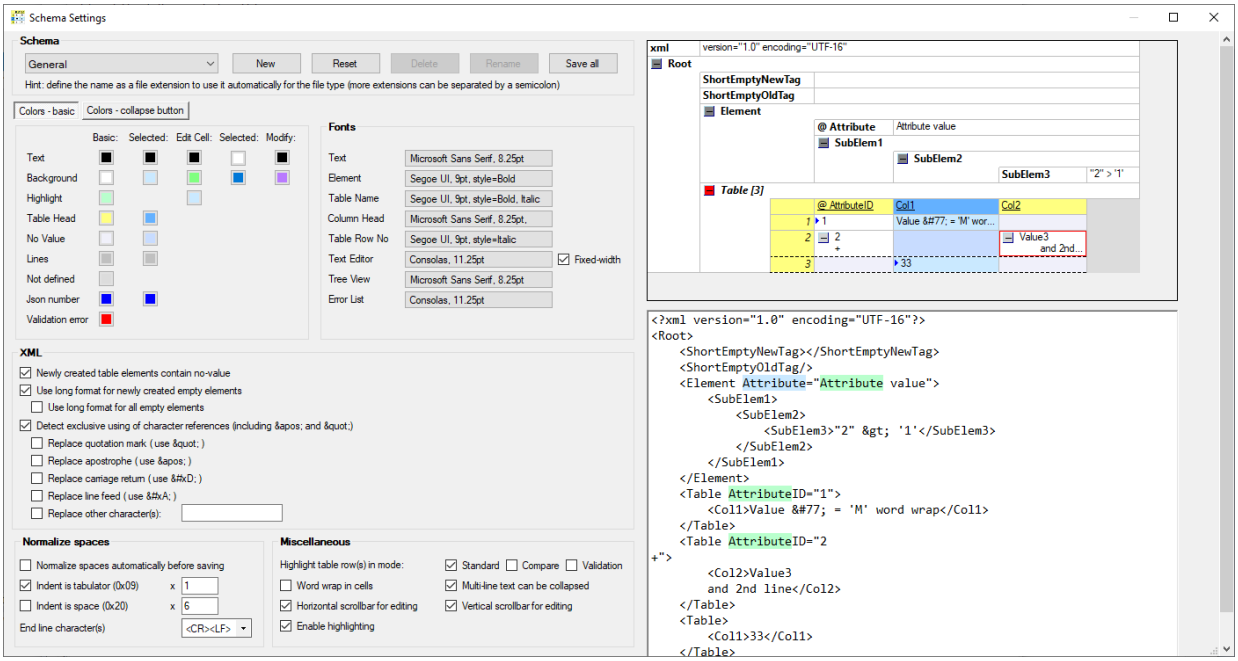
- *Switch to validation mode when starting up*

By this option we can start program in validation mode. This option is mutually exclusive with the option *Switch to compare mode when starting up – always*.

21.2 Data and Table...

Settings for xml data and grid displaying can be arranged here.

Changing of these options is almost every time demonstrated immediately on the right side of the dialog so we can observe effects of changes (e.g. changing fonts, colors or xml output settings).



21.2.1 Managing schemas

By default there exists only one schema named 'General'. But we have a choice to create and define other schemas suited for our purposes. A schema can be selected directly in a grid editor and the settings are applied immediately. In a right bottom corner of the main screen is located a list with all defined schemas.

For schemas managing there are several buttons:

New – creates a new schema with default values
Rename – sets the new name for a schema
Delete – deletes a schema (the default General schema can't be deleted at all)
Reset – resets all values of a schema to default ones
Save all – stores all schemas to the Settings file

Note: Any schema besides the default one can be renamed and associated to one or more file extensions. If we want to associate a schema for some file extension we must set the extension name for a schema's name.

When some file is loaded and there exists a schema which match the extension, this schema is used instead of the default one.

For instance: we have created a new schema which we want to use mainly for files with extensions .abc, .abd and .abe. Then we should set the name of a schema to "ABC;ABD;ABE". Extensions are separated by a semicolon character. Then anytime a file with one of these extension is opened this schema will be used.

21.2.2 Fonts section

We have a possibility to change fonts used for displaying Texts (data values), Elements (names), Table names, Column heads (names in the columns' heads of a table), Row number in a table and a font used in a text editor. All these settings should be observed in a Test grid or in a Test editor on the right side of the dialog.

Settings Tree View and Error List are related to validation mode and they are not demonstrated in this dialog. If selected font for Tree View has Italic style it's ignored and used without this style because it's reserved for other purposes.

21.2.3 Colors section

21.2.3.1 Colors - basic

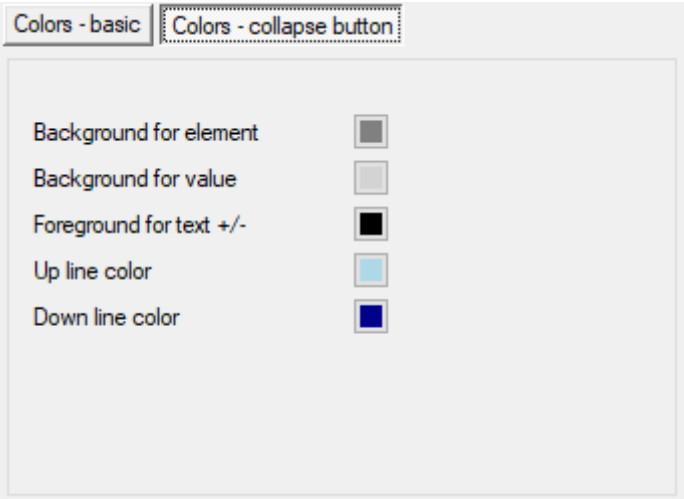
For colors there are five different groups: not selected cell, selected cell, edited cell, selection in edited cell and a cell highlighted for a modification (20.4 `Find and Modify`).

We can change a default color for a text (data values), a normal cell background, background for highlighted texts which are equal to the selected text, a background of a table column head, a color for a no-value, a color for grid lines and a color for not defined cells (empty space where are no data - between tables for instance).

For JSON files we can define the colour of a small triangle in front of the values which are considered as numbers or constants (will be saved without quotation marks).

For validation mode the colour for highlighting cells with errors can be specified.

21.2.3.2 Colors - collapse button



We can set here colors for collapse buttons. The background color – there are two versions – one is used for elements and one for text value if it contains more than one line and the setting 'Multi-line text can be collapsed' is selected.

Foreground text is a color for plus or minus character on the button. Up or down line color appears on top and left side of the button or down and right side of the button respectively.

21.2.4 XML section

In this section we can set xml properties. Most of these settings are applied during saving.

- *Newly created table elements contain no-value*

When a new element (which is a part of a table) is created it can contain empty value or no-value. An element which contains no-value and nothing else is not saved. This option is ignored sometime – for instance a new table element column will always contain no-values. This setting is used mainly for a new empty row inserting and for some pasting operations.

Remark: Currently there is a possibility to change blank values to no-values and reversely from a context menu.

- *Use long format for newly created empty elements*

There are two ways how to store an empty element:

1. Long format: <Tag></Tag>
2. Short format: <Tag/>

- *Use long format for all empty elements*

This option can be useful if we want to transform all empty elements in the file to a long format. Some programs don't need to support a short format.

- *Detect exclusive using of character references (including ' and ")*

A character in an XML file can be represented by a character itself or by a reference which is looking like &#xN; where N is a hexadecimal number of a character (a decimal number can be used in a form &#N; as well).

Moreover for some characters a special text can be used. For an apostrophe can be used ' and for a quotation mark can be used " (it's so called entity reference).

Some programs can prefer xml files using one style of a character representation (e.g.
 instead of a line feed character). If we want to keep representation already used also for newly added characters this option can help us to keep the changes consistent.

Some additional related information can be obtained in the editor by a "File/XML information" command.

For example if in an XML file is used only
 and there is nowhere used line feed character in the original file then all newly entered line feed characters will be converted to a character reference string even it hasn't been set in a replacement option for a line feed.

Let S1 := line feed character and S2 := line feed character reference representation

Then we can distinguish possibilities sorted in a table from the point of a saved file result.
Value 'X' means any option (Yes or No it doesn't matter).

S1 is in a source file?	S2 is in a source file?	Detect exclusive... option checked?	Replace line feed option checked?	Let's add S1 to some element value and save a file. Will be S1 transformed to S2?
Yes	No	X	No	No
Yes	No	X	Yes	Yes
No	Yes	Yes	X	Yes
X	Yes	No	No	No
X	Yes	X	Yes	Yes
Yes	Yes	Yes	No	No

The conclusion is that if there are mixed up a character with its reference and we want to have references only in a result file we must set and define a related "Replace character" option.

If there is used only one type of a character we can rely on an option "Detect exclusive using of character references". It's recommended to use this option.

- *Replace quotation mark (use ")*

This option can force replacing quotation characters by a special reference "

- *Replace apostrophe (use ')*

This option can force replacing apostrophe characters by a special reference '

- *Replace carriage return (use
)*

This option can force replacing carriage return characters by a special reference

- *Replace line feed (use
)*

This option can force replacing line feed characters by a special reference

- *Replace other character(s):*

Here we can define values greater than zero (hexadecimal values must be prefixed by 'x') separated by a semicolon for characters we want to replace.

Characters (which can form references itself) will be ignored: '0'-'9','a'-'f','A'-'F','x',';','#','&'.

Example: "32;x9" means that all spaces (a space has a value 32 decimal or 0x20 hexadecimal) and tabulators (a code 0x9) in xml file will be replaced by texts and 	 in the result.

21.2.5 Normalize spaces

This section is similar to the one which was described in section 20.8 `Normalize spaces`. Each schema can define its own parameters for space normalization.

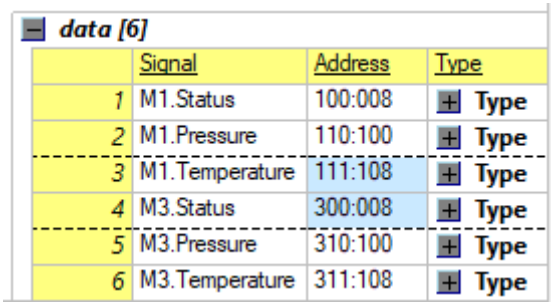
Moreover this normalization can be set to be done automatically before each saving by the checkbox - *Normalize spaces automatically before saving*.

Please note that spaces normalization will be done only in the table mode when the file is saved. That means that automatic normalization will not happen while the editor is in text mode or if you switch from table mode to text mode.

21.2.6 Miscellaneous section

Highlight table row(s) in mode: Standard / Compare / Validation

For better orientation in large tables it's possible to turn on highlighting selected row. Each mode has its own setting. By default this option is enabled only in a standard mode.



Word wrap in cells

This option can split (word wrap) longer cell's text into several lines. If 'word wrap' option is selected, the cells with longer texts increase their heights and try to display all texts.
If this option is unselected the longer text line is displayed in one line with an ellipsis character.

From version 1.6 there is a possibility to define the first table's row number which is displayed. By default it is number 1, but it can be redefined by the xml tag for the schema in the section Data:<FirstTableRowNumber>N</FirstTableRowNumber> where N is a new start number (e.g. 0). So the complete XPath is /XiMpLe/Schema[x]/Data/FirstTableRowNumber, where "x" is the index of the schema you want to modify.

Multi-line text can be collapsed

This option displays the collapse button for a text value which has more than 1 line if it's selected.

Horizontal scrollbar for editing

This option is related to the edit box which appears if we edit some cell [8 `Editing values`]. If text in the edit box is wider than the width of the box the horizontal bar under the edit box appears to enable scrolling the edited value.

Vertical scrollbar for editing

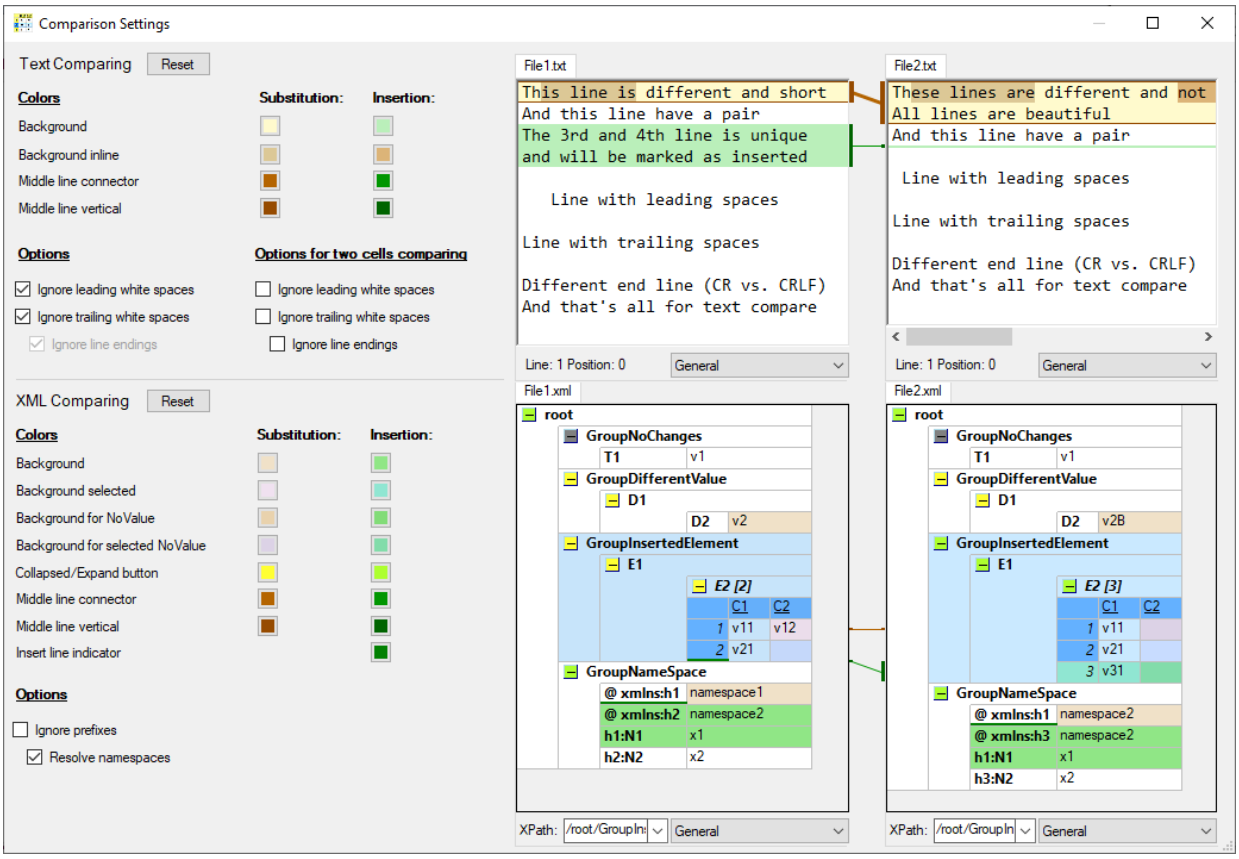
This option is related to the edit box which appears if we edit some cell [8 `Editing values`]. If text in the edit box has more rows than the height of the box the vertical bar next to the edit box appears to enable scrolling the edited value (or a wheel mouse can be used as well).

Enable highlighting

This option enables or disables highlighting of texts which are the same as currently selected text if it doesn't exceed one line and 128 characters in length.

Comparing...

Settings for text and xml comparison can be arranged by this command.



The upper half of the settings window is related to comparison of two texts, the bottom half is related to xml comparison.

Colours for displaying differences in texts and xml structures can be set here. The changes are displayed in the examples on the right side of the window.

For comparison purposes we recognize and define two kinds of differences:

a) *Substitution* – a part of text which can be replaced by another text on the adjacent position from the second file (to decrease number of differences between two files)

b) *Insertion* – a part of text which must be inserted to second file or deleted from the first file (to decrease number of differences between two files)

Text comparing settings:

- *Ignore leading white spaces*

If set then all spaces from the beginning of each line will be removed before comparing starts. For instance line " Hello" will be equivalent to line "Hello".

- *Ignore trailing white spaces*

If set then all spaces from the end of the line (included end line character(s)) will be removed before comparing starts. The line "Hello " will be equivalent to line "Hello" but not equivalent to " Hello".

- *Ignore line endings*

If set then line endings will be removed before comparing starts. Line endings might differ in different operating systems.
This option is automatically set if *Ignore trailing white spaces* option is active.

These three options are duplicated in the settings for "*two cells comparing*". It's a special function when comparing two xml files - there is a possibility to inspect two different cell values as texts (it's quite common that xml values are multiple lines strings and sometimes it's hard to see differences). For this functionality are used these duplicated settings.
[For example we want to ignore leading spaces when comparing text files but when we compare two multiline cells in xml structure we want to see every difference in between these two cells' values.]

- *Button "Reset"*

It sets the colours and options to default values.

Xml comparing settings:

- *Ignore prefixes*

By this setting elements' namespace prefixes will be ignored while comparing. That means that element <n1:Element1> will be equivalent to <n2:Element1> and it will be equivalent to <Element1>. There are two exceptions: namespace declaration prefix "xmlns:" and a special element <xml:space> are not ignored.

- *Resolve namespaces*

This option can be set only when "*Ignore prefixes*" is not set. By this option we make equivalent elements with different prefixes when they have same value.

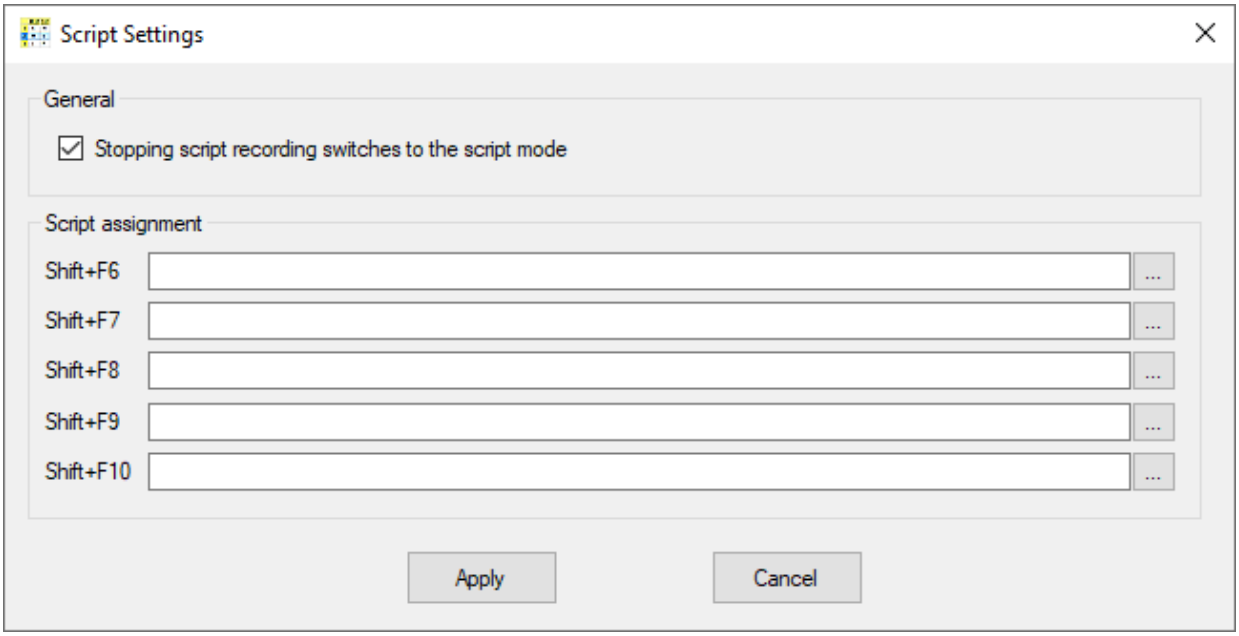
A simple example could be:
Let's have in the first file prefix h1 defined as xmlns:h1="NamespaceX", in the second file prefix h2 defined as xmlns:h2="NamespaceX". Both values are the same. Then an element <h1:Element1> from the first xml file will be considered equivalent to element <h2:Element1> from the second xml file because both prefixes are defined same.

- *Button "Reset"*

It sets the colours and options to default values.
If we do some changes in these settings, then a question dialog will appear (when we are closing the window) asking for saving and applying changes.
We can save and apply the settings (button 'Yes'), discard them (button 'No') or continue editing (button 'Cancel').
For comparison mode there is only one set of options (in contrast with standard mode where schemas allow multiple settings).

21.3 Scripting...

Here we can change the settings related to scripting.



General:

- *Stopping script recording switches to the script mode*

If it's set, then each stopping of the script recording will switch automatically to the script mode.

Script assignment:

- *Shift+Fx*

For key shortcuts Shift+F6, F7, F8, F9, F10 can be assigned scripts. The path for the script is expected. This will work only for registered users.

21.4 Set file association

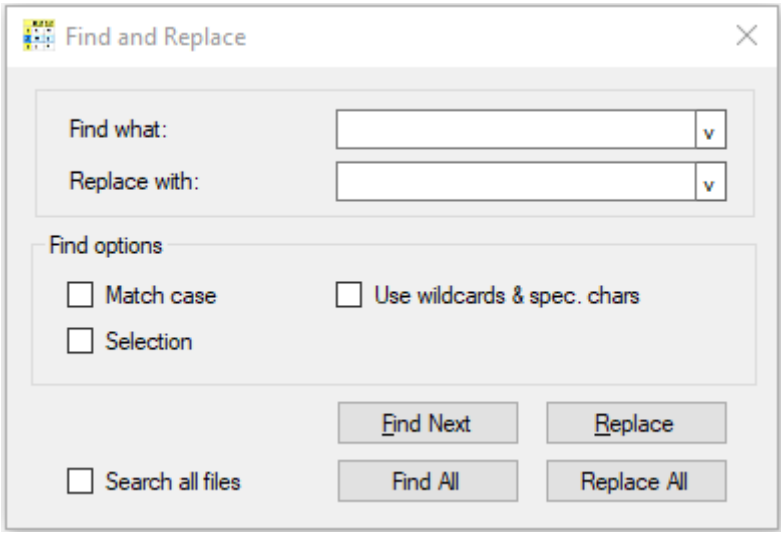
If we want to make association with the files which have the same file extension like the currently opened file we can use this option. In some cases if the file extension was already associated by a user in the system to some other program it doesn't have to be possible to overwrite this association. In such case a XiMpLe editor is added to the list of programs (associated with the extension) only.

22 Text Editor

Each loaded xml file can be switched to the text editor by the 'Text' tab in the left bottom corner. If you don't see the 'Text' tab please check the option 'Hide text tab if xml is well-formed' in General Settings [21.1].

In a text mode we can search and replace as well. We can use wildcards in the same way as was described in the chapter [20.3 `Find and Replace`] for xml editor. It's possible to search in the

selected area only and use lookup table and number modifiers for replacing (lookup table must be defined in the well-formed xml file).
Moreover we can search in all opened files which are in a text view (option Search all files) and we can list all found results in a table (Find All).



There is of course a standard possibility to Copy, Cut and Paste block of texts and an option to go to the specified line of a text. Changes can be reverted by Undo and Redo – this functionality is always active, the option for undo in General Settings doesn't affect text editor.

The editor supports bigger files and very long lines which can appear sometime in xml files.
For a better performance a mono-space font in the editor is recommended.

Text editor preserves a content of the file so in theory it's possible to use it also for editing binary files.

23 Tools Menu

23.1 Compare Mode

This command switches to the comparison mode (available from version 1.3). Comparison mode divides the main window into two halves and creates a toolbar with commands on top. Files can be loaded (or dragged and dropped) to the left and right half and compared as two texts or two xml files. Differences can be merged. A hybrid comparison between text and xml is not supported.

Commands from the main menu (e.g. open, save, save as, undo etc.) are valid also in the comparison mode but they are working on the left or on the right side according to which part is currently active and focused. Active part has a frame border around. Moreover, the main menu "File" label is changed to "File >" when right side is active and to "File <" when left side is active.

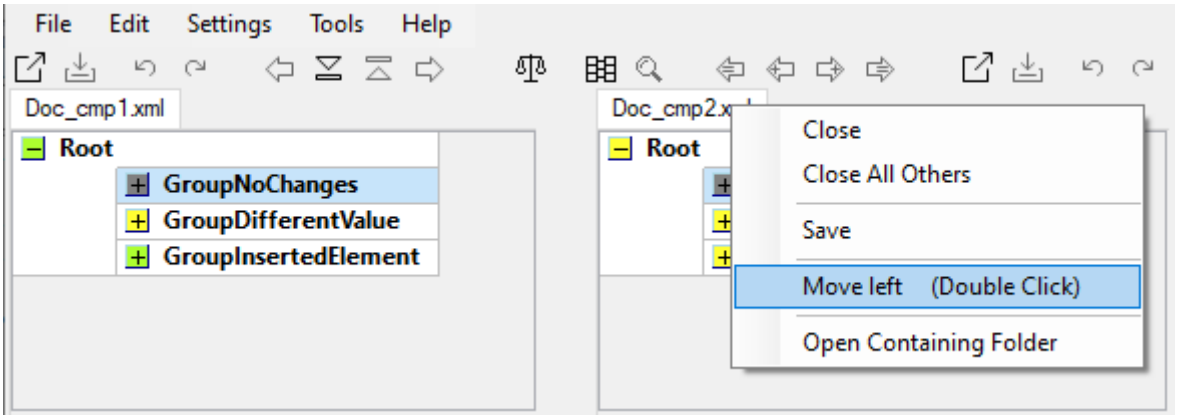
For text comparing there is used an algorithm of the longest common subsequence applied on the lines. The comparing supports bigger files and very long lines as well. It's possible to merge differences, show inline differences and go through them. Details will be described in the next chapter.

If both files are valid xml files, they can be compared in a grid view as two xml structures (as far as I know there is only one other program which supports this special kind of xml comparing in a grid view). When comparing two xml files the order of elements and attributes inside the same structure level is ignored. The strategy is to find same pairs of elements on the same level if they exist.
For the comparison of xml each cell (should) has some adjacent cell from the second xml. The element which is only on one side is adjacent to the place (cell) where such element should be inserted in the second xml structure - the element will create one way pair.

When any cell is selected the corresponding cell on the other side is selected as well. This could provide a better orientation.
Moreover the expand/collapse buttons are coloured to identify branches where are no changes or only different values without elements which must be inserted.

The mode supports merging differences, go through them, inspect in detail longer or multiline (different) values and reorder table rows according the adjacent table. Details will be described later.

Files can be moved from one side to another any time by double clicking the tab with the file name on the top. It will switch file position to other side. The second possibility is to use a command from a context menu for the file tab (see picture below).



Files can be arranged arbitrarily and of course there is a possibility to have more files open and compare them in combinations. For comparing they must be on different sides of the editor.

23.1.1 Text comparison

A toolbar for text comparison looks like this:

The icons have their own tooltips describing the functionality. Some of them can be disabled under certain circumstances.

- A(A'): Open a file to the left (right); it's also possible to drag and drop a file directly
- B(B'): Save file on the left (right) side

Note: If shortcut Ctrl+S is used then the file on the currently active side is saved.

- C (C'): Undo for left (right) side
- D (D'): Redo for left (right) side

- E(H): Merge current difference from right to left (from left to right)

Note: For these actions there is a key shortcut *Alt* + *Left* (*Alt* + *Right*)

- F: Go to next difference from the current position

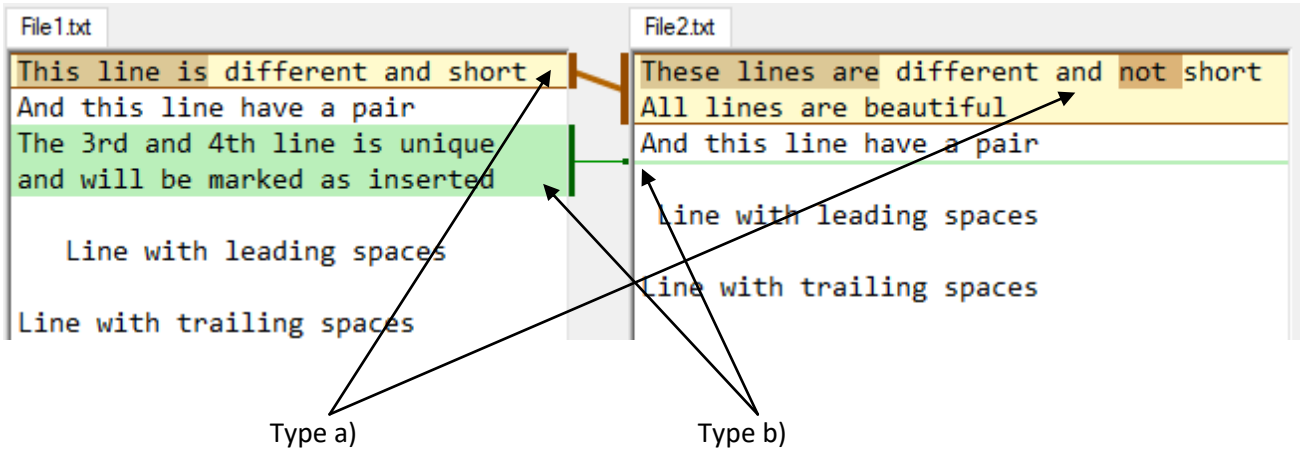
Note: For this action can be used also a shortcut *Alt* + *Down*.
If there is no following difference this icon is disabled.

- G: Go to previous difference from the current position

Note: For this action can be used also a shortcut *Alt* + *Up*.
If there is no precedent difference this icon is disabled.

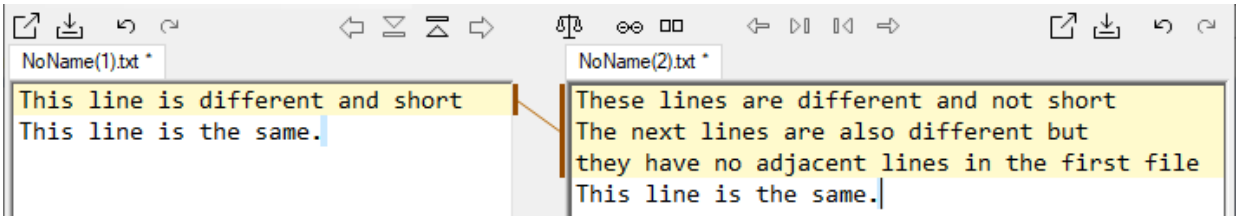
Between two files there can be two types of differences:

a) text (one or more lines) which can be replaced by some other text from the second file to decrease number of differences
b) text (one or more lines) which can be inserted into second file or deleted in the first file to decrease number of differences

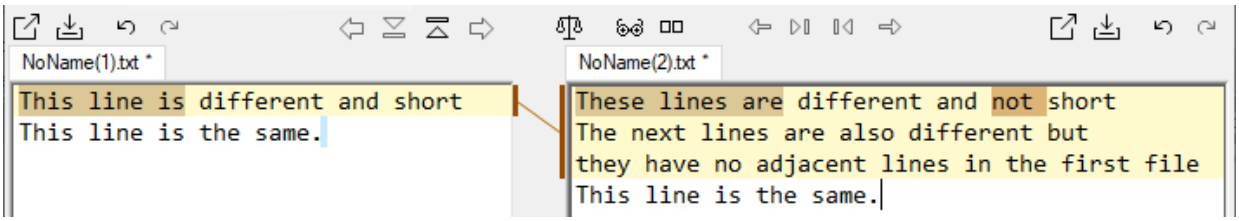


It's possible to move from one difference to another one in forward or backward direction (commands F or G), check visually the differences and if needed merge the difference from one side to another (commands E or H).
After each merging or changing the comparing is recalled and the result is updated.

- J: Compare files. If comparing is not executed automatically [see section 21.1 option *Compare files automatically*] this command will do it (shortcut *F5*).
- K: Turn on/off showing inline differences. If this option is 'On' the differences for related lines (in a substitution block) are highlighted.



Inline differences are 'Off'.



Inline differences are 'On'.

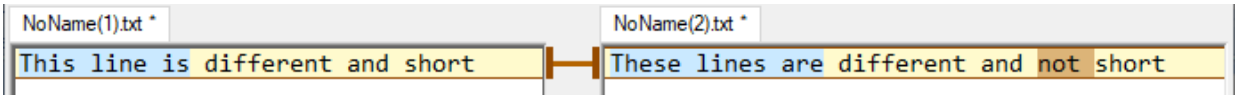
- L: Turn on/off independent scrolling. For dependent scrolling vertical scrollbar is bound with the second file and while moving through the file the second file is moving as well. For vertical movement doesn't matter if scrollbar is moved or if we use keyboards (e.g. PgDn, PgUp etc.). Horizontal movement is synchronized only if horizontal scrollbar is moving (it's not affected by cursor movements along the line).
- M(P): Merge line difference from right to left (from left to right). It's possible to go through particular differences on the line – see next commands.
  In the screenshot above the first line have two line differences: "This line is" vs. "These lines are" and "" (empty string) vs. "not ".
  The found and selected difference can be merged from one side to another.

  If inline differences are 'Off' we must firstly find the inline difference and then we can do inline merge. If inline differences are 'On' for inline merging is sufficient if we place the cursor on both sides to the same difference.

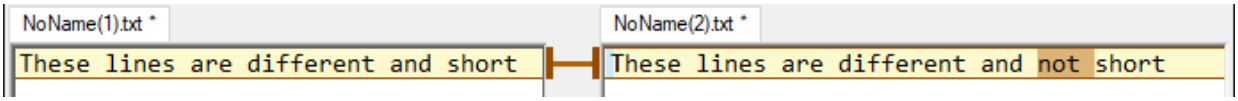
Note: for these actions can be used shortcuts *Shift* + *Alt* + *Left* or *Shift* + *Alt* + *Right*

- N(O): Find next (previous) line difference on the line with cursor.

Note: for these actions can be used shortcuts *Shift* + *Alt* + *Down* or *Shift* + *Alt* + *Up*



Found the first difference on the line by command N (when the cursor is at the beginning of the line). Difference is selected on both sides.



This is the situation after merging the line difference from right to left by command M.

23.1.2 Xml comparison

A toolbar for xml comparison looks like this:

The icons have their own tooltips describing the functionality. Some of them can be disabled under certain circumstances.

- Icons A-D, A'-D' have the same functionality like for text comparison (see their descriptions in the previous chapter).
- Icons E, F, G and H have the same shortcuts and meaning but applied on xml structure.

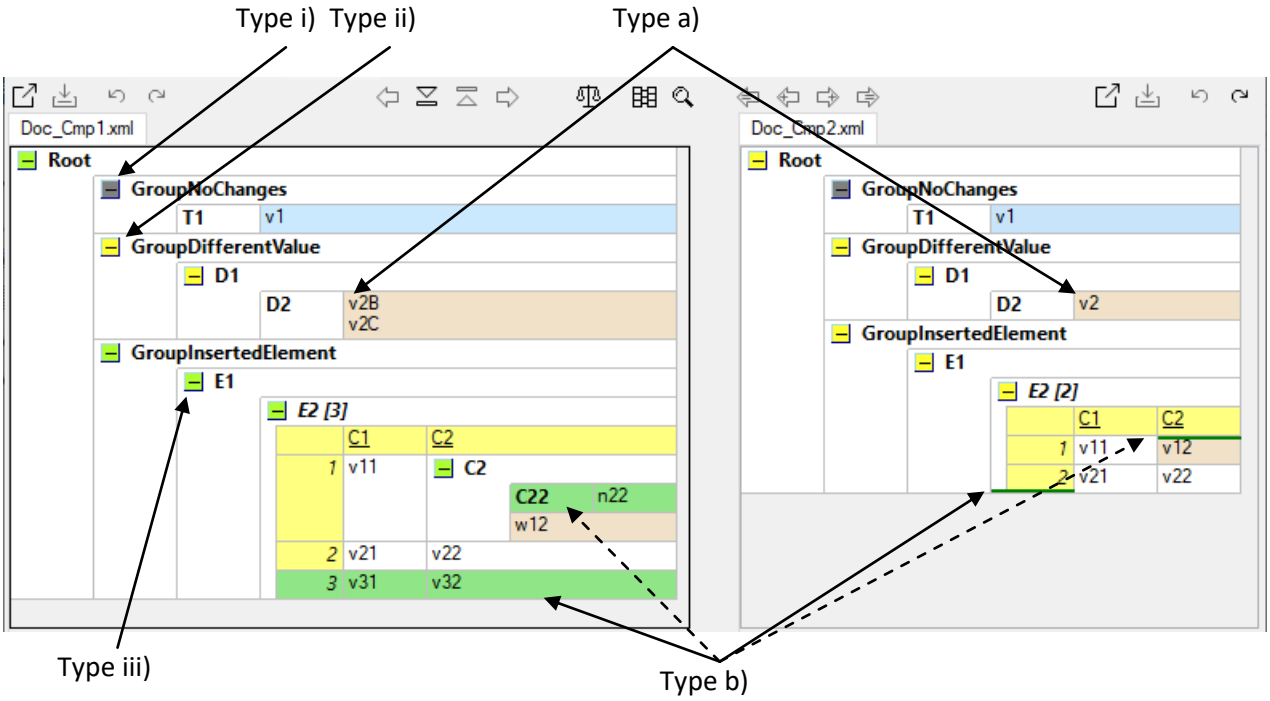
Note: Shortcuts *Alt* + *Up/Down/Left/Right* are used in standard mode for table rows' and columns' manipulations. In the mode of comparison these shortcuts are occupied by frequently used

functions for finding and merging differences. Therefore, original shortcuts for rows' and columns' manipulations require additional key *Ctrl* (e.g. instead of *Alt* + *Down* there is a shortcut *Ctrl* + *Alt* + *Down*) in the comparison mode.

See also the table of shortcuts [section 27].

Between xml files there can be two types of differences:

a) Different value – in this case one value is replaced by another one from the paired cell of the second file
b) Different element – this means that some element is missing in the second file and to decrease number of differences this element must be inserted to the second file or deleted from the first file



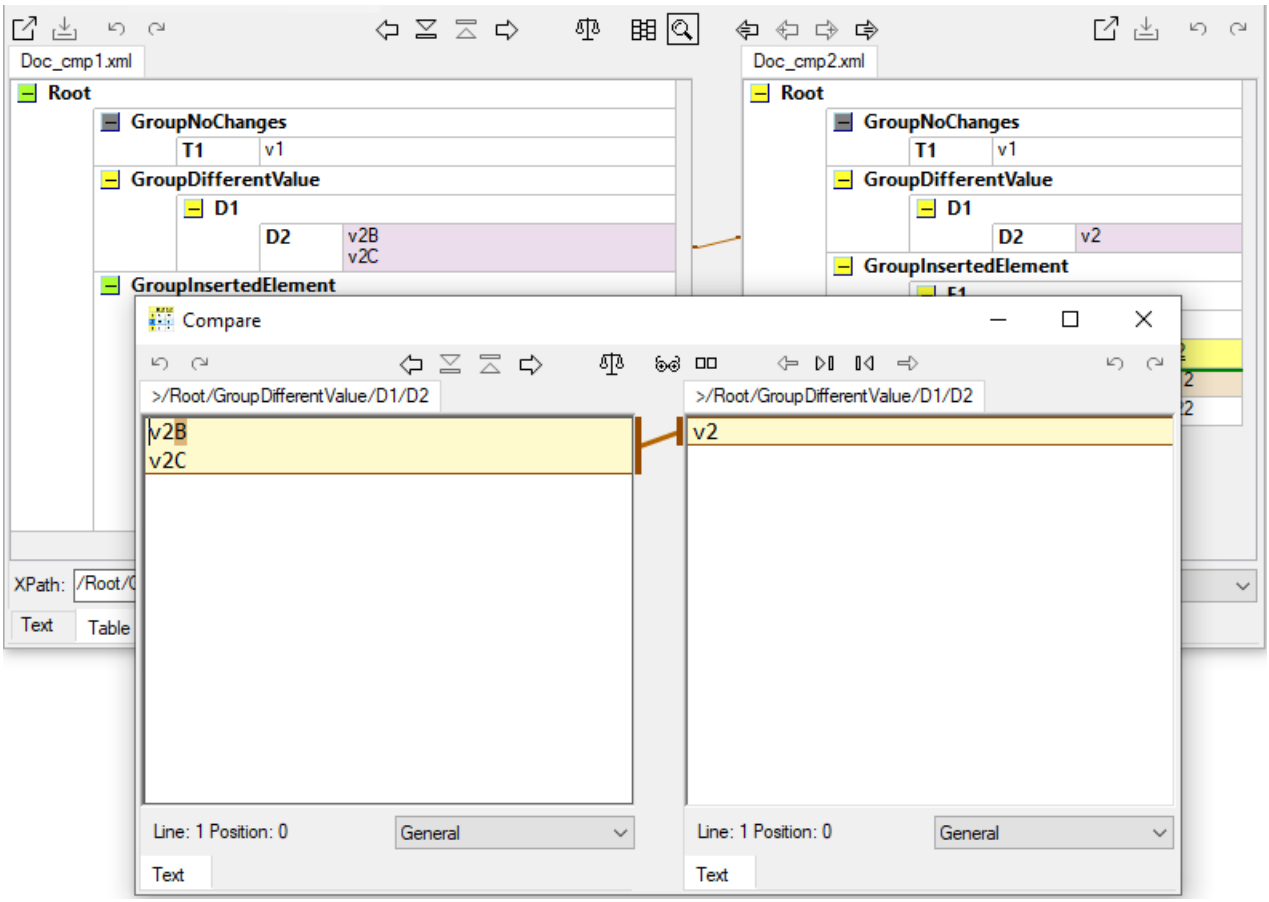
It's possible to find and move from one difference to another one in forward or backward direction (commands F or G), check visually the differences and if needed merge the difference(s) from one side to another (commands E or H).
After each change the comparing is recalled and the result is updated.

The colour of collapse/expand button carries information about nested data. These colours are customizable in the comparison settings [see section 21.2.5] but for our description we use the default colours (see also a picture above as an example):

i) Gray colour means that inside the branch there are no differences at all
ii) Yellow colour indicates that inside there are only different values (at least one) and that there are no different elements for insertion
iii) Green colour is used if inside is at least one different element for insertion (there may or may not be also different values in this case)

This could give a basic overview about differences between files.

- J: Compare files. If comparing is not executed automatically [see section 21.1 option *Compare files automatically*] this command will do it (shortcut *F5*).
- L': Turn on/off independent selection. For dependent selection each cell is bound with the cell in the second file.
- Q: Compare two different values in details as texts (shortcut *Alt + F2*). The values in xml elements could be quite complex (long or multi-lines) texts and sometimes it's a little bit unclear what the difference between related cells is. For such case we can use this command and invoke a sub-window which compares a selected value pair in the text mode (see picture).



Texts can be modified, edited and merged like for two text files (there is no open or save icon naturally). The result will be then used in the appropriate cells. There is no cancel in this window, but undo can be used while editing texts or afterwards in xml (in case we want return changes back).

- R(U): Merge selected difference(s) from right to left (from left to right) but merge only different values (type a) and ignore different elements (type b).
- S(T): Merge selected difference(s) from right to left (from left to right) but merge only different elements (type b) and ignore different elements (type a).

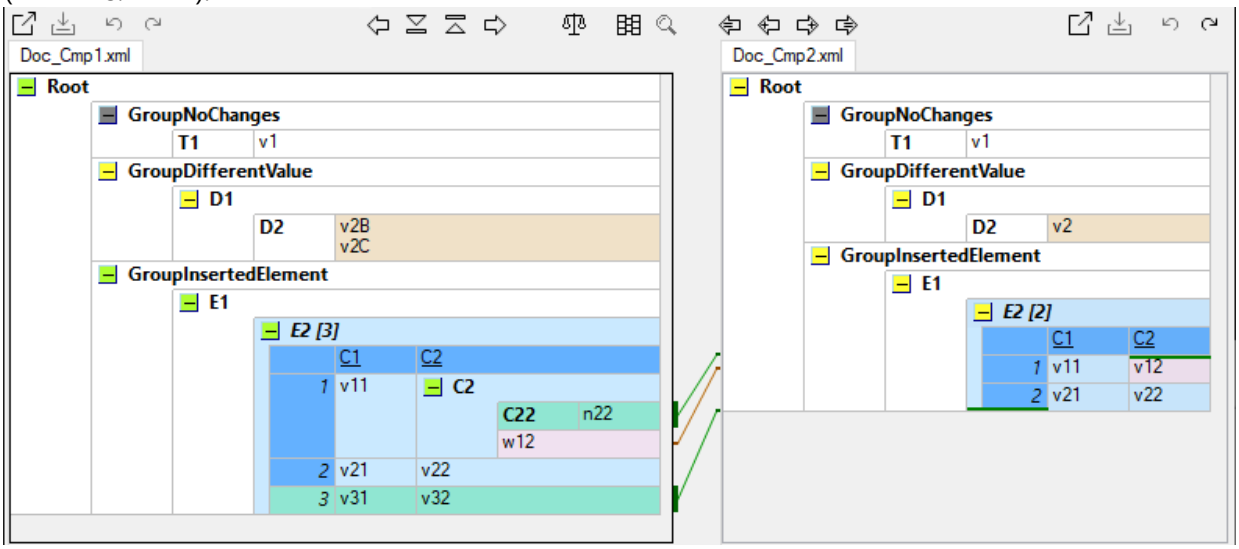
Note: For a single difference these functions have no practical use because there is no mixture of the types.

These commands will become more interesting and useful in the enhanced merging when more differences of different types are in the selected area (see section 23.1.2.2).

23.1.2.1 Middle part

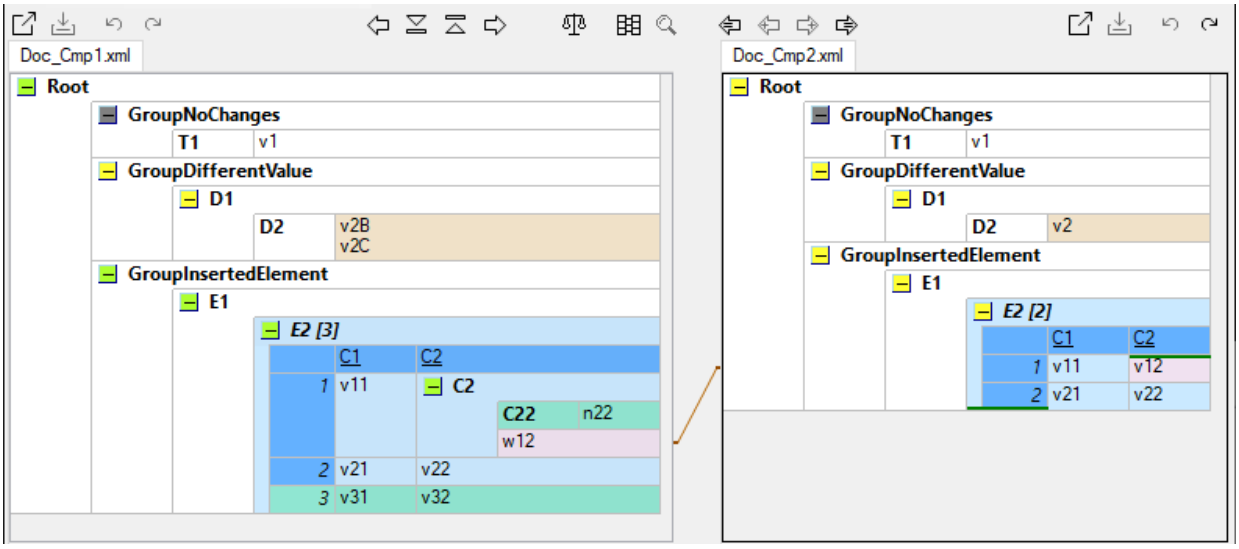
In between files there is an area in the middle which is used for lines highlighting the relations between differences. When comparing in text mode the situation is quite straightforward because lines aren't crossing each other.
Fox xml structures the situation could be much more complicated and displaying all lines is often a mess. That's why lines in the middle part are displayed only for selected cells and only with relation to the active side.

It will be clear in the next example:
On the picture below the left part is active and focused (notice the black framework on the left side). Three lines are going from left side to the right side – two are green for inserting elements (C22 and 3rd table row) which are oriented from left to right, one line is brown for different value ("w12" vs. "v12").



Situation 1

Now the same situation but the right side is focused (next picture). To switch between one side to another can be used a shortcut *Shift* + *Tab*. We see only one brown line for different value. The positions for inserting cells (green horizontal lines inside the table "E2[2]") play no role now.



Situation 2

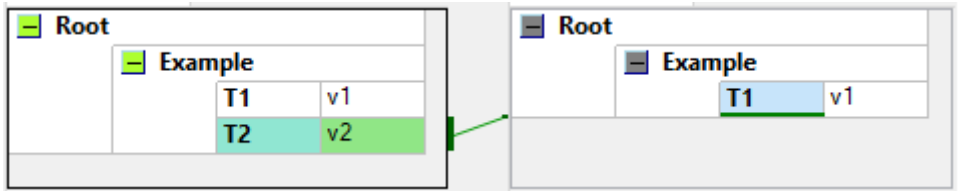
We can see that there is no symmetry in general (the symmetry is held only for different values) and it depends on the direction (on the active side).

23.1.2.2 Enhanced merging of xml differences

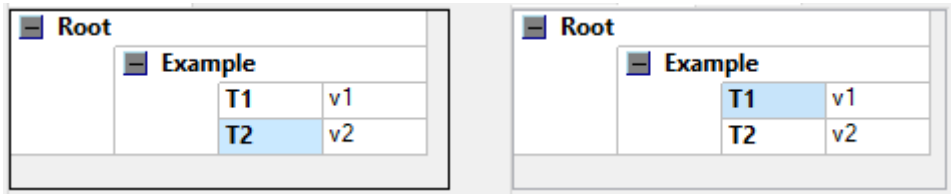
The editor provides a possibility to merge more than one simple difference at the same time. A block of cells (a table, columns, some part of the table etc.) can be selected and merged all together. But for this benefit there is one limitation which will be described in next two points.

- For a single difference it doesn't matter on which side we are currently staying - we can merge a difference (for instance from left to right) with the same result if the active side is left or right.

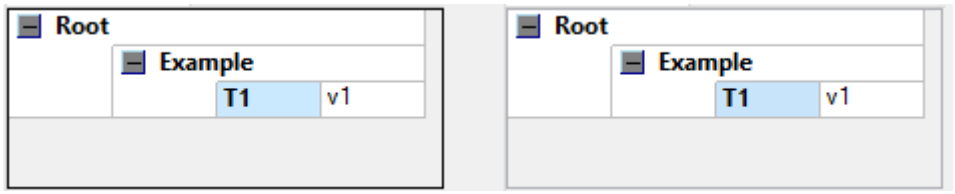
For instance, let's have a simple element for inserting on the left side and a place for insertion on the right side:



If we are on the left side and *merge from left to right,* then the element will be *inserted to the right* side; if we are on the right side and *merge from left to right,* then the element from left side will be *inserted to the right* side:



If we are on the left side and *merge from right to left,* then the element will be *deleted on the left* side; if we are on the right side and *merge from right to left,* then the element will be *deleted on the left* side:

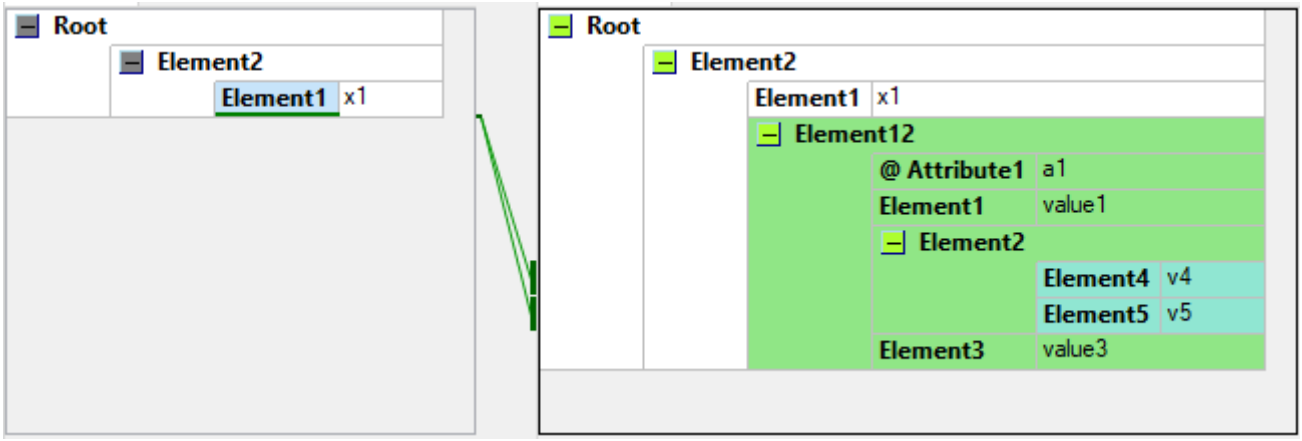


- For a complex block of cells the merging is done only for selected cells – if there are some positions for inserting elements in the selection they have no importance for merging.
  In such case focus must be on the other side if we want merge and insert such cells.

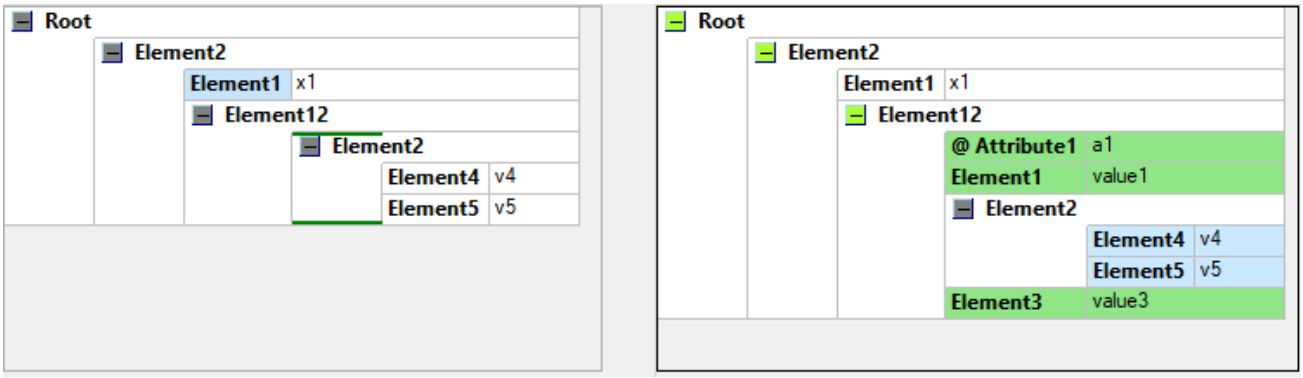
For explanation we will take as an example picture above named "Situation 2" – *right side is active* and the block "E2[2]" is selected. There are two positions (green horizontal lines) for inserting elements from left side. If we try to merge from left to right, only a different value "w12" will be merged to the right side (according lines in the middle part). The similar result we get if we try to merge from right to left – again only the different value is affected.
If we want to merge inserted elements on the left side to the right we must *stay on the left side* (see "Situation 1"). From this position merging from left to right will insert elements to the right side and merging from right to left will delete inserted elements on the left side.

The selected block can contain both types of differences (a and b) and the functions marked as R, S, T, U become now more useful – they give us the possibility to merge only different values (type a) or only different elements (type b).
A typical case could be a table with plenty of different values and also plenty of rows for inserting – and we want to merge only different values.

Another feature which is available is the possibility to insert only a part of a different branch. Let's have this simple example where the full Element12 is missing on the left side. If we want merge only Element4 and Element5 to the left side we can select these two elements inside and...



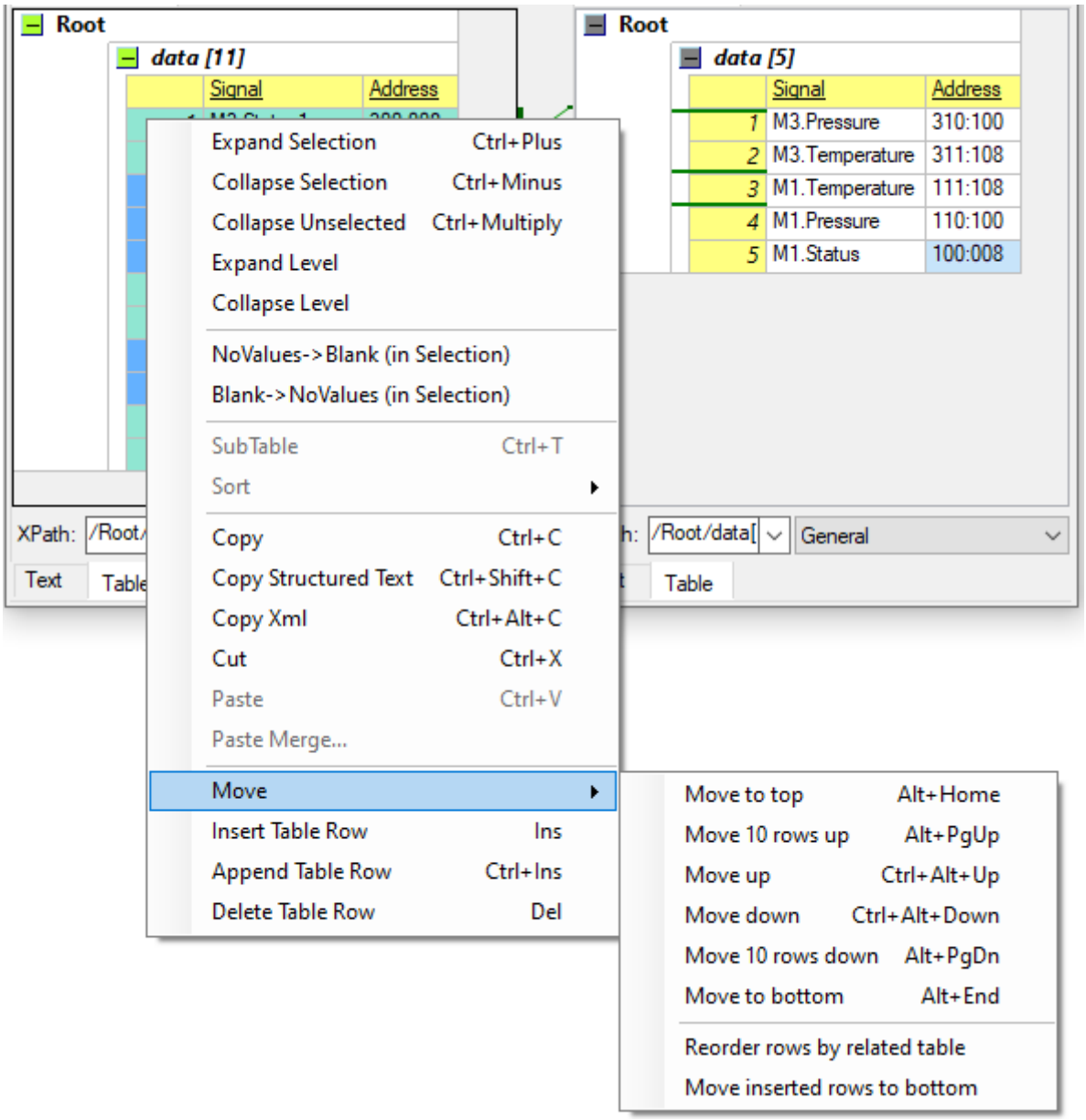
...do merge from right to left



Selected elements are inserted with a complete structure which is holding inserted elements (in this case Element12/Element2/).

23.1.2.3 Two additional features

For xml comparing there are two additional and a little bit hidden functions which should be mentioned. They can be found in the context menu for moving table rows.



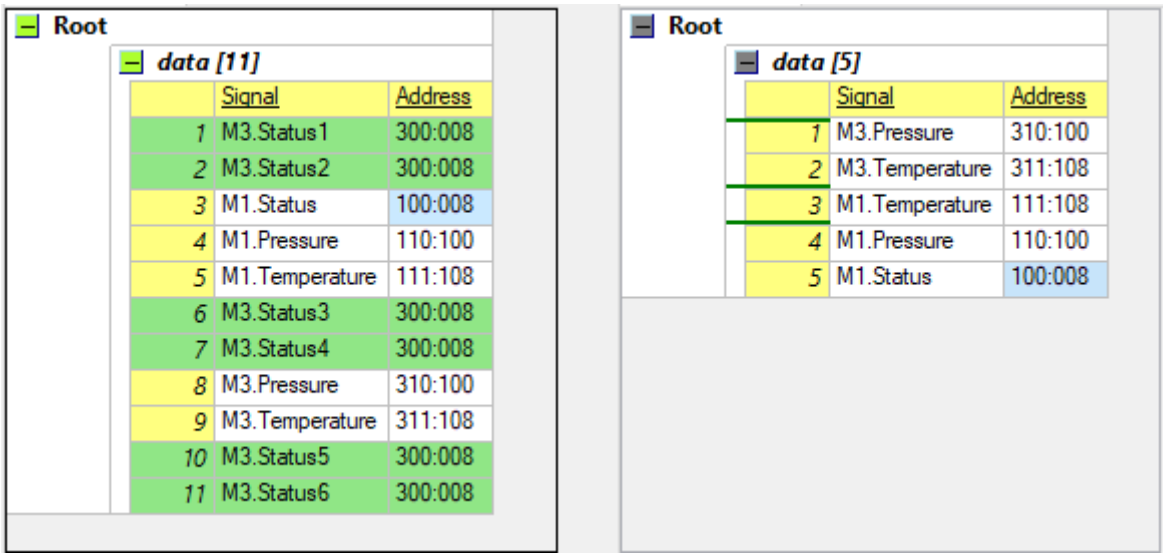
- *Reorder rows by related table* – this command will change the order of the table's rows (depends on which rows are selected) according to the table which is in a pair. It has sense only if the paired table contains rows which are the same. If tables contain completely different rows this function keeps the order unchanged.

Note: Table rows which are different and must be inserted are virtually bound to the first previous row which has a pair in the adjacent table of the second file. When such row is reordered by this function it will take all rows under which have no pairs with. See also example below.

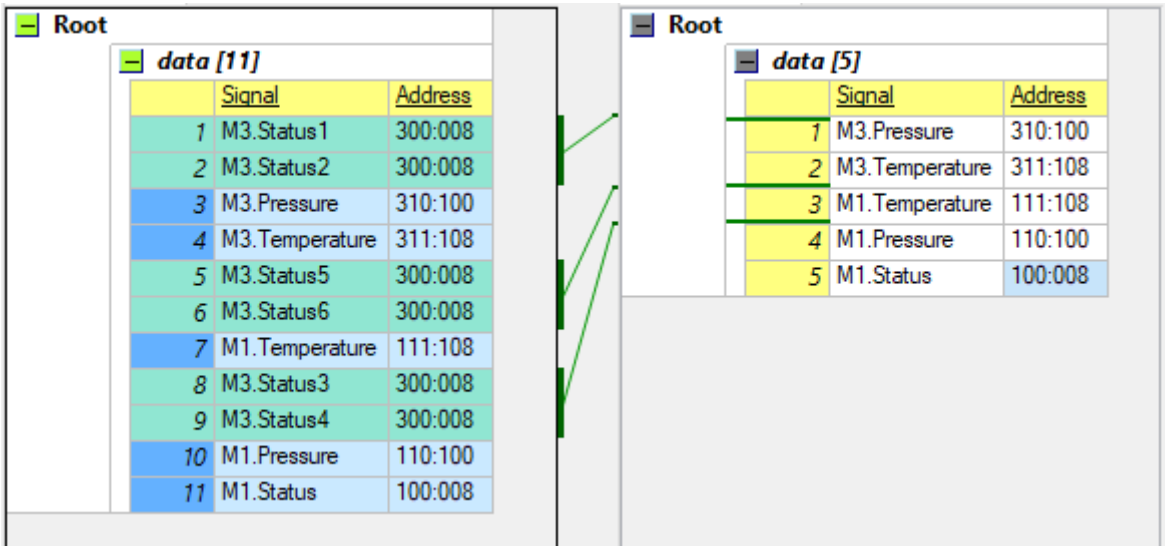
If we want to avoid such behaviour, we can move inserted rows to bottom at first or try to exclude such rows from the selection.

- *Move inserted rows to bottom* – this command will move all table rows in selection, which are "marked" as inserted, to the end of the selected block. If the whole table is selected, they will be moved to the end of the table.

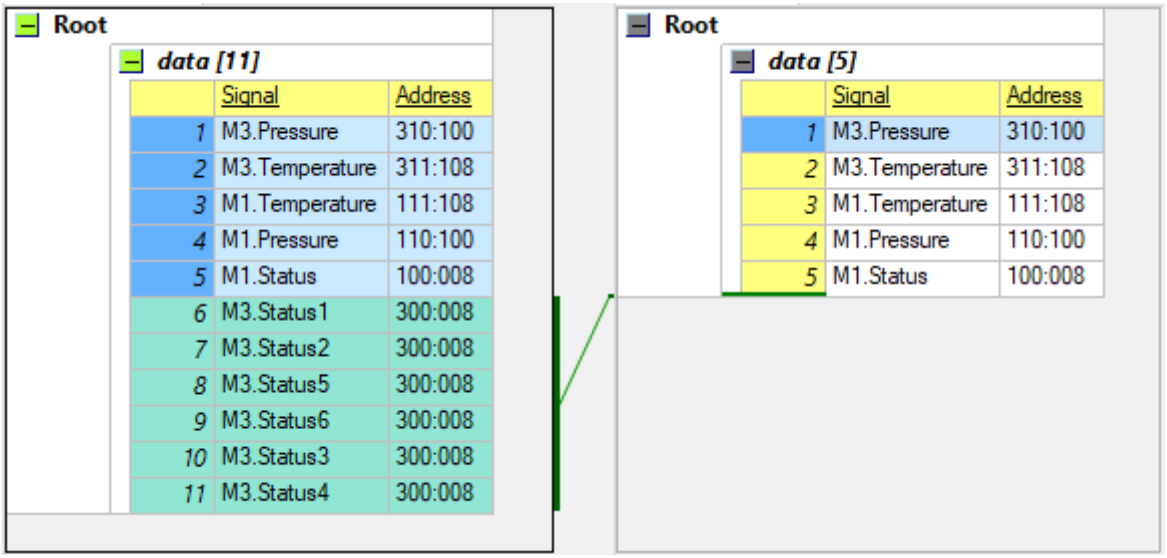
Example: Let's have these two tables.



Let's select the whole table "data[11]" and use the function "Reorder rows by related table". All rows from "data[11]" which have pairs in "data[5]" will be reordered. In this case rows number 3, 4, 5, 8, 9 have pairs in the second table. So these five rows (on the left) will be ordered by the ordering in the second table (on the right).
Moreover, rows which are different will be moved together with the leading row. That means rows number 6, 7 will be moved together with row number 5 and rows 10, 11 will be moved together with row number 9. Rows 1 and 2 have no leading row which can change a position so they will stay where they are. This will be the result:



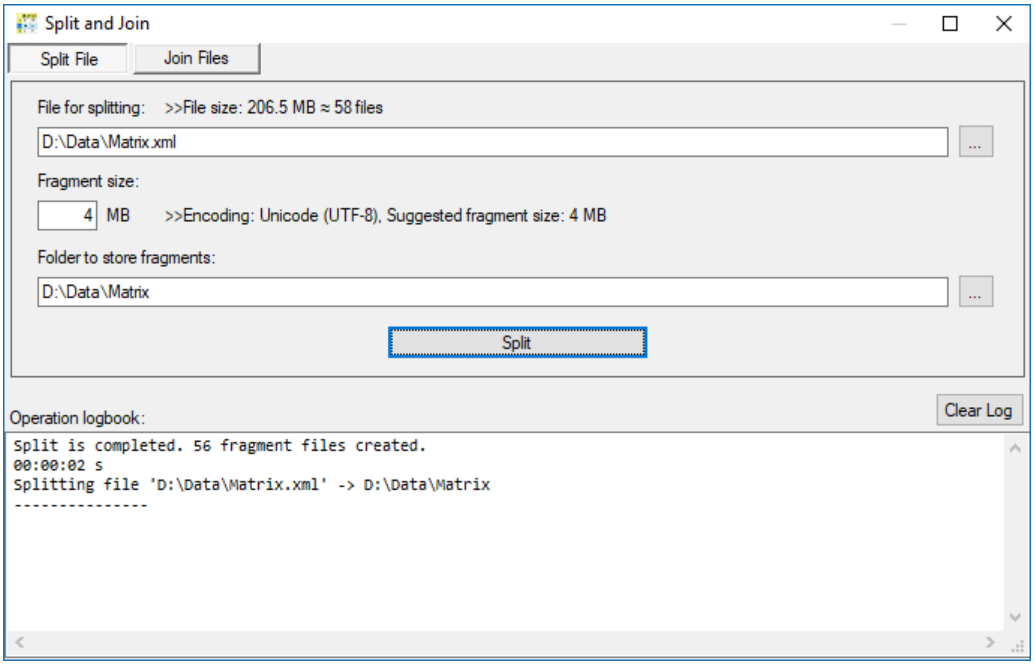
Now let's call the function "Move inserted rows to bottom" and keep the whole table selected.



23.2 Split XML File…

There are some hardware and system limits for editing xml file and it can happen that we want to edit an xml file which is too big to be loaded in memory or it's loaded but the editor is too slow for manipulations. For these cases there exists a solution - split the original xml file into fragments, edit them and at the end put them back all together. All these steps can be realised in XiMpLe.

Many of xml splitting programs require from the user some information about internal xml structure (like a tag's name on some level depth). Don't worry, we won't need anything like this – if we have several gigabytes' file we don't want to peek inside the giant pit.



For xml file splitting we need to specify:

- *File for splitting*

Specify the xml file which we want to split. For a selected file there appears additional information about the file size and a rough estimate about number of fragments which will be created by splitting. The number of fragments depends mainly on the fragment size value.
A file can be dragged and dropped to this field.

- *Fragment size*

This is the approximate maximum size of a fragment. In general, the size of each fragment can vary because the program tries to find locally the most appropriate place (with a minimum depth) for splitting. The size is expected in megabytes and can be entered in the range from 0.1 MB to 100 MB. For a selected file there appears additional information about the detecting coding of the input file and a suggested fragment size (it depends on the coding).

- *Folder to store fragments*

Created fragments will be stored on the disk in the folder which will be specified here. If such directory doesn't exist, it will be created before the splitting without asking.
A folder can be dragged and dropped to this field.

- *Button "Split"*

It starts the splitting action and fragments of requested size will be creating in the desired destination folder. This process can take some time that's why a progress bar appears with some time estimation and the button for cancelling the action.

If the action is cancelled already created fragment files are not deleted!
After finishing the action, a basic info appears in the log.

If some unrecoverable error appears during the process the splitting is interrupted, and the reason appears in the log. The good news is that the fragment where an error appears should exist in the destination folder (and its size is small enough to be edited) followed by the whole rest of the unprocessed file. In such case we have a chance to fix the problem in the fragment. After fixing we can use joiner (see next chapter) to put these parts back together and we can try to repeat the splitting process again (to find another one error 😉). Anyway, we have a way to identify and repair bugs in a big file.

23.2.1 Additional information

To be able to put later all fragments back together a special xml file is created in the destination folder. Its name is in the form "FileName*.join.xml*" and it contains information about fragments' paths and names and their orders. The structure of this file is simple but important and it will be discussed in the next chapter.

Each fragment except the last one will get an appendix at the end. It's an xml comment and will look like this "*<!-- SplitDepth=3 (used for joining) -->*". The reason for this appendix can be demonstrated by an example:

Let's have two different xml file:

File 1:

```
<root>
   <section>
        <element1>value1</element1>
        <element2>value2</element2>
   </section>
</root>
```

File 2:

```
<root>
   <section>
        <element1>value1</element1>
   </section>
   <section>
        <element2>value2</element2>
   </section>
</root>
```

Both are well-formed and different xml files. And both can be split into same two fragments:

Fragment 1:

```
<root>
   <section>
        <element1>value1</element1>
   </section>
</root>
```

Fragment 2:

```
<root>
   <section>
        <element2>value2</element2>
   </section>
</root>
```

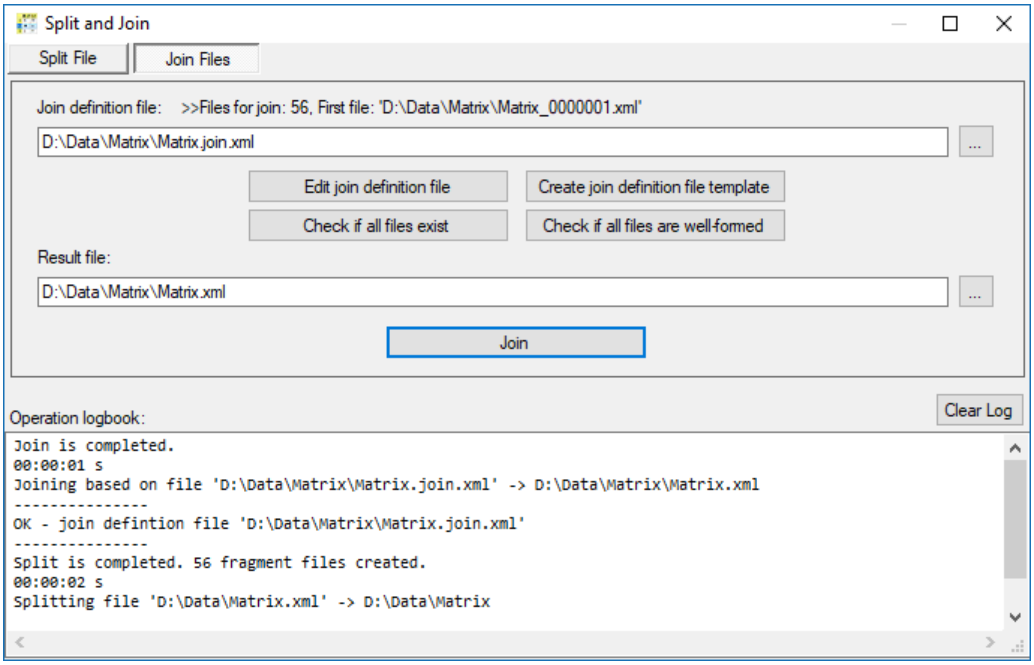
To distinguish between these two cases, we have to add additional information about the depth which will be used when fragments are joining together.
For the first example file there will be added a comment *<!-- SplitDepth=2 -->* which tells the program that last two elements (*root* and *section*) are common and should be looked for in the next fragment and that they are working together.
For the second example file there will be added a comment *<!-- SplitDepth=1 -->* which tells the program that only the root element is common.

By using this information, the XML joiner tool will be able to restore the file back to the original state.

If no comment with SplitDepth information is found at the end of the fragment, then the maximum possible depth is taken. That means that the program compares the element names from the end of a fragment and from the beginning of the next fragment as long as these names are matching together.

23.3 Join XML Files…

In the previous chapter we have successfully split the file and maybe edited some fragments and now we would like to put it all back together to sweep the tracks.



For joining files together, we will need:

- *Join definition file*

This is the file usually created during the splitting. A file can be dragged and dropped to this field. It contains information which files and in which order should be put together. It's a simple xml file with this structure (this is only an example):

```
<XiMpLe>
        <Files>
                <Path>C:\Temp\FragmentFolder</Path>
                <BaseName>BaseFileName_*.xml</BaseName>
                <IndexStart>1</IndexStart>
                <IndexEnd>10</IndexEnd>
                <IndexDigits>7</IndexDigits>
        </Files>
        ... the section <Files> can repeat ...
</XiMpLe>
```

In this example will be used files C:*Temp\FragmentFolder\BaseFileName_*.xml* where the '*' character will be replaced gradually by numbers from 1 up 10 and formatted to 7 digits (0000001 up 0000010).

- If *Path* is not specified, the previous path is used. If the first path is not specified, the folder with the join definition file is taken.
- If *BaseName* is not specified, the record is skipped.
- If *IndexStart* or *IndexEnd* is not specified and in the *BaseName* is used a character '*', the record is skipped.

- If *IndexStart* or *IndexEnd* is specified and in the *BaseName* is not used a character '*' the indexes are ignored and only one instance of a file is taken.
- It's allowed also decreasing order of indexes (e.g. from 10 to 1).
- If IndexDigits is not specified, the previous IndexDigits is used. If the first one is not specified, the default value is taken (7).
- *Result file*

This is the path and the name of a file which we want to create from the fragments. A file can be dragged and dropped to this field. If the file already exists a user is asked if it can be overwritten.

Important: Please note that for joining purposes two elements with the same name but different attributes or different attribute values are considered as different. If some attribute is missing, then it's considered as a subset. For example:

Let's have these three xml fragments:

F1: <root a="1" b="2">content1</root>

F2: <root a="1">content2</root>

F3: <root b="2">content3</root>

Element root from F2 (or F3) is a subset of F1, element roots from F2 and F3 are different.

If we join F1 and F2 the same root element is kept and schematically we can write the result:

<root a="1" b="2">content1 content2</root>

If we join F1 and F2 and F3 the same root element is kept (F3 is a subset of a leading root element F1) and schematically we can write the result:

<root a="1" b="2">content1 content2 content3</root>

If we join F2 and F3 then we get the result (the root of F3 is not a subset of F2 so it must be included as it's):

<root a="1">content2<root b="2">content3</root></root>

- *Button "Join"*

It starts the joining action and fragments are put together in the order specified by join definition file. The process can take some time that's why a progress bar appears with some time estimation and the button for cancelling the action.

If some file is not existed, info in the log appears and it's skipped to the next fragment. A user can interrupt this action by a cancel button.

After finishing the action, a basic info appears in the log.

If some unrecoverable error appears during the process, the joining is interrupted, and the reason appears in the log.

- *Button "Edit join definition file"*

Put the currently selected file into XiMpLe editor for editing.

- *Button "Create join definition file template"*

It creates a template for a definition file and put it into XiMpLe editor for filling. It can be used if we don't have or lost the original definition file created during the split.

In principal we can join any xml file with anyone else file and it doesn't matter if these files were created by a splitting tool or not. Just create a new definition file and execute it. We can create only a subset of the original file if we want. The system was designed to be flexible.

- *Button "Check if all files exist"*

Check if all related files we want to use for joining exist. All files are also listed at the same time in the log so we can check if the files are in expecting order. The list is updating from the bottom to the top.

- *Button "Check if all files are well-formed"*

Check if all fragment files are well-formed xml files. This request can take some time that's why a progress bar appears with some time estimation and the button for cancelling the action.

- *Button "Clear Log"*

Clear the operation logbook information on the bottom.

23.3.1 Split & Join properties

Splitting and joining operation were designed in the way to have (or at least should have) some useful and interesting properties:

1) Well-formed preservation:
   a) If the original file is well-formed xml file, then all split fragments are well-formed xml files.
      Consequence: We can edit them in XiMpLe table view editor.

   b) If all split fragments are well-formed xml files and no multiple roots are created during the joining process, then the result xml file is also well-formed.

      Remark: Multiple roots are reported during the joining so we can easily detect this fact. The multiple root will be created during the joining if and only if SplitDepth=0 is declared at the end of some fragment.
      Consequence: These two points give us the recipe how to verify if the original xml file is well-formed. If we can split the original file and there is no warning about multiple root elements, then the original xml file is well-formed if and only if all fragments are well-formed.

2) Invertibility: If the original file is an xml file which was split then joining of not modified fragments will give the file which is the same as original one.

3) Associativity for files with the same root element: Let's split file A into fragments B, C, D. Let's join fragments B and C to E, fragments C and D to F. Then joining E and D will get the same file as joining B and F and the same file as A (E+D=B+F=A).

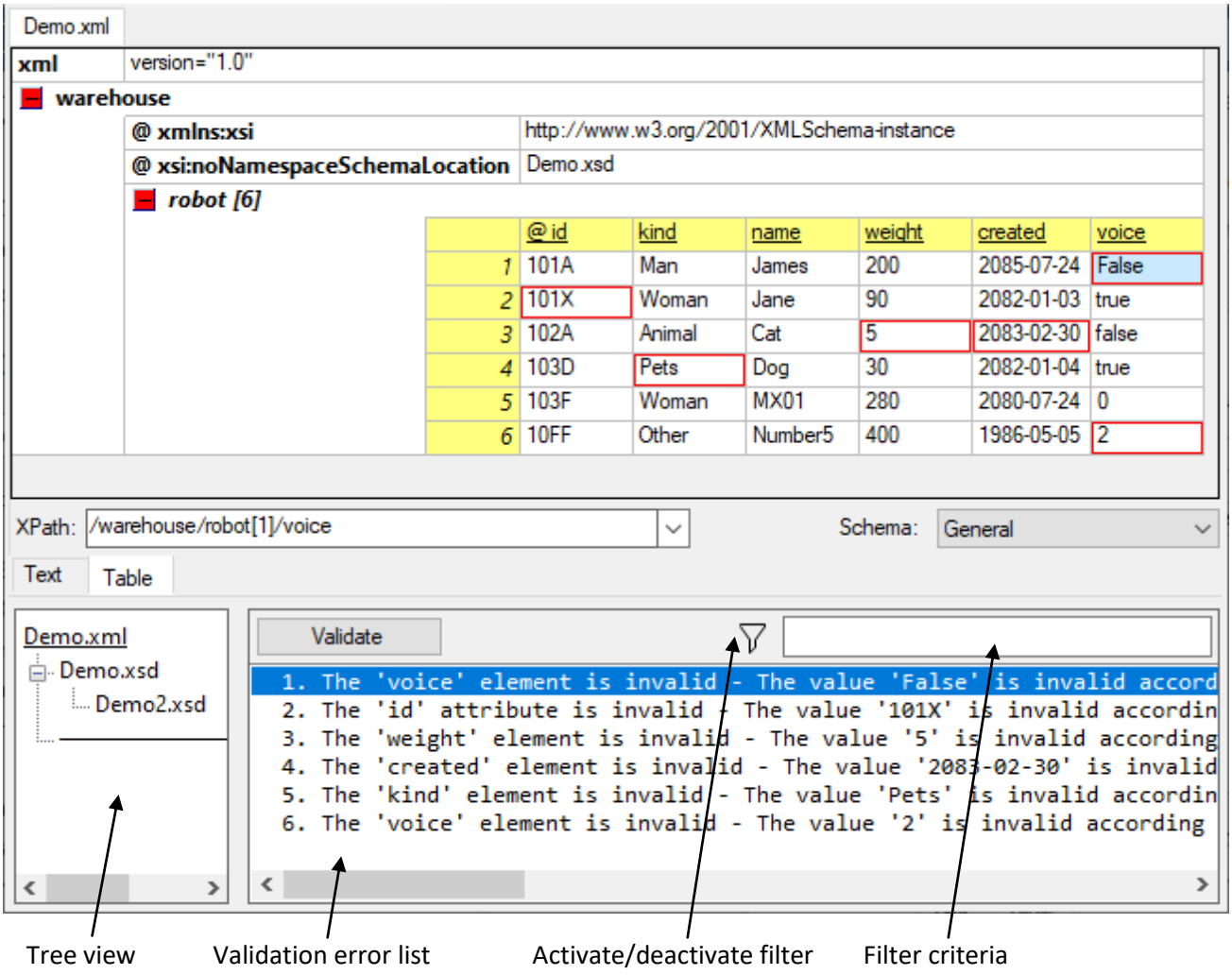
23.4 Validation Mode

A new feature in version 1.5 provides the mode for validating xml files by xsd schemas.

The dtd schemas are not supported and if they are used, the validation skips them. Generally, it should be possible to convert dtd schema to xsd schema by an external tool.

Note: Validation mode and Compare mode are mutually exclusive. They can't be used both at the same time. When one mode is selected the other one is ended if it's active.

Validation mode adds two new areas at the bottom of the application.



Xml file and its schema validation

On the left side is a tree view which displays used xsd schemas and their dependencies (a schema can include other schemas). Details are described in section [23.4.1].

On the right side is a list with found errors. Details are described in section [23.4.2].

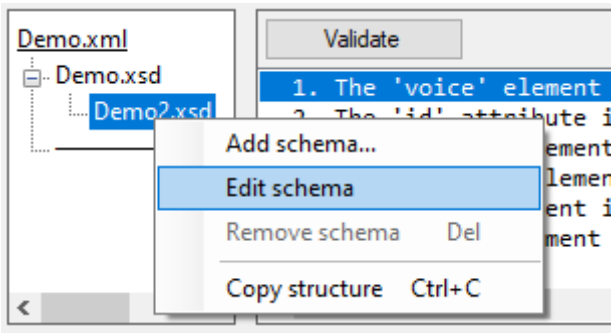
The validation can be performed in the table view and text view as well.

23.4.1 Tree view

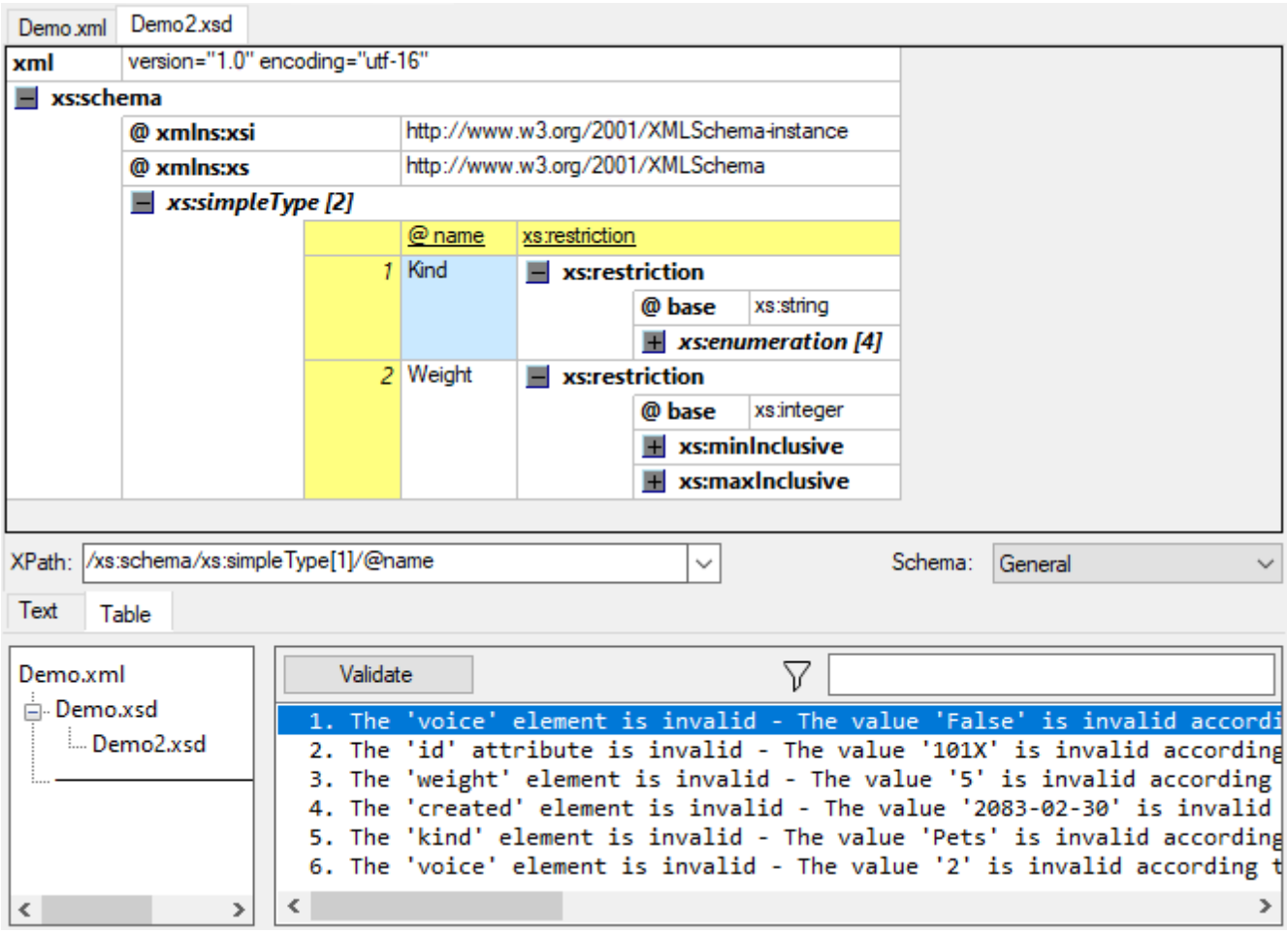
In the tree view we can see which schemas are used for the xml file and what are their dependencies (a schema can include or import another schema as its child).

Top element is the name of the file which is currently validated and for which the errors are displayed. Be aware that validated file can be different from a file which is active in the editor. For instance, if we are validating xml file which uses some schema and the schema itself is correct and without errors, we will see original list of errors instead of a message that there is no error.

We will demonstrate it on example – let's have a situation from the picture above. We see there are some errors. Notice that the top root tree element 'Demo.xml' is underlined. Let's invoke command 'Edit schema' for 'Demo2.xsd' from the tree view by context menu.

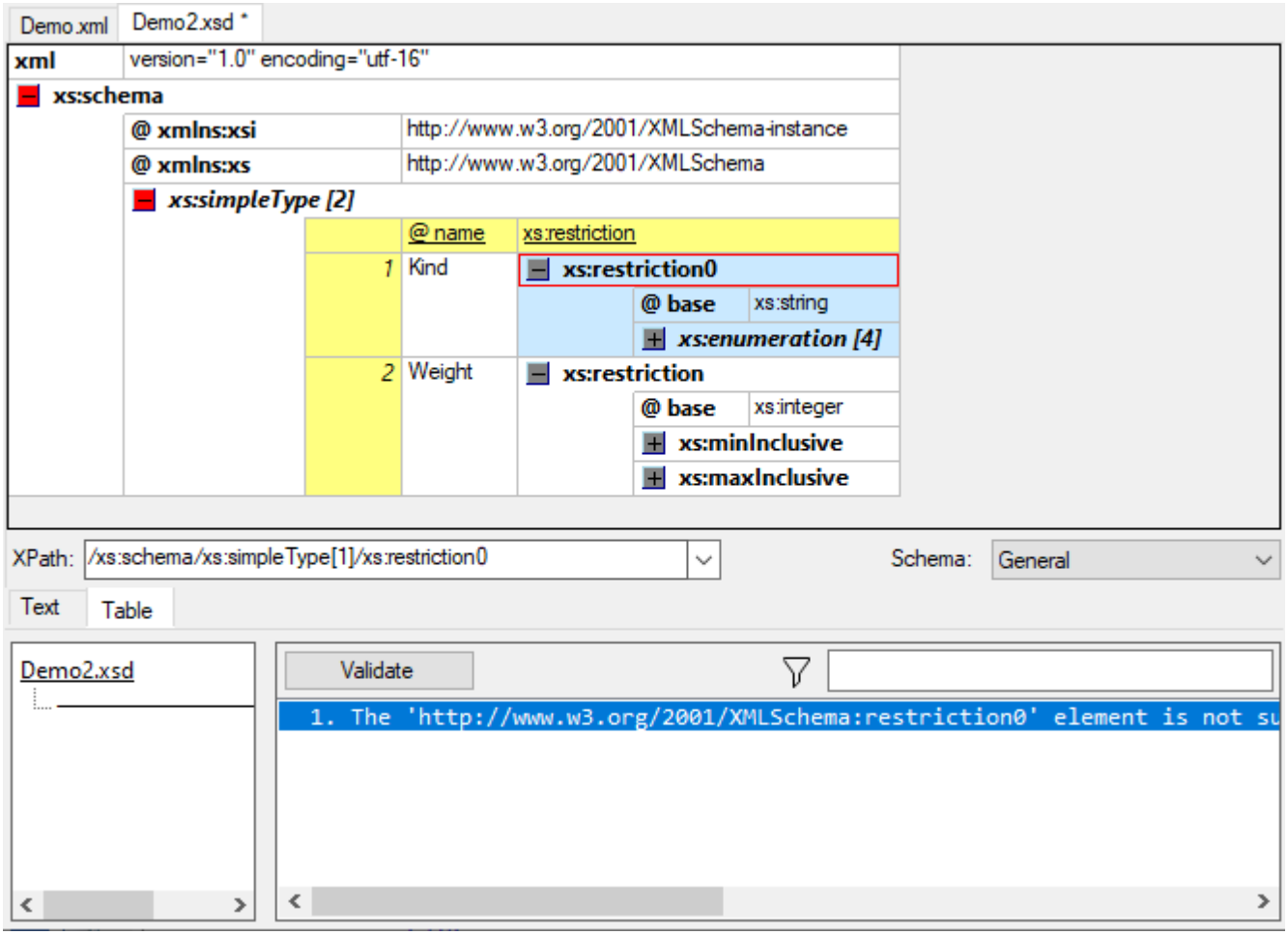


The schema file opens in the editor...



...and we still see the errors from the original file. The top root element 'Demo.xml' is now not underlined because we are editing different file. The reason we still see the errors for original xml file is, that this schema file contains no errors. In such case instead of "no errors" message the original errors remain displayed for better orientation.

Whenever we make some error in the schema, the list will be updated to the current file.



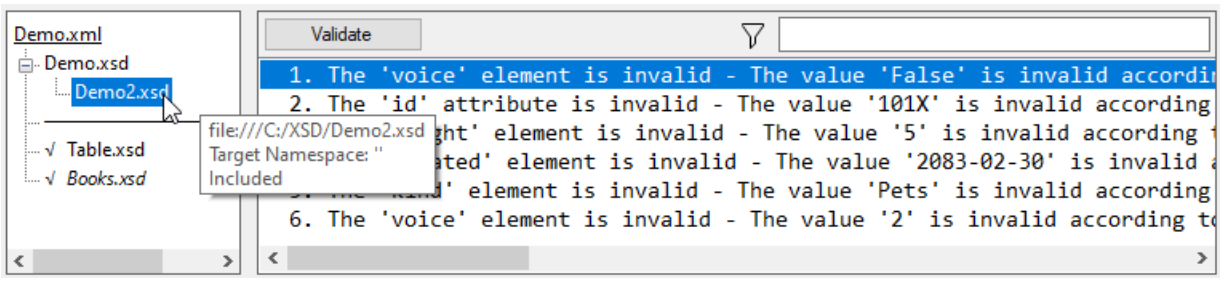
In the picture above we entered wrong name of schema's reserved word and the error is announced in the list. Also, the top root element in the tree view has changed accordingly to the error – now the source is in the active schema file 'Demo2.xsd'.

There are three types of relations between xml and xsd file:

1. Xml file contains reference (schema location) for an xsd schema; this reference is usually relative or absolute file path or a web address
2. Xml file contains schema definition as its own part (inline schema)
3. Schema is loaded additionally for a validation

Types 1 and 2 are automatically used by validation. Type 3 requires adding schema to the tree list to be able to use it.

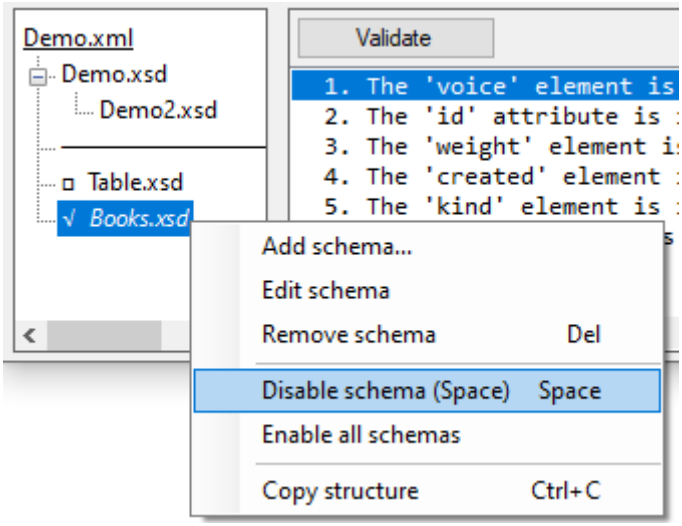
The horizontal line in the tree view is separating schemas of type 1, 2 and loaded schemas of type 3.



In the picture we have some additionally loaded schemas ('Table.xsd' and 'Books.xsd'). In a tooltip for each schema, we can see information about the file or a web location (web schemas are prefixed by a small point before a name '●') and the name of a target namespace.

Target namespace is important identifier for the relation between xml file and schema. If schema has target namespace which is not used in the validated xml file, the name is displayed by *cursive* in the tree view. The schema 'Books.xsd' has a target namespace which is not declared in the 'Demo.xml' file. The list of declared namespaces in the file can be checked in [`XML Information`].

Schemas loaded additionally (from context menu by command 'Add schema...') are common for all xml files. That's why these schemas can be individually enabled or disabled for each file independently. The change can be done by clicking on the symbol '√' or '□' which is displayed before the name. Other possibility is using the context menu.



Disabled schemas are ignored during the validation.

Note: When we do some changes in the xsd schema, these changes must be saved to be able to affect validation of xml file. Changes on the disk are detected by program and the updated version of the schema is used automatically. On the other hand, changes in a web schema are not detected and if necessary, we must request updating schema from the context menu by command 'Update schema' – this command is available only for web related schemas.

Note: Schema or multiple schemas can be added to the tree also by drag & drop.

At the end of this section, we describe all commands from the tree view context menu:

- *Add schema…*

Invokes open dialog and we can select schema(s) to be loaded and used for validation. In principle it's possible to enter a valid web URL address – in such case the web file is downloaded to temporary location and used in a copy.

- *Update schema*

A request for updating loaded schema which is on the web and changed meanwhile.

- *Edit schema*

Open the selected schema in the xml editor.

- *Edit copy of schema*

This option is related for a web related schema if we want to see what is inside. It takes the schema from the web and opens it in the editor as a new xml file. If we do some changes, it's necessary to upload changes back to the web or change the location in our xml file to point to this local schema copy.

- *Remove schema*

Remove schema from the list. It can be done only for schemas which were additionally added. Delete key can be used while standing on the schema we want to remove.

- *Enable/Disable schema*

Enables or disables schema in the list to be used or not used for the validation. The option is specific for each file. Space key can be used while standing on the schema we want to enable or disable.

- *Enable/Disable all schemas*

Enables or disables all additionally loaded schemas. The option is based on the status of the first schema – if it's enabled the option 'Disable all schemas' appears and vice versa.

- *Copy structure*

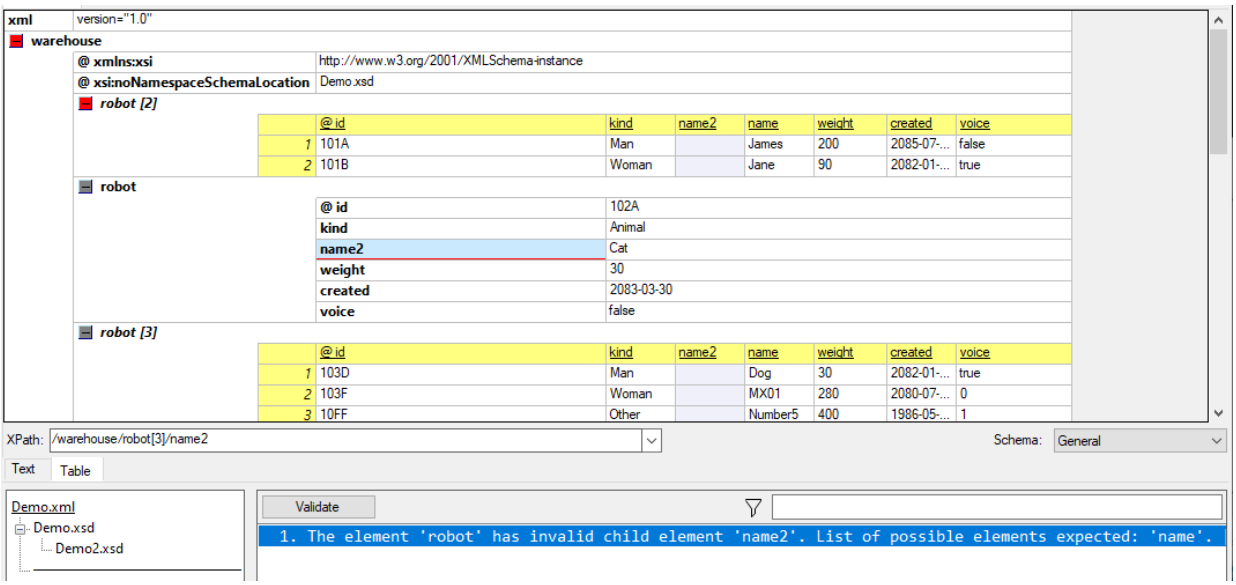
Copy tree structure with locations and target namespaces as text to the clipboard. A shortcut Ctrl+C can be used while the tree view is focused.

23.4.2 Validation error list

In the bottom right side of the application window there is a list with validation errors. The list contains information about the issues, sometimes supplied by additional info. At the end of the message is usually information about the source (path and name), line and column position where the issue was found. If the message is longer than list width it appears in the tool tip.

The list is interactive - if you select some line and the appropriate source file is opened in the editor, then the cell (where the error occurs) is selected (in a text mode the line and column position is selected).
If the position of the validation error is somehow hidden – for instance if the error is in the name of

a table's column – and we are not able to directly edit the value of the element, then a feature "Expose table row" is automatically used. It makes the hidden element available for editing (see also section [5.2 `Expose table row(s)`]).



In the picture is demonstrated how the table row (with wrong column element 'name2' which should be 'name') is virtually exposed as a standalone element which has all children elements accessible for changing.

If source file hasn't been opened yet we have a possibility to open the file and go to the error position by double clicking the line in the list or by using a context menu or by pressing the Enter key while standing on the selected line.

Similar functionality is working also in the opposite way in a table view – when we select some cell with an issue in the editor the appropriate line in the list is selected.

Be aware that if the error is related to some web related file, only the copy of this file could be opened in the editor. In such case the name of the file starts with "Copy of " and it's considered as a new file.

Note: If there are more than 100 errors during the validation the list is truncated and only the first 100 errors are displayed.

Results can be filtered by criteria entered in a filter text box. The filter must be set as active by a filter button which turns filter on and off. Results can be filtered by simple words or by exact phrase, which is encapsulated by a quotation characters. The last option is using a word prefixed by a sign minus '-', which will exclude the word. Phrases are case sensitive, others are case insensitive.

If we want use quotation character in the filter it must be written as double quotation character (""). If we want use minus sign it must be used in a phrase (e.g. "-" or "-23").

Examples:

- *Hello world –happy*
  Show all lines which contains words 'Hello' and 'world' (anywhere in any order) and doesn't contain word 'happy' (all case insensitive).

- *"Hello world" ""2 3"" "-happy"*
  Show all lines which contains exact phrase 'Hello world' (case sensitive) and "2 and 3" anywhere in the text (""2 3"" is not a phrase in this case because double quotation is only escaping of the quotation character) and doesn't contain word '-happy' (case sensitive phrase beginning with the minus sign).

Button "Validate" invokes the validation of currently active file in the editor. By default the validation runs automatically for the first file only.
Validation is also automatically triggered when we modify the file or if a new schema is additionally loaded into tree view.

Commands available from the context menu:

- *Edit schema*

Opens the schema related to the currently selected record in the error list.

- *Edit copy of schema*

This option appears for a schema which is located on the World Wide Web. It creates a copy of the schema.

- *Copy errors*

Copy all displayed (filtered) records from the list to the clipboard. A shortcut Ctrl+C can be used while the list is focused.

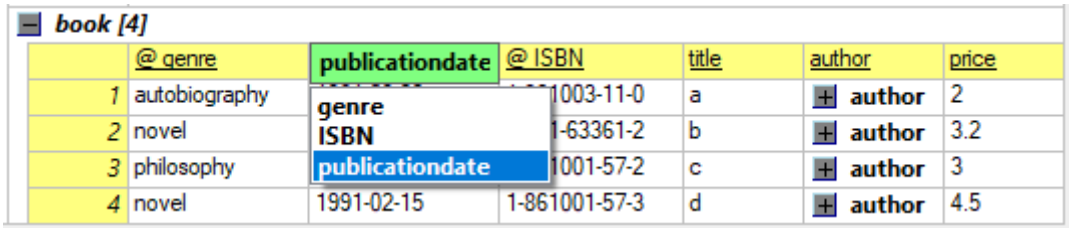
- *Show warnings*

By this option we can see also warnings from the validation. This can be used if we want to verify that the schema fits properly. Warnings might reveal if some elements are not defined by the schema (in principal in such case there is no suggestion for the element).

23.4.3 Auto-complete

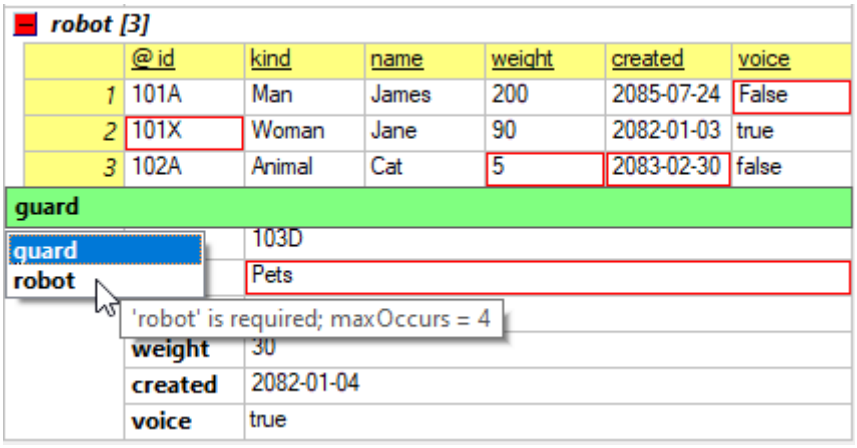
While in validation mode the applied schemas determine a structure of the xml file, the elements' names and also their values (or at least their types). When editing a cell in a table view, the editor will give suggestions what name(s) or value(s) are expected.

For element names and attribute names a list with possible options should appear for edited cell. The list can be closed by Escape key and revoked by F2 or Ctrl + Space (or by a cursor down if the edited value has only one line).

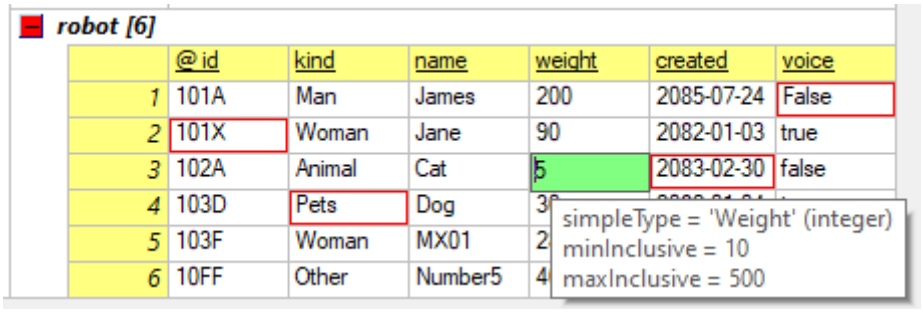


Attributes can be in any order so we see in the picture a list with all possible attribute names while editing the attribute name.

In the list we can use additional tooltip with the information if the attribute/element is optional or required and what is the allowed number of repeating. Also the word 'choice' can appear and it might declare a required element from several choices. In the picture below we can see an example of the information that element 'robot' is required but can be defined at most 4 times.



For values the situation is more complex and xsd schema specifies values' types in different ways. If the value is defined as an enumeration we will see a list of possible values while editing the value cell. But more common is the situation when the value is specified only by its type with some additional rules. In such case only a tool tip appears with information about expected value (see a picture).



The tool tip in the picture gives us the information that edited value is of type 'Weight' (the name of type defined somewhere in the schema) and this type is derived from basic type 'integer'. Moreover there is some restriction for minimal and maximal value (interval 10-500).

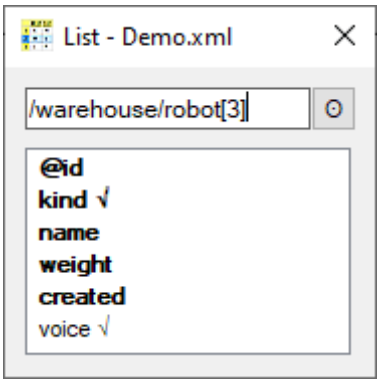
Exception: If some element is not valid, then all elements on the same level which are in the structure behind or inside this element will be without any suggestions because the "track is lost".

23.4.4 Adding missing attributes and elements

If some area is selected in validation mode with the schema(s) which defines the structure of the xml file then a new command group "Schema" will appear in the context menu (see picture below). This group has three commands.

- *Show list*

It opens the dialog with the list of all attributes and elements according the schema definition. In the list we can see the order of defined attributes and elements and in the tooltip its description. We can recognize if the attribute/element is already added (it's marked by a sign "√"), if it's required (bold font) or optional (normal font) and if it's a choice (prefixed by a sign "•").



If we click (or press 'enter' key) on the existing attribute/element (marked "√"), this item is selected in the table. If we double click (or press 'space' key) on element or attribute which hasn't been yet added, it's added as a child and selected.

On top of the dialog there is a text box where the XPath to the element, which we are interested in, can be written (by 'enter' key the list is updated). Next to the text box there is a button by which we can copy current XPath from the active file and evaluate the list.

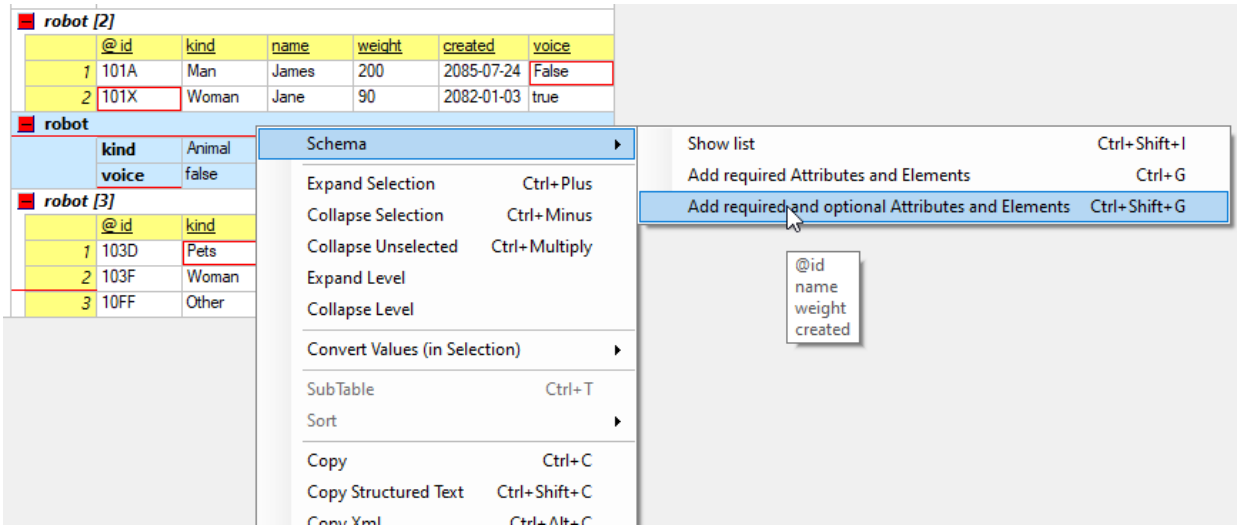
Dialog itself can exist only in validation mode and only in table's view. If we change the mode or switch to text mode or open a sub-table, the dialog is closed.

- *Add required Attributes and Elements*

This will add all attributes and elements which are missing and required according the schema definition with the required minimum number of repeating.

- *Add required and optional Attributes and Elements*

Like the previous one, but additionally it adds also optional attributes and elements.



If the selection contains only one element then we can check in the tooltip which attributes and elements will be added while staying over the menu command. For multiple elements (like whole tables or multiple table rows) there is no preview.

Please note that the command is working with the top selected element and it adds missing elements which are children of this top element only (for instance if we select table rows then elements forming table columns only can be added).

The commands don't fill any values and they leave them empty.

23.5 Generate XSD schema...

For an xml file we can generate xsd schema for later validation. This function will generate xsd schema (or more schemas if necessary) according the current xml file and store it to the selected folder. The xml file must be well-formed.

Please note that for one xml file can be generated more xsd schemas. Number of generated schemas depends on used namespaces and on the complexity of the xml file.

If more than one schema is required they are stored with the indexes (starting from number 2) in round brackets (e.g. "schema.xsd", "schema(2).xsd", "schema(3).xsd").
If the first file exists it's overwritten.
If some other indexed file exists, then a new name sequence with underscores is generated and used to avoid conflicts.

For instance, if exists a file "schema(2).xsd", then files "schema.xsd", "schema_(2).xsd", "schema_(3).xsd" will be created.

All these files should be loaded afterwards for validating the original file (see also command 'Add schema...' in the section 23.4.1 `Tree view`).

23.5.1 Schema location

Validation schema can be additionally loaded or it can be specified directly in the xml file and used automatically. For curiosity we can introduce two trivial examples to demonstrate how the schema can be defined in the xml file.

There are two directives to define schema location – one is for a schema without namespace (noNamespaceSchemaLocation) and the other one for a schema with a namespace (schemaLocation).

1. noNamespaceSchemaLocation

```
<Root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///C:/XSD/trivial.xsd">
        <Element>E1</Element>
</Root>
```

The other possibility is to include schema with a relative path to the position of xml file (for instance xsi:noNamespaceSchemaLocation="trivial.xsd").

And appropriate xsd file (trivial.xsd):

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="Root">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="Element" type="xs:string"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
</xs:schema>
```


2. schemaLocation

```
<x:Root xmlns:x="ximple.cz" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="ximple.cz trivial2.xsd">

        <Element>E1</Element>

</x:Root>
```

And appropriate xsd file (trivial2.xsd):

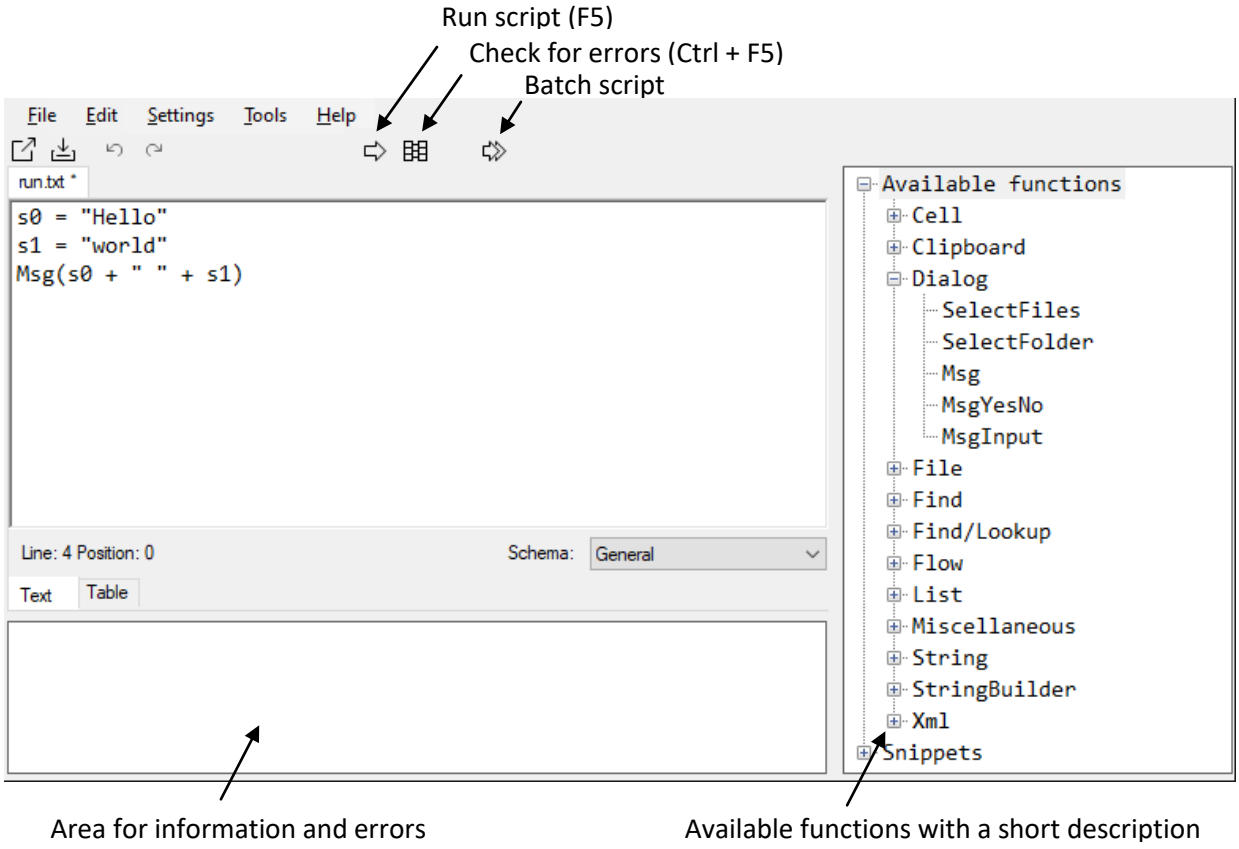
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="ximple.cz">
        <xs:element name="Root">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="Element" type="xs:string"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

Note: The white space in the file path is not allowed by the schema specification (a space character is reserved). The workaround might be a substitution.

23.6 Script mode

From version 1.6 XiMpLe provides functions which correspond to some user actions in the editor. The main idea is to give a user the possibility to prepare often repeated tasks and use them as scripts with a benefit on multiple files.
The script itself is a text file, but running script and checking for errors are available only for registered users (see [28.2 `Registration...`]).

The script mode looks like this:



On the right side there is a tree with available functions and moving mouse over the function will display a short description of the function, its parameters and a returned value. See also [`Appendix A`] for more details.
In the "Snippets" section you can find pieces of codes for some frequently used tasks.

Listed functions can be inserted to the text by double click or by a context menu. This is not possible in the table view or if there is no opened text file.
Entered script code can be checked for errors or run. Another way to run a script is via the File menu [19.10 `Run script...`] or from the command line [23.6.2 `Running script from the command line`].

For scripting exist several predefined variables – it is not possible to define own variable names – from this point of view, it's like the instruction set.

Available **variables** and their types (names are not case sensitive):

- `B0, B1, B2, B3, B4, B5, B6, B7, B8, B9` – Boolean variables (1/true or 0/false)
- `I0, I1, I2, I3, I4, I5, I6, I7, I8, I9` – Integer variables
- `D0, D1, D2, D3, D4, D5, D6, D7, D8, D9` – Real number variables
- `S0, S1, S2, S3, S4, S5, S6, S7, S8, S9` – Text variables
- `C0, C1, C2, C3, C4, C5, C6, C7, C8, C9` – Cell variables (an internal reference to the cell from table view). These variables can be "reset" to no value - null - which is written in the conditions as zero – for instance: "If (C0 != 0)"

- `LI0, LI1, LI2, LI3, LI4, LI5, LI6, LI7, LI8, LI9` – lists of integers (the length of each list is not limited)
- `LS0, LS1, LS2, LS3, LS4, LS5, LS6, LS7, LS8, LS9` - lists of texts (the length of each list is not limited)
- `LC0, LC1, LC2, LC3, LC4, LC5, LC6, LC7, LC8, LC9` - lists of cells (the length of each list is not limited)
- `SX` – string builder – this is a text variable with a special text handling - it should be used for concatenating many longer strings, because its performance is much faster; it can't be assigned directly
- `F, T` – special variables representing the find settings and lookup table settings – these two variables have their own special methods and they are used for find and replace functions; they can't be assigned directly

The variables (except the cell type variables and special variables) can be assigned to each other. If the value doesn't make sense (e.g. B0 = "hello"), an empty string or zero is assigned instead. Otherwise the conversion is done: real number is truncated to integer, non-zero value is converted to true (represented by number 1) and zero value is converted to false (represented by number 0). Text is converted to the real number if it's possible.

List of variables must be firstly filled up by a method Add(). Then items from the list are accessible by index (counted from zero) for reading and writing (e.g. LI2(3) = LI2(0) + 2).

For conditions there can be used these **operators**:
And, Or, Not, >=, >, <=, <, ==, != or <>
Be careful with comparing (two equals) and assignment (one equal). A command „If (I0 = 3)" assigns the value 3 to variable I0 and then jump inside the If because the result of assignment is „true". The correct condition is „If (I0 == 3)".

When script is running the Undo for table's view is disabled.

There are several rules for scripting:

1. In the line can't be more than one leading command (e.g. 'if', 'else' or assignment can't be on the same line; but it's possible to call a function as a parameter of another function).
2. Functions' parameters and conditions must be encapsulated in the round brackets. On the other side if function doesn't use any parameter the brackets are not necessary.
   If the condition contains logical operands (And, Or), each part should be encapsulated as well. For instance:
   ```
   If ((i0 == 1) and (i1 == 2))
     Open("file.txt", 0)
     Close                    // same as Close()
   Endif
   ```
3. Usually it's necessary to assign firstly the function's result to a variable and then use another function or index.
   For instance: it's not possible to write C1 = *C0.Children(0)* – it must be split:
   LC0 = C0.Children and then on the new line: *C1 = LC0(0)*.
   There are some exceptions when there is no index or any parameter used. For instance this is allowed: *C1 = C0.Parent.Parent* or *I0 = C0.Children.Count*
4. Commands and variables are case insensitive
5. Only predefined variables can be used to store values

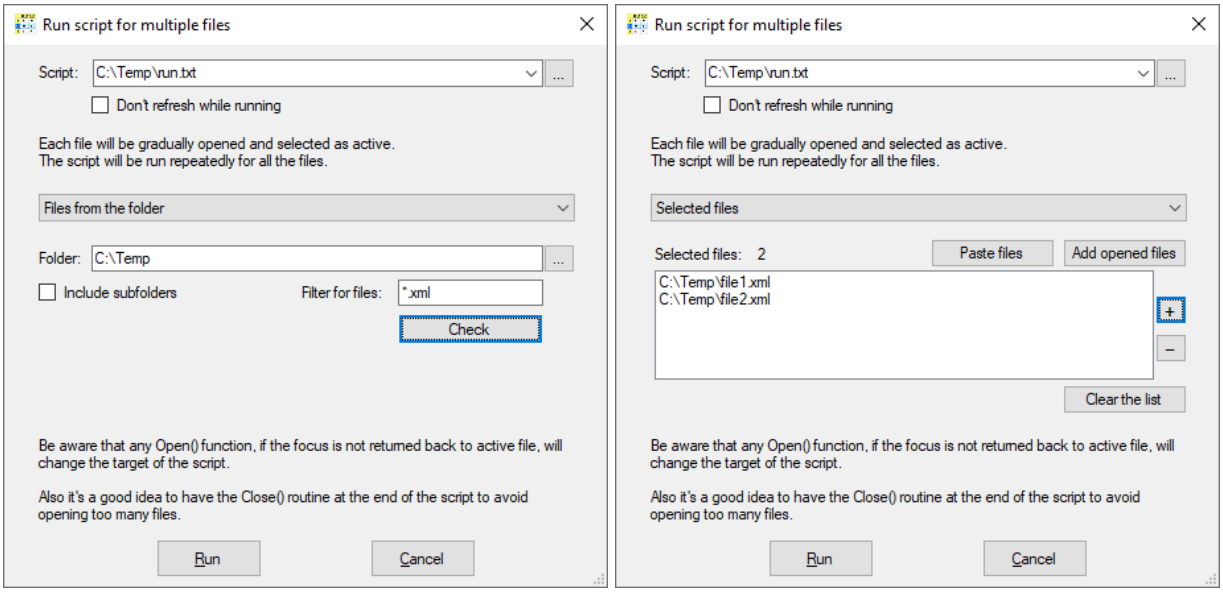
6. Script execution doesn't raise any runtime errors (e.g. if index is outside the array size, the action is ignored, property of not initialized or null object is ignored and so on)
7. Backslash in the string is used for writing special characters:
   \r, \n, \t, \" for carriage return, line feed, tabulator and quotation mark
   The backslash itself must be written as double backslash (e.g. *S0* = "*C:**Temp*").
   If the string is prefixed by the character @ all characters inside the string are taken as they are. This might be useful especially when the file path is entered.
   Example: *S0* = @"*C:**Temp*"
8. A string must be in one line, it can't be broken on several lines. But it's possible to concatenate strings by a plus sign, e.g. S0 = S0 + "another part of text"
9. The "null" is represented by a zero and can be used for assignment to the cell variables (C0 = 0) and for conditions.
10. For comment until end of line can be used double slash // or an apostrophe ' and everything behind these characters is ignored until end of line.
    For bounded comment (which can be also over several lines) the slash-asterisk '/*' and asterisk-slash '*/' can be used. For instance:
    If (I0 /* this is integer variable */ == 2 /* number two */)
11. The conversion between different types (except cell type) is done automatically by assignment if possible (e.g. s0 = i0, i0 = d0, i0 = s0, b0 = i0)
12. Texts and other basic values can be concatenated by a plus sign (e.g. "<V=" + I0 + ">")

The script can be interrupted by pressing the Escape key. Then a dialog appears with a question if you want to stop the script or continue. The dialog also displays values B0, B1... B9 and L0, L1 ... L9 which could help in the decision. For instance there could be seen the state of the long loop or some other information filled by the author of the script.

23.6.1 Batch script

The buttons "Run script" and "Check for errors" are working on the currently selected (and possibly edited) text file. The third button "Batch script" opens new dialog window where we select the script to run and the files which should be processed gradually by the script. There are two main possibilities:

1) Apply the script on the files from the selected folder (and all subfolders if required) which match defined filter (picture on the left below).
2) Apply the script on the selected files from the defined list (picture on the right below).



Normally, we can write or record the script and then encapsulate the script inside the loop, which will run the script for all files we want to process, by using the function GetOpenedFiles() or GetFiles() (see also the snippets) with other necessary modifications.

The "Batch script" functionality tries to do the encapsulating for the user, but some modifications are necessary too. It's important to understand that if we record the script, it works with one or more files and there is always used (at least at the beginning) a function Open() which opens (and select as active) the file for next actions.
The "Batch script" takes the files and opens them in the editor one by one. Then it executes the script and when script finishes, it opens next file and runs the script again.
That means that any Open() function in the user's script will change the active file for the batch. So it's necessary to

a) disable such command by remark (see example)
b) return the active file back (see example)

Example:
We have started to record script and then we have opened file "C:\Temp\Demo.xml", selected the second column in the table /Demo/Private/Variables and delete it. Then we have stopped the recording. The code is generated:

```
Open(@"C:\Temp\Demo.xml", 1)
C0 = XpathQuery("/Demo/Private/Variables[1]")
C0.TableColsSelect(1)
Delete(1)
```

Now if we want to run this script as batch for other files Demo2.xml and Demo3.xml (we assume that these files have the same xml structure), we need to modify the script:

a) We don't need to open any file because it will be opened by Batch functionality.

```
// Open(@"C:\Temp\Demo.xml", 1)
C0 = XpathQuery("/Demo/Private/Variables[1]")
```

```
C0.TableColsSelect(1)
Delete(1)
```

b) We must return the active file back

```
// now the active file is the one opened by Batch functionality
S0 = GetActiveFile()      // remember what was the active file
Open(@"C:\Temp\Demo.xml", 1)   // we need to do something with Demo.xml so make it active
// now do some action with Demo.xml
Open(S0)                  // select back the previously active file
C0 = XpathQuery("/Demo/Private/Variables[1]")
C0.TableColsSelect(1)
Delete(1)
```

Another thing which we should have on mind is the closing of the files to avoid too many opened files in the editor. So always consider the Close() function at the end of the script.

Description of the dialog itself:

- *Script:*

It defines the path to the script which will be run. If it's opened in the editor and modified, it will run the unsaved modified version.

- *Don't refresh while running*

By this option we can suppress editor refreshing and increase the performance of the execution.

- *Files from the folder*

If this option is selected, we need to specify the folder and the filter for the files.

  - *Folder:*
    It defines folder from which the files will be taken.
  - *Include subfolders*
    By this selection we could include also all files from all subfolders.
  - *Filter for files:*
    It defines the filter for the files. In filter we can use wildcards '*' (any group of characters) and '?' (any one character). For instance: filter "m?x*.x*" will match the files like "maximum.xml" or "max.xyz"
  - *Button "Check":*
    By this button we can check the results of the settings. It displays number of found files and paths of the first five and last five files in the list.

- *Selected files*

If this option is selected, we need to specify the list of files.

  - *Button "Paste files"*
    Add to the list files from the clipboard. Full paths are expected, each file on a separate line and the files should exist at the time of pasting.
  - *Button "Add opened files"*
    Add to the list all files which are currently opened.
  - *Button "+"*
    Select file or more files and add it or them to the list.

- *Button* "-"
  Delete all records in the list which are selected.
- *Button* "*Clear the list*"
  Clear the list.

- *Run:*

It starts the execution. If the selected script contains some error, the execution is interrupted reporting the error. Otherwise it will open the files according the settings and execute the selected script repeatedly.

23.6.2 Running script from the command line

Another feature, related to the scripts, is the possibility to run the script from the command line (available only for registered users). Following switches can be used (equal sign is the part of the switch).

- For defining the script for execution exists the switch /script=
  /script="path to the script which will be executed"
- It's possible to pass text or value to the script from command line by switches /S0= /S1= and so on up to /S9=.
  /S0="text or value which we want to transfer to the script in the variable S0"
- To hide the editor's window completely, the switch /hide is used. At the end of the script the editor will be shown unless the switch /quit is used.
  /hide
- The editor can be quit after the execution by the switch /quit. It has the same effect like the command Quit() in the script file. It can be used only together with switch /hide.
  /quit

Example: XiMpLe.exe Data.xml /script="C:\Temp\test.txt" /S0="23.45" /S1="text with spaces"

Such command opens XiMpLe editor then it loads the file Data.xml and runs the script 'test.txt'. When script starts the value for S0 will be set to 23.45 (such value can be assigned to double value like D0=S0) and S1 will contain "text with spaces".
Remark: the quotation marks are not necessary if there are no spaces – so the switch /S0=23.45 should also work.

24 Window Menu

In this menu we can see an overview of all opened files. After start-up this menu is empty. Each opened file appears here with the full path and information if the file is currently opened in a text view or in a table view.

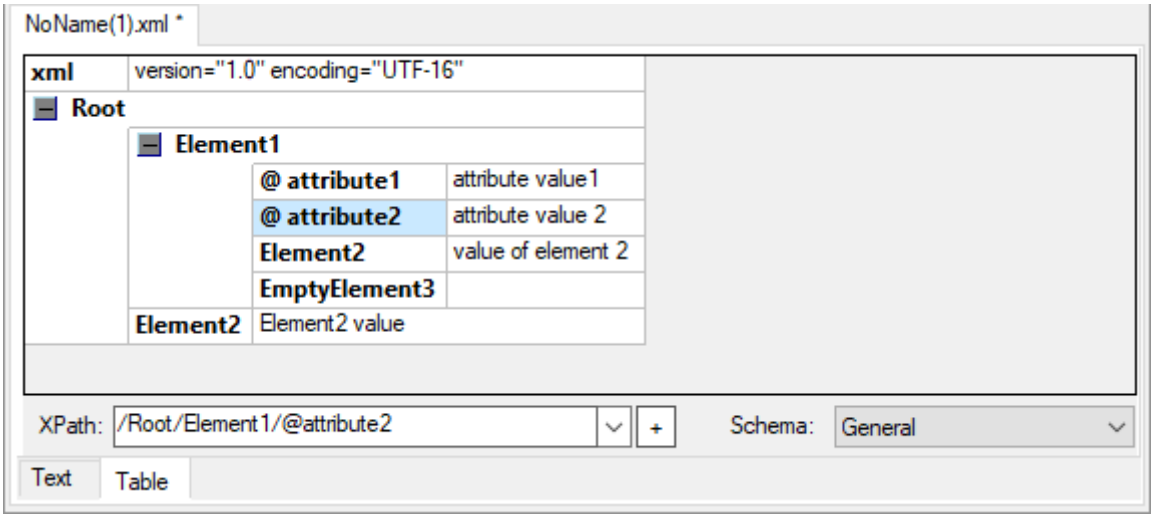
In the compare mode [section 23.1 Compare Mode] there appears a splitter between files on the left and right side. In this mode we can click on the menu items (files) by a right mouse button to move it from one side to another without closing the menu and arrange the files easily.

25 XPath

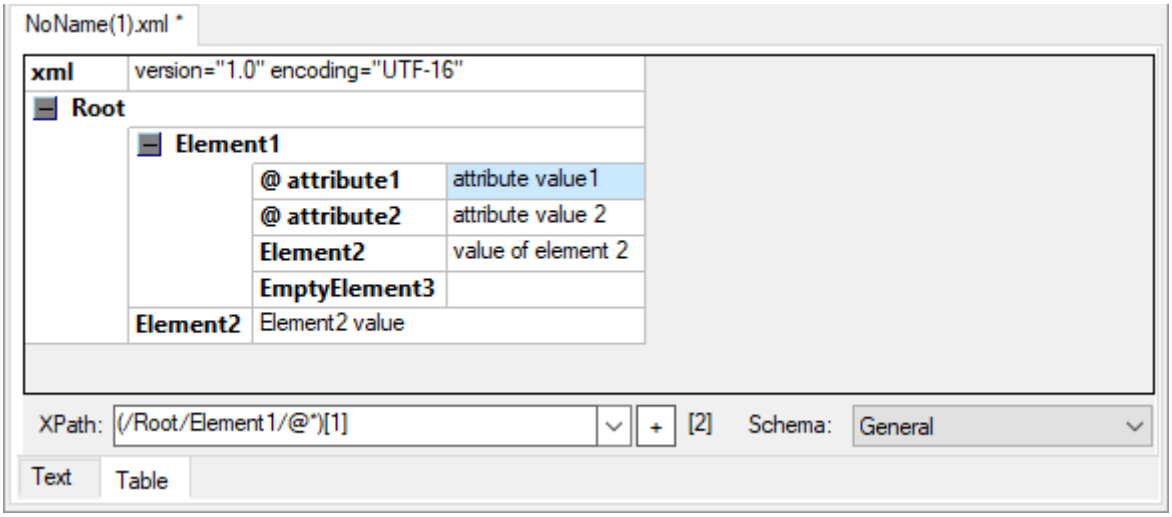
XPath is something like a file path but used in a context of an xml data structure (simplified). It gives the way to some object (attribute, element etc.) inside xml data.

For a selected cell (generally for a top-left cell in a selected area) an XPath is displayed below the editor. Please note that XPath is case-sensitive.

XPath can be used also reversely for a jump to some place. XPath can be modified and when the "Enter" key is pressed the first related cell is selected (if XPath doesn't exist a background color is changed to grey).



It's possible to enter an XPath query as well. If there is more than one result for a query the first match is displayed and on the right side of the XPath appears information about the number of finding records.



In this example (see picture above) we have entered the query /Root/Element1/@* which should found all attributes of /Root/Element1 – in our case there exist two attributes (@attribute1, @attribute2) that's why there is information "[2]" on the right side of the XPath. The XPath string was slightly modified by a program to the form (Root/Element1/@*)[1] which should associate that the first record is matched. If we want to search the second record we should change the index [1] to [2]: (Root/Element1/@*)[2].

Please note that for instance (Root/Element1/*)[2] and Root/Element1/*[2] do not need to be the same.

Right click on the XPath text box will invoke a context menu with the commands:

- *Copy XPath*

It copies the content of the XPath text box

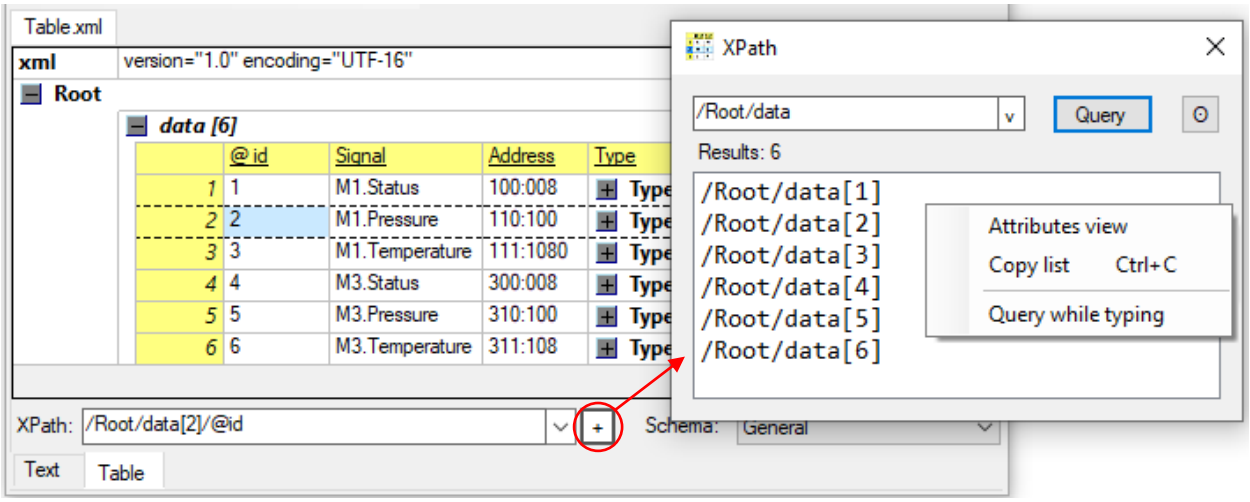
- *Copy XPath without "xmlns:" prefix*

It copies XPath text and remove prefixes "xmlns:" if they are used in the XPath (these prefixes are used when the default namespace is redefined by the directive xmlns="something").

25.1 XPath dialog

Additionally has been added an XPath dialog which can provide results of an XPath query in a more comfortable way. Mainly when there are more results for the query.

The dialog is invoked by a button "+", which is next to XPath edit box (see next picture).



The XPath dialog contains an edit box where the XPath query can be entered, button for getting results (the query is processed also by pressing Enter while in the edit box).

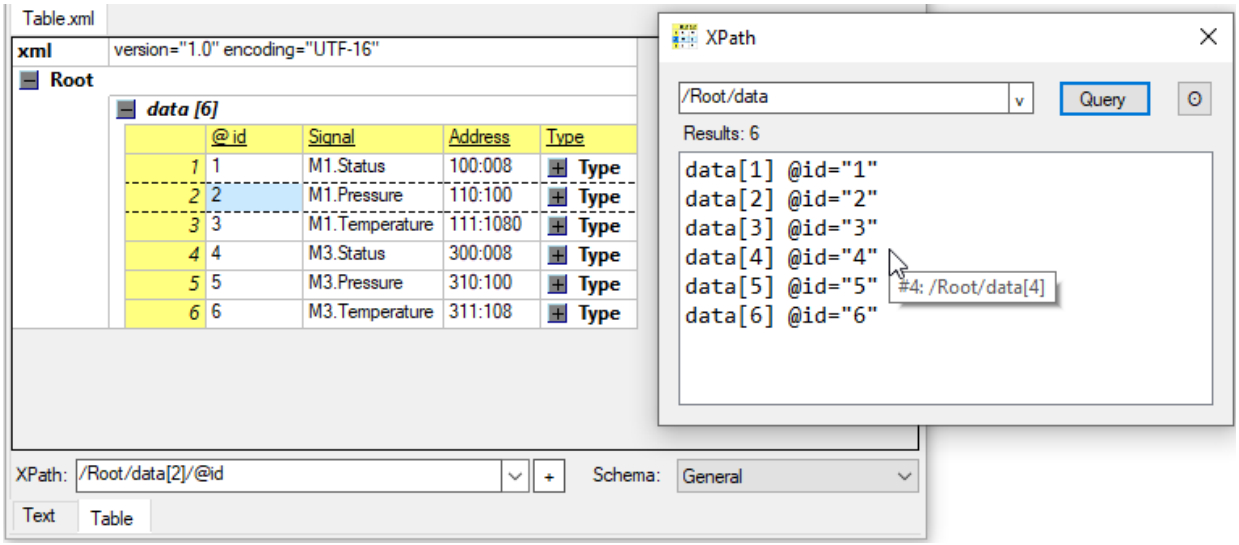
By option 'Query while typing' can be turn on the mode when the XPath query is evaluated always when the text is changed (no confirmation is required). In this mode are not stored entered queries until the button 'Query' or Enter key is used.

The button in the right top corner is for copying XPath from the active file (in the picture above will be put a string "/Root/data[2]/@id" into the edit box). When some part of the text is selected this function will overwrite only selected text instead of overwriting complete string.

The largest area is the list with the query's result. Results can be viewed in two ways and switching is done by a context menu.

- In the "XPath view" full XPath is displayed for each result.
- In the "Attributes view" results are displayed as the name of the last element with associated attributes and corresponding values.

When selecting some result from the list the appropriate cell in the file is selected.



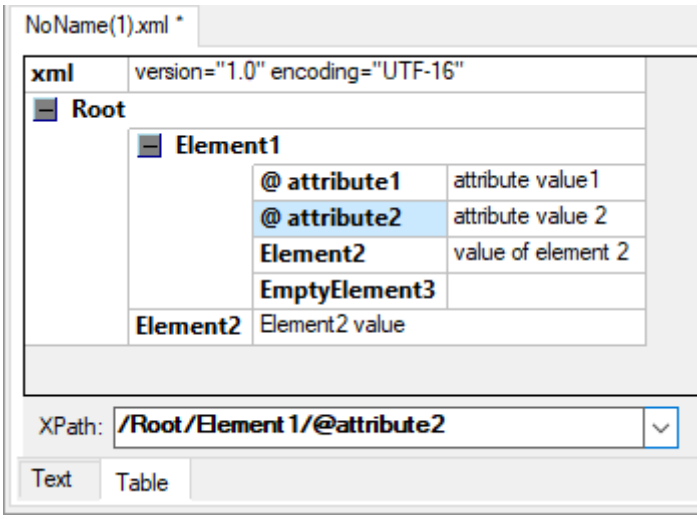
In this picture results are displayed in 'Attribute view'.

Last queries are stored and remembered and can be invoked by the small button "v" which is next to the edit box.

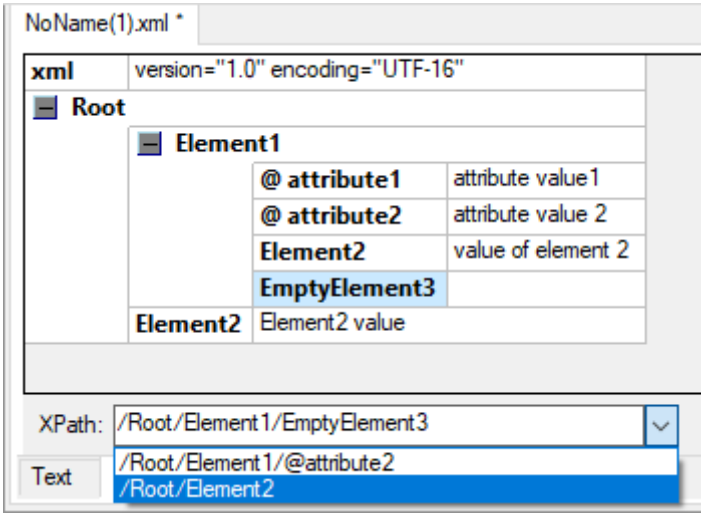
26 XPath Bookmarks

XPath which appears in the edit box can be locally stored (each opened file has its own bookmarks) and used later as a shortcut to some cell.

For storing XPath as a bookmark we can use a key combination Ctrl + B or a command from the menu Edit -> Bookmarks -> Add / Delete Bookmark. XPath string is displayed as bold if it's bookmarked. If the bookmark already exists then Ctrl + B will delete it from the list.



We can select a bookmarked XPath from the list in the combo box next to XPath edit field or in the editor we can use keys F6 (next bookmark) and F7 (previous bookmark) directly for jumping to the bookmarked cell (in case the XPath string is valid for current xml file).



In this picture the current cell EmptyElement3 which is not bookmarked (XPath text is not bold) and in the combo box we can see two records – bookmarks: /Root/Element1/@attribute2, /Root/Element2. We can jump between these two cells in editor by pressing F6 and F7.

27 Shortcuts

A list of keyboard shortcuts which can be used in the XML editor is divided to several categories according to the place of activity.

27.1 Standard mode

	Global	Inside a table	Outside a table	Editing
Ctrl + Home	Go to the first line			
Ctrl + End	Go to the last line			
Home		Go to the up-left corner		
End		Go to the right-down corner		
Page Up	Previous page			
Page Down	Next page			
Left/Right/Up/Down	Move to the neighbourhood cell			
Left/Right/Up/Down + Shift	Move to the neighbourhood cell with a selection			
Left/Right/Up/Down + Ctrl		Go to the boundary element in the pressed direction (can be combined with Shift)	Go to a next non-simple (=can be collapsed) element or a table	
Left/Right + Alt (+ Ctrl)[7]		Go to the first "different"[5] value type in the pressed direction (can be combined with Shift)		
Up/Down + Alt (+ Ctrl)[7]		Go to the first "different"[6] value type in the pressed direction (can be combined with Shift)		
Tab		Go one level deeper (if possible)		
Enter	Expand / Collapse current cell			Leave editing and accept changes[1]
Ctrl + Enter				Insert a new line[1]
Alt + Enter				Leave editing and accept changes always
Ctrl + A		Select all table cells		

Ctrl + Plus	Expand all selected cells			
Ctrl + Minus	Collapse all selected cells			
Ctrl + *	Collapse all non-selected cells			
Ins		Insert a row[2]	Insert an element to a cell[4]	
Ctrl + Ins		Append a row[2]	Append an element to a cell[4]	
Shift + Ins			Insert an attribute to a cell[4]	
Ctrl + Shift + Ins			Append an attribute to a cell[4]	
Del		Delete all selected rows / Delete values in selected cells	Delete all selected cells	
Ctrl + Del	Remove data	Remove selected column(s)[3]		
(Ctrl +)[7] Alt + Up		Move selected row(s) one row up[2]		
(Ctrl +)[7] Alt + Down		Move selected row(s) one row down[2]		
Alt + Page Up		Move selected row(s) 10 rows up[2]		
Alt + Page Down		Move selected row(s) 10 rows down[2]		
Alt + Home		Move selected row(s) to the top[2] / Move selected column(s) to the left edge[3]		
Alt + End		Move selected row(s) to the bottom[2] / Move selected column(s) to the right edge[3]		
(Ctrl +)[7] Alt + Left		Move selected column(s) one column left[3]		
(Ctrl +)[7] Alt + Right		Move selected column(s) one column right[3]		
Esc				Leave editing and ignore changes
F2	Start editing			
F3	Find next			
F4	Replace			

F8	Start/stop script recording			
Ctrl + C	Copy selected area to clipboard			
Ctrl + Shift + C	Copy selected area as structured text			
Ctrl + Alt + C	Copy selected area as xml			
Ctrl + V	Paste clipboard data			
Ctrl + X	Cut selected area to clipboard			
Ctrl + T		Sub-table for a selected table column		
Ctrl + Z	Undo (not working in a sub-table)			
Ctrl + Y	Redo (not working in a sub-table)			
Ctrl + M	Adapt column widths			
Ctrl + F Ctrl + H	Find and Replace			
Ctrl + D	Find and Modify			
Ctrl + Shift + D		Count duplicates		
Ctrl + I	File information			
Ctrl + R	Rebuild tables and refresh grid			
Ctrl + N	Create new empty XML file (Unicode encoding)			
Ctrl + W	Create new empty text file			
Ctrl + J	Create new empty JSON file			
Ctrl + E		Expose table row(s)	Expose single element as table	
Ctrl + Shift + E	For the current file open containing folder			
Ctrl + K	Comment selected area			
Ctrl + Shift + K	Uncomment comments in the selected area			
Ctrl + Shift + F		Freeze table heading		

Ctrl + Shift + U	Unfreeze table heading			
Ctrl + P	Copy content of the XPath editbox			
Ctrl + Shift + P	Copy content of the XPath editbox without prefixes "xmlns:" if they are presented			
Ctrl + B	Add or Delete a bookmark			
F6	Go to the next bookmark			
F7	Go to the previous bookmark			
Ctrl + mouse wheel	Zoom in/out			
Alt + R	Run a script			
Ctrl + 1	Standard mode			
Ctrl + 2	Compare mode			
Ctrl + 3	Validation mode			
Ctrl + 4	Script mode			
Shift + F6, ..., F10	Run a script assigned in the script settings			

[1]Functionality is switched if we are editing multiline text (see also 8. `Editing values`)
[2]A whole row (or more rows) must be selected
[3]A whole column (or more columns) must be selected
[4]A single element cell should be selected
[5]For this functionality a program recognizes four value types: no-value; empty string; non-empty string; element structure; and tries to find the first different one in a left/right direction – if all values are of the same type nothing happens
[6]For this functionality a program recognizes five value types: no-value; empty string; non-empty string; element structure; and tries to find the first different one in a up/down direction – if all values are of the same value type then it goes to the first different value (if all values are the same in the selected direction nothing happens). When editing JSON file there is one more type – it distinguishes between json numbers and texts.
The last feature can be used in the column where are many same repeated values and we want to find the first different one. But be aware that "different value type" has higher priority than "different value".
Example: let's have a column with ten values – first five values are "0" then four values are set to "1" and the last cell is empty. If we stay on the first cell and go 'alt + down' the cursor will be set on the last cell which is empty (different value type has higher priority). If we stay on the eighth cell and go 'alt + up' then the cursor will be set to the fifth cell because from the eighth cell in the up direction, there are all values of the same type and the first different value is found on the fifth position.
[7]Control key is necessary in comparing mode because shortcuts *Alt+Up/Down/Left/Right* are occupied by more frequently used 'next/previous difference' and 'merge difference' functions. In the standard mode *Ctrl* is not required for the affected shortcuts.

27.2 Comparison mode

Here are shortcuts which expand a standard mode and can be used in the comparison mode only.

	Global	Text comparing only	Xml comparing only
Shift + Tab	Switch focus between left and right side		
F5	Do comparing		
Alt + Down	Go to next difference		
Alt + Up	Go to previous difference		
Alt + Left	Merge current difference(s) from right to left		
Alt + Right	Merge current difference(s) from left to right		
Shift + Alt + Down		Go to next inline difference	
Shift + Alt + Up		Go to previous inline difference	
Shift + Alt + Left		Merge current inline difference from right to left	
Shift + Alt + Right		Merge current inline difference from left to right	
Alt + F2			Show detailed differences in text mode for two values

27.3 Validation mode

Shortcuts which expand a standard mode and they can be used in the validation mode only.

	Global	Cell editing
F5	Do validation	
Ctrl + Shift + I	Show list of all attributes and elements according schema(s)	
Ctrl + G	Add required attributes and elements according schema(s)	
Ctrl + Shift + G	Add optional and required attributes and elements according schema(s)	
F2 / Ctrl + Space		Invoke list of allowed elements/attributes according schema(s) if it was closed while editing

27.4 Script mode

Shortcuts which expand a standard mode and they can be used in the script mode only.

	Global	
F5	Run a script from the currently open text file	
Ctrl + F5	Check a script from the currently open text file for errors	

28 Help

28.1 About...

Display information about the program and the author.

- *Button "Check for a new version"*

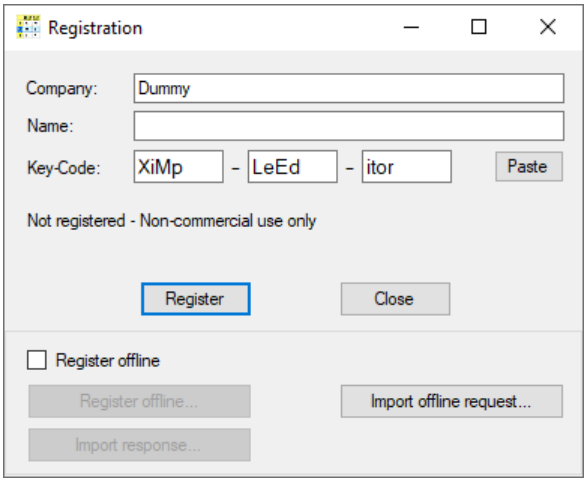
On the user request the program asks the web site if there exists some newer version. During this action program sends out the information about current program version. If newer version exists, the user is asked if she/he wants to visit the page for download.

Anyway, you can visit a web page http://www.ximple.cz or you can contact the author via email (English or Czech): ximple@email.cz

Don't hesitate to contact the author. Especially appreciated are bug reports and words of appreciation. ☺

28.2 Registration...

In the case you want to use a program for a commercial use you need a license. A license key can be entered in the registration dialog.



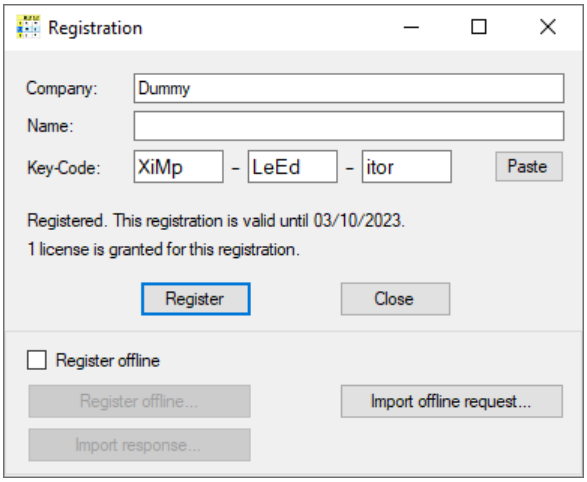
From version 1.6 the registration is necessary for scripting features and the registration itself has been extended to online registration.

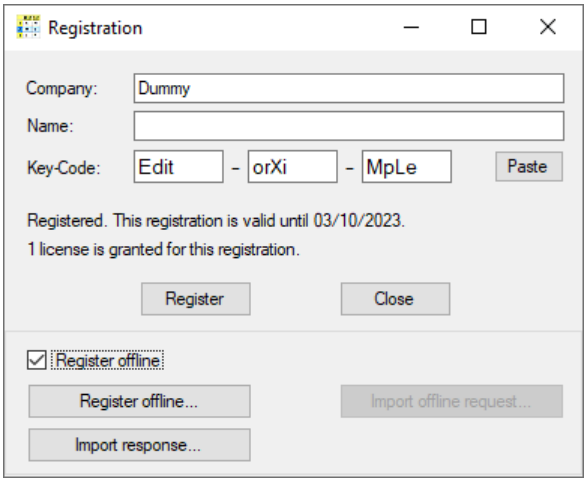
It's done automatically when the registration is confirmed. In case, that the computer is not connected to the internet, only offline registration is done. A user can recognize it, for instance, on

the command in the menu File -> Run script... If it's disabled, then the online registration is not completed.

Note: If you have registered in some previous version, you need to go to registration and press "Register" button once more time to register it online and get the access to scripting features.

A valid license key contains information about the company name or personal name, number of licenses and the expiration time.





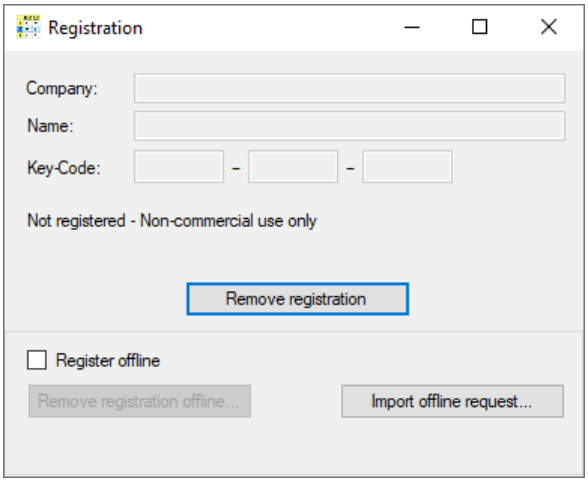
In case, the computer is not connected to internet, it's possible to transfer registration request and import back the response. The procedure is as follows:

a) Select the checkbox "Register offline" to enable buttons for offline registration
b) By "Register offline..." button a file will be created and this fill should be transferred
c) On the place, where is the connection, the file from step b) is imported by "Import offline request...". New file will be created as a response to this request (you will not see the result of the request). This file should be transferred back to the original computer
d) By "Import response..." the file from step c) is imported and if everything is working, the license will be completed.

After successful online registration you can see how many registrations are in use from the total amount of registrations (e.g. 4/10 means that 4 licenses are used and 6 are still available).

28.3 Remove registration...

If you want to remove a license you can do it by this command. After a confirmation all registration information will be removed and the program will be unregistered.



If you want to remove online registration from the computer without internet connection, please use the button "Remove registration offline...". A new file will be created and the registration will be removed from the computer. That means it will not be possible to generate another request, so don't delete the created file until you finish the complete procedure. It's important to transfer that file on the place with the internet connection and then imported it by button "Import offline request...". This will release the registration from the server so another registration can be done elsewhere.

29 Appendix A – Script Functions

Conventions for this chapter:

- Names of the functions are written as bold.
- For each parameter there is written the type or more possible types of the parameter after the colon
- The optional parameters are written in the square brackets.
- For optional parameters there is mentioned the default value after the equal sign, which is used if the parameter is missing.

29.1 Flow commands

We start with the commands for driving the flow of the script. Most of commands in this section don't return any value. Each command must stay alone on the line.
For instance: *If (I0 < 1) Break* is not allowed.

Whenever the parameter "condition" appears in this section, it means the evaluate-able expression which can use also logical operators *And*, *Or*, *Not*.

`Break`
Break the Loop - EndLoop cycle. It jumps outside the cycle.

`Continue`
Jump to the end of the Loop - EndLoop cycle but stay inside the cycle.

`Else`
Do code if the related 'If'/'ElseIf' condition is not true.

`ElseIf``(condition: Boolean)`
Test the "condition" when all previous related 'If'/'ElseIf' conditions are false and if the "condition" is true, do code next to this instruction.

`EndIf`
End of the 'If'/'ElseIf' scope. It's always required to close the previously used 'if'.

`EndLoop``([condition: Boolean])`
End of the 'Loop' scope. Optional condition can be used. In such case the loop is left if the condition is false, otherwise it jumps to the beginning 'Loop' instruction,

`Exit`
Stop execution of the current script and return to the parent script or stop it completely if it's the main script.

`If``(condition: Boolean)`
Test the condition and if the result is true do code between 'If' and appropriate 'Else'/'ElseIf'/'EndIf'. If the condition is false, it jumps to next appropriate 'Else'/'ElseIf'/'EndIf'.

Loop([condition: Boolean])
Begin of the 'Loop' scope. Optional condition can be used. In such case the loop is entered if the condition is true. The test is done every loop.

Example: Next three loops are doing the same:

```
I0 = 0
I1 = 5
Loop (I0 < I1)
  I0 = I0 + 1
  S0 = S0 + I0
EndLoop
S0 = S0 + "\r\n"
```

```
I0 = 0
I1 = 5
Loop
  I0 = I0 + 1
  S0 = S0 + I0
EndLoop (I0 < I1)
S0 = S0 + "\r\n"
```

```
I0 = 0
I1 = 5
Loop
  I0 = I0 + 1
  S0 = S0 + I0
  If (I0 >= I1)
    Break
  EndIf
EndLoop
```

[Long =] **RunProgram**(programName: String, [programParams: String = ""], [wait: Long/Boolean = 1], [workingDirectory: String])

Run external program. Parameter "programName" can be a relative path – in such case the path "DirScripts" is used as the parent folder. It's possible to also set parameters and working directory. The execution can wait until the external program is closed. In such case the program exit code is returned.
If "wait" = 0, then function returns 1 if the execution is successful and 0 if it fails or was interrupted.

[Long =] **RunScript**(scriptName: String)

Run a script in another file. Parameter "scriptName" can be a relative path – in such case the path "DirScripts" is used as the parent folder. It runs in the same variable context.

Return 1 if file exists and run. Return 0 if the file is not found or there is an error in the syntax.

Stop
Stop running script completely. If it's called in the main script, this command has the same effect as the command "Exit".

Quit
Quit the application (editor) without asking. If there are some unsaved changes, they are lost.

29.2 Cell type functions

These functions have the cell variable as the base parameter and give the support to walk through the xml structure.

[Cell list =] cell**.Children**

Get the list of all cell's children. If there are no children the list is empty.

Example: *LC0 = C0.Children*

[Cell =] cell.**FirstChild**

Get the first child of the cell. If the cell has no children, it returns 0.

Example: *C1 = C0.FirstChild*

[Cell =] cell.**GetTable**

Get the nearest cell which forms table's row (it could be the original cell itself). To get further information about a table, we are using the table's row as a reference cell in the script.
If the cell is not inside any table it returns 0 (null).

Example: *C1 = C0.GetTable*

```
If (C1 != 0)
    Msg("C1 is table: " + C1.IsTable)
Else
    Msg("C1 is null, C0 is not in any table")
EndIf
```

[Long =] cell.**IsAttribute**

Returns 1 if the cell is an attribute or the attribute's value, otherwise returns 0.

[Long =] cell.**IsComment**

Returns 1 if the cell is a comment or the comment's value, otherwise returns 0.

[Long =] cell.**IsElement**

Returns 1 if the cell is an element or the element's value, otherwise returns 0.

[Long =] cell.**IsTable**

Returns 1 if the cell is a table's row, otherwise returns 0.

[Long =] cell.**IsValue**

Returns 1 if the cell is a text value of an element, attribute or comment. Otherwise it returns 0.

[Cell =] cell.**NextCell**

Get the next cell of the xml structure. That means:

a) return the first child if the cell has some child
b) return next sibling of the cell if there are no children but there is some next sibling
c) return parent's next sibling if the cell is the last sibling and so on

Only the last cell of the whole structure returns 0 (null).

`[Cell =] cell.NextSibling`

Get the next sibling of the cell or 0 (null) if the cell is the last child of the parent.

`[Cell =] cell.Parent`

Get the parent of the cell or 0 (null) if the cell is the top cell (if the cell is an element then it's a root element).

`[Cell =] cell.Select([append: Long/Boolean = 0])`

Select the cell in the currently active table's view.
`append` = 1: The selected cell is appended to already selected block.

Return the selected cell or 0 (null), if the cell is not from the currently active xml file or the active page is not in a table's view.

`[Cell =] cell.SetValue(newValue: String/Long/Boolean/Double)`

Set the new value to the cell. If the cell is an element or an attribute, rename it. The cell must be from the currently active table's view.

Return the cell or 0 (null), if the cell is not from the currently active xml file or the active page is not in a table's view or the value can't be changed (e.g. not correct value or a target cell is not editable).

`[Cell =] cell.TableCell(rowIndex: Long, colNameOrIndex: String/Long)`

Get the table's cell from the specified position. The row index and column index is counted from zero (first row and first column has an index 0). The column can be specified also by the case-sensitive name of the column. The attribute name must be prefixed by '@'.

Return the cell or 0 (null), if the cell is not a table's row or some parameter is outside the table or the column's name doesn't exist.

Example:
```
C1 = C0.GetTable
    If (C1 != 0)
        C2 = C1.TableCell(0, "@Id")
        If (C2 != 0)
            Msg("We have table's cell(0, \"Id\")")  // C2.Value == "Id", C2.IsAttribute == 1
        Else
            Msg("C2 is not valid")
        Endif
    Else
        Msg("C1 is not valid")
    Endif
```


`[String =] cell.`**`TableCellTextValue`**`(rowIndex: Long, colNameOrIndex: String/Long)`

Get table's cell text value from the specified position. The row index and column index is counted from zero (first row and first column has an index 0). The column can be specified also by the case-sensitive name of the column. The attribute name must be prefixed by '@'.

Remark: If in the table there are only simple text values without any other elements, then C0.TableCellTextValue(x, y) is the same as C0.TableCell(x, y).FirstChild.Value.

Return a string which is the text value of the cell. If the cell is not a table's row or some parameter is outside the table or the column's name doesn't exist, returned string is empty.

`[Long =] cell.`**`TableColRename`**`(colName: String, newColName: String, [disableMessages: Long/Boolean = 0])`

Rename table's column "*colName*" in the currently active table's view to the new value "*newColName*". Attribute names must be prefixed by '@'. The "*cell*" must be the table's row.
By the parameter "*disableMessages*" = 1, the eventual messages (e.g. some questions or warnings) can be suppressed.

Return a value 1 if the renaming was successful and 0 if it fails.

`[Long =] cell.`**`TableColsCount`**

Get number of columns of the table. If the cell is not a table's row, the result is 0.

`[String list =] cell.`**`TableColsNames`**

Get the string list of column names in the table. Attributes are prefixed by '@'. If the cell is not a table's row, the result is empty list.

Example:
```
LS0 = C0.TableColsNames
    If (LS0.Count > 0)
        Msg("Column's name with index 0 is " + LS0(0))
    Endif
```

`[Long =] cell.`**`TableColsSelect`**`(colNameOrIndex: String/Long, [colNameOrIndex2: String/Long])`

Select one or more table's columns in the currently active table's view. If the second parameter is used then both specified columns and all columns between are selected.
Column parameter can be the name of the column (attributes are prefixed by '@') or index of the column.

Return a number of selected columns. If the cell is not from the currently active xml file or the active page is not in a table's view or cell is not a table's row or column index/name doesn't fit, return 0.

`[Long =] cell.TableRename(newTableName: String)`

Rename the table in the currently active table's view.

Return a value 1 if the renaming was successful and 0 if it fails.

`[Cell =] cell.TableRowCell(rowIndex: Long)`

Get table's row cell. Index is counted from 0. If cell is not a table's row or index is outside the range, return 0 (null).

`[Long =] cell.TableRow`

Get table's row index. Index is counted from 0. If cell is not a table's row, return -1.

`[Long =] cell.TableRowsCount`

Get number of rows in the table. Return 0 if the cell is not a table's row.

Example:
```
I0 = 0
I1 = C0.TableRowsCount
Loop (I0 < I1)
    LC0.Add(C0.TableRowCell(I0))
    I0 = I0 + 1
EndLoop
```

`[Long =] cell.TableSelect([append: Long/Boolean = 0])`

Select the whole table in the currently active table's view.
`append` = 1: The selected table is appended to already selected block.

Return 1 if it succeeds and 0 if it fails.

`[String =] cell.Value`

Return the value of the cell. Please note that it could be the element name, attribute name, comment declaration or value of the element, attribute or comment.

Example:
```
If (C0.IsValue)
    Msg("The value of parent '" + C0.Parent.Value + "' is '" + C0.Value + "'")
Else
    If (C0.IsElement)
        Msg("Element name is '" + C0.Value + "'")
    ElseIf (C0.IsAttribute)
        Msg("Attribute name is '" + C0.Value + "'")
    Endif
Endif
```



```
[String =] cell.XPath
```

Get the XPath of the cell. If the cell is not from the currently active xml file or the active page is not in a table's view, return empty string.

29.3 Clipboard functions

These functions manipulates with the clipboard.

```
Clipboard(normal: Long/Boolean)
```

Sets clipboard to normal (=1) or private (=0) mode. In private mode the clipboard is not written or read and all clipboard operations are done only internally in the memory - data are exchanged only inside the editor itself without clipboard. This increases the performance and reliability and doesn't affect the clipboard. Sometime it's a benefit, sometimes not, depending on the task.
When the clipboard mode is set to private from the normal mode, the content of the clipboard is read and used as a starting content.

```
[Long =] Copy([type: Long = 0])
```

Copy selected area in the currently active text or table's view to the clipboard.
Parameter "type" is ignored in text view. In table view it supports different types of data copy:
0 = automatically decide how to copy
1 = copy as structured text
2 = copy as xml

Return 1 if function succeeds and 0 if it fails.

```
[Long =] Cut([type: Long = 0])
```

Cut selected area (copy and delete the area) in the currently active text or table's view to the clipboard. Parameter "type" is ignored in text view. In table view it supports different types of data copy:
0 = automatically decide how to copy
1 = copy as structured text

Return 1 if function succeeds and 0 if it fails.

```
[Long =] DragDrop(dropTarget: Cell, [copy: Long/Boolean = 0],
[append: Long/Boolean = 1], [outsideTable: Long/Boolean = 0])
```

Do drag & drop on the selected cells. Parameter "dropTarget" is the cell pointing to the drop place. Selected cell(s) can be copied or moved according the parameter "copy" (=1 for copy, =0 for move). If "append" = 1, then it is appended to target cell otherwise it's inserted before.
If the drop target is the first or the last table's row we can decide by parameter "outsideTable" if we want drop inside the table or outside the table.

Return 1 if it succeeds and 0 if it fails.

Note: some combinations of drag&drop might not be possible and returns 0 (for instance, trying to move complex element into attribute will fail).

```
[String =] ReadFromClipboard()
```

Read the text content of the clipboard.

```
[Long =] WriteToClipboard(value: String builder/String/Long
/Boolean/Double)
```

Write value as string to the clipboard. Numbers are converted to text. If the string is empty the clipboard is cleared.

Return 1 if succeeds and 0 if it fails.

```
[Long =] Paste([disableMessages: Long/Boolean = 0])
```

Paste clipboard to the currently selected place in text or table view.
Parameter "*disableMessages*" = 1 suppress eventual messages.

Return 1 if succeeds and 0 if it fails.

```
[Long =] PasteMerge(colName: string, [pasteDifferentValues:
Long/Boolean = 1], [caseSensitive: Long/Boolean = 1],
[disableMessages: Long/Boolean = 0])
```

Paste the clipboard to the currently selected cell in table's view with merge. Selected cell must be the table's row. See also [14 `Paste Merge...`]. The attribute column name must be prefixed by '@'.

Returns number of pasted rows or -1 if it fails (e.g. cell is not table's row or active view is text).

Note: The function fails while the sub-table is opened.

29.4 Dialogs

In this section are functions which support the user interaction.

```
[String list =] SelectFiles([title: String = "Select file"], [folder:
String], [multipleFiles: Long/Boolean = 0], [filter: String = "*.*"])
```

Show a dialog for selecting one or more files. The starting folder can be specified by the "folder" parameter. If the title is empty string the "Select file" or "Select files" (depends on multipleFiles parameter) is used for the dialog title.

Return string list containing full path(s) of the selected file(s) or an empty list if the dialog is cancelled.

Example: *LS0 = SelectFiles("", @"C:\Temp", 0, "*.xml")*


```
If (LS0.Count > 0)
	S0 = LS0(0)
	Msg("Selected file: " + S0)
Else
	Msg("Action is cancelled")
	Stop
EndIf
```

```
[String =] SelectFolder([title: String = "Select a folder"],
[folder: String])
```

Show a dialog for selecting a folder. The title of dialog can be specified by the first parameter and the starting folder by the second parameter.

Return the selected folder path or empty string if the dialog is cancelled.

```
[Long =] Log(value: String builder/String/Long /Boolean/Double,
[timeStamp: Long = 0])
```

Write "value" to the log (bottom area in the script mode). Before message can be prefixed time stamp. If parameter "timestamp" = 0 the nothing is prefixed, = 1 for time information and = 2 for date and time information.

Return 1 if it succeeds and 0 if it fails.

Remark: This function refreshes the log area, if function ReDraw(0) was used before, but no more than once.

```
[Long =] Msg(value: String builder/String/Long /Boolean/Double)
```

Display message box with a button 'OK' and wait. Values can be connected by a plus sign.

Return 1 if it succeeds and 0 if it fails.

Example: *Msg("Fixed text and variables: " + S0 + ", " + I0 + ", " + LS0(2) + " etc.")*

```
[Long =] MsgYesNo(value: String builder/String/Long /Boolean/Double)
```

Display message box with a button 'Yes' and 'No' and wait. Values can be connected by a plus sign.

Return 6 if 'Yes' has been selected and 7 if 'No' has been selected.

```
[Long =] MsgInput(value: String builder/String/Long /Boolean/Double,
[startValue: String = ""], [escapeValue: String = ""])
```

Display message box with a text box where an input text can be entered.

Return entered string or the "escapeValue" if the dialog wasn't closed by 'OK' button.

29.5 File manipulations

`[String =] DirFiles([folder: String])`

Get or set the working directory for files. This directory is used if the file path is a relative path.

Return a directory path which is currently used.

`[String =] DirScripts([folder: String])`

Get or set the working directory for scripts. This directory is used for functions related to script execution if the file path is a relative path.

Return a directory path which is currently used.

`[Long =] Close([fileName: String])`

Close the file without saving. If "fileName" parameter is not specified the function closes the currently active file. Otherwise it closes the file with the path "fileName" if it's opened.
Parameter "fileName" can be relative path – in such case the path "DirFiles" is used as the parent folder.

Return 1 if the file was found as opened file and it was closed. Otherwise it returns 0.

Note: The function fails while the sub-table is opened.

`[Long =] FileSide(fileName: String, [side: Long = 0])`

Get the side or set the left/right side for the specified file, which must be already opened.
Parameter "side" can specify left side (value -1), right side (value +1) or nothing (value = 0).
If the parameter "side" is 0 or not specified the function returns the position of the file only.
If the parameter "side" is 1 or -1 the file will be moved to the right side or to the left side. This function has effect only in compare mode, in other modes it does anything and the side will be always -1 (left side).

Note: The function fails while the sub-table is opened.

Return 0 if function fails, return -1 if the file is on the left side and +1 if the file is on the right side.

`[Long =] FileSize(filename: String)`

Get the file size in bytes or -1 if the file was not found.

`[String =] GetActiveFile([side: Long = 0])`

Get the full path of the currently active file. Parameter "side" = 0 if we want to get focused file, = -1 if we want to get active file on the left side, = +1 if we want to get active file on the right side.

Return full path of the file or empty string if on the left/right side there is no file. Also for parameter "side"=0 we can get empty result, if the focus is on the side where is no file.

Remark: If the command is not run in the compare mode, GetActiveFile(1) returns always empty string and GetActiveFile(-1) = GetActiveFile(0) = GetActiveFile().

```
[String list =] GetFiles(folder: String, [filter: String = "*"],
[topFolderOnly: Long/Boolean = 1])
```

Return string list containing the full paths of files from the folder which match the filter. The result can be from the directory and all subdirectories ("topFolderOnly" = 0) or only from the top directory (default). For the "filter" parameter can be used a wildcard '*' for any text and '?' for any one character.

Example:
```
LS0 = GetFiles(@"C:\Temp", "a*l.*", 1)     // files beginning with 'a' and ending by 'l'
I0 = 0
I1 = LS0.Count
SX.Clear
Loop (I0 < I1)
    SX.Append(LS0(I0) + "\r\n")
    I0 = I0 + 1
EndLoop
Msg(SX)
```

```
[String list =] GetDirectories(folder: String, [filter: String = "*"],
[topFolderOnly: Long/Boolean = 1])
```

Return string list containing the full paths of folders, which match the filter, from the top “folder”. The result can be from the directory and all subdirectories ("topFolderOnly" = 0) or only from the top directory (default). For the "filter" parameter can be used a wildcard '*' for any text and '?' for any one character.
This function can result in exception dialog if some folders are not accessible.

```
[String list =] GetOpenedFiles()
```

Return string list containing the full paths of all opened files in the editor.

```
[Long =] IsActiveFileModified
```

Return the modified flag of currently selected active file. If the file has set the modified flag, it returns 1, otherwise 0 (file is not modified).

```
[String =] Join(fileNameJoin: String, [disableMessages: Long/Boolean
= 0], [fileNameOutput: String])
```

Join the files according the “join definition file” and store them to the output file (see also [TBD]). It’s possible to hide all messages and the progress bar by “disableMessages” parameter.

Return the name of the created file or an empty string if the function fails.

Note: The function fails while the sub-table is opened.

```
[String =] NewFile([fileType: Long = 0])
```

Create a new file in the editor. The “fileType” can be one of these values:
0 = xml file UTF8, 1 = xml file Unicode, 2 = xml file UTF32, 3 = json file, 4 = text file

Return empty string if function fails or the file name f the file has been created. Be aware that after saving the file the file name will change from the short name to a full path.

Note: The function fails while the sub-table is opened.

Example:
```
S0 = NewFile(0)          // new xml file, S0 contains "NoName(1).xml"
Save()                   // now the file name is a full path, e.g. C:\Temp\NoName(1).xml
S1 = GetActiveFile()     // S1 != S0
Msg(S0 + " vs. " + S1)
```

```
[Long =] Open(fileName: String, [viewType: Long = 0] , [side:
Long = 0])
```

Open the file or select already opened file (make it active). Parameter "fileName" can be a relative path – in such case the path "DirFiles" is used as the parent folder.
The “viewType” can be one of these values:
0 = automatically determine if the file is opened in text’s or table’s view, 1 = try to open it in table’s view, 2 = open it in text’s view
Parameter "side" can determine left (-1) or right (+1) side, where the file is opened. This parameter is used only in compare mode. Default parameter 0 opens the file on the side where the focus currently is.

Return 0 if function fails, return 1 if the file has been opened in a table’s view and 2 if the file has been opened in a text’s view.

Note: The function fails while the sub-table is opened.

Remark: If you are working with many files, e.g. Open() in combination with GetFiles(), try to close the opened files as soon as possible to avoid the memory issue.

```
[String list =] ReadAllLines(fileNameRead: String, [encodingType:
Long = -1])
```

Read the content of the file from the disk. Parameter "fileNameRead" can be a relative path – in such case the path "DirFiles" is used as the parent folder.
The “encodingType” can be one of these values:
-1 = automatically determine the encoding, 0 = UTF8, 1 = Unicode, 2 = UTF32, 3 = ASCII, 4 = UTF7

Return the string list, each record is one line of the file. If the file doesn't exist or the reading fails, return empty list.

`[String =] RunScriptName`

Get the full path of the currently running script file.

`[Long =] WriteAllLines(fileNameWrite: String, content: String list, [encodingType: Long = -1])`

Write the content (string list) to the file. Parameter "fileNameWrite" can be a relative path – in such case the path "DirFiles" is used as the parent folder.
If the target file already exists, it is overwritten.

Return 1 if succeeds and 0 if it fails.

`[Long =] Save([newFileName: String])`

Save currently active file. If the "newFileName" is specified, then it's saved with a new name (save as). New file name can be a relative path – in such case the path "DirFiles" is used as the parent folder.

Return 1 if succeeds and 0 if it fails.

Note: The function fails while the sub-table is opened.

`[Long =] Split(fileToSplit: String, [disableMessages: Long/Boolean = 0], [fragmentSizeMB: Long/Double = 4], [folderOutput: String])`

Split the target file into fragments of the approximate size "fragmentSize" (in MB). The range is 0.1 to 100 (MB) for 32bit application and 0.1 to 750 (MB) for 64bit application.
It's possible to hide all messages and the progress bar by "disableMessages" parameter.

Return number of created parts. Return 0 or negative number of already created parts if some error happened.

Note: The function fails while the sub-table is opened.

29.6 Find and replace

In this chapter are functions supporting the actions with "find and modify" or "find and replace" dialogs. For these functions are dedicated a special variable **F** - find settings (see also [23.6 `Script mode`]).
Whenever a parameter "findWhat" or "newText" is used, all wildcards, which are used in the editor's Find dialog, are allowed.

[Long =] **F.EndCycle**

This function can be called after the find (find next) operation to determine if the end of cycle was reached, and the finding has reached the starting point again.

Return 1 if the end of cycle is reached, otherwise 0.

[Cell =] **F.FindReplace**(findWhat: String, [replaceWith: String])

Find the text in the table's view using the options which can be set by other functions. Searched and replaced string is formed in the same way like in the editor (see [20.3 Find and Replace]). The wildcards can be used as well if the option "use wildcards" is enabled in the function "SetOptions".
If the parameter "replaceWith" is used, then a replacement of the already found text is done and then a next occurrence of the "findWhat" text is found.
If you want to do only "find", don't use the second parameter.

Return cell where the text is found or 0 (null) if nothing is found.

Example:
```
Open(@"C:\Temp\File.xml")
    F.SetOptions(0, 0, 0, 0, 0, 1, 0)
    C0 = F.FindReplace("Temperature")
    I0 = 1
    Loop(Not F.EndCycle)
        If (C0 != 0)
            // replace text "Temperature" by "T1","T2"...
            C0 = F.FindReplace("Temperature", "T" + I0)
            I0 = I0 + 1
        Else
            Break
        EndIf
    EndLoop
    Msg("Number of replacements: " + (I0 - 1))
```

[String =] **F.FindReplaceInText**(findWhat: String, [replaceWith: String])

Find the text in the text's view using the options which can be set by other functions. Searched and replaced string is formed in the same way like in the editor. The wildcards can be used as well if the option "use wildcards" is enabled in the function "SetOptions".
If the parameter "replaceWith" is used, then a replacement of the already found text is done and then a next occurrence of the "findWhat" text is found.
If you want to do only "find", don't use the second parameter.

Return found text or empty string if nothing is found.

Example:
```
Open(@"C:\Temp\File.xml", 2)        // open in text mode
    F.SetOptions(0, 0, 0, 0, 0, 1, 0)
```



```
S0 = F.FindReplaceInText("Temperature")
I0 = 1
Loop(Not F.EndCycle)
	If (S0 != "")
		S0 = F.FindReplaceInText("Temperature", "T" + I0)
		I0 = I0 + 1
	Else
		Break
	EndIf
EndLoop
Msg("Number of replacements: " + (I0 - 1))
```

`[Cell =] F.FindModify(findWhat: String, [action: Long], [name: String = ""], [value: String = ""])`

Do a "find and modify" functionality (see also [20.4 `Find and Modify`]) in table's view. If the parameter "action" is not used, only the "next find" is done.

The parameter "action" has this meaning:
1 = Delete; 2 = Insert attribute; 3 = Insert element; 4 = Insert child attribute; 5 = Insert child element; 6 = Append attribute; 7 = Append element; 8 = Append child attribute; 9 = Append child element; 10 = Paste

Return the cell of next finding or 0 (null) if nothing next is found.

`[Long =] F.ModifyAll(findWhat: String, action: Long, [name: String = ""], [value: String = ""])`

Do "modify all" functionality in table's view.
The parameter "action" has this meaning:
1 = Delete; 2 = Insert attribute; 3 = Insert element; 4 = Insert child attribute; 5 = Insert child element; 6 = Append attribute; 7 = Append element; 8 = Append child attribute; 9 = Append child element; 10 = Paste

Return number of modifications.

`[Cell list =] F.FindAll(findWhat: String)`

Do "find all" functionality in table's view. Return list of cells where 'findWhat' have been found. If there are multiple occurrences of the searched text in the same cell, such cell is in the list repeated in the number of occurrences. The same cells are always grouped together in the list. So we can know how many occurrences have been found in the cell if we test its followers in the list.

`[Long =] F.ReplaceAll(findWhat: String, newText: String)`

Do "replace all" functionality and return number of replacements.

`[Long =] F.FindSortColumn(findWhat: String)`

Do "Sort->Find" functionality in table's view. Return number of rows in which the appropriate column cell (or any of its children) contains the 'findWhat' text. All these table's rows are moved up to the top of the table. In principle it can be viewed as a partial filtering.

`[Long =] F.SetModifiers(index: Long, offset: Long, [stepOffset: Long = 0], [accumulated: Long = 0], [minWidth: Long = 0])`

Set numeric modifiers related to the hash wildcard character(s) '#'. These modifiers can be used for replacement. See [20.3.3 `Replacing using number modifiers`] for details.
Parameter "index" is a number 1-4. It's possible to define up to 4 modifiers.

This function always returns 1.

29.7 Find/Lookup settings

Here are functions which define parameters for findings and for the lookup table.
Special variables **F** (find settings) and **T** (lookup table settings) can be used.
All functions in this section always returns 1.

`F/T.SetOptions([useWildcards: Long/Boolean = -1], [inSelection: Long/Boolean = -1], [caseSensitive: Long/Boolean = -1], [startWith: Long/Boolean = -1], [entireCell: Long/Boolean = -1], [skipNotEditable: Long/Boolean = -1], [modifiersActive: Long/Boolean = -1])`

Set options for find and replace functionalities. Parameters can be 1/true or 0/false or -1 if the parameter shouldn't be changed.

Parameters:

1. use wildcards (for lookup table this parameter is ignored)
2. find in selection
3. case sensitive searching
4. start with (ignored in text's view searching)
5. entire cell (ignored in text's view searching)
6. skip not editable and not related cells (ignored in text's view searching); this value is true if 'find in selection' is true
7. counter modifiers (wildcard #) are active

`F/T.SetParents([parentAttributeName: String], [parentElementName: String], [grandfatherElementName: String])`

Set options for find functionalities. If we don't want to use some of these criteria, assign the empty string for appropriate parameter.

F/T.SetRestrictions([resultCanBeAttribute: Long/Boolean = -1], [resultCanBeElement: Long/Boolean = -1], [resultCanBeValue: Long/Boolean = -1], [resultCanBeComment: Long/Boolean = -1], [resultCanBeJsonText: Long/Boolean = -1], [resultCanBeJsonNumber: Long/Boolean = -1])

Set options for find functionalities. Parameters can be 1/true or 0/false or -1 if the parameter shouldn't be changed.

T.SetLookupTable(openedFileName: String, attributeName: String, elementName: String, [findWhat: String = "%"])

Set parameters for the lookup table replacement. For details see the section [20.3.2 Replacing using a lookup table].

29.8 Lists

Functions for variables of the list type (long lists, string lists and cell lists).
In this scope the type "*List*" means any one of the List Long, List String or List Cell and the type "*LSC*" means any one of the Long, String or Cell.

[Long =] list.**Add**(valueOrRange: LSC/List)

Add a value or the range of values to the list. The type of the added value(s) must be the same as the target list.

Return number of added items.

Example: *I0 = LS0.Add("hello")* *// I0 = 1*
I0 = LS0.Add(LS1) *// I0 = LS1.Count*

[Long =] list.**Clear**

Clear the list and return 1 if it succeeds or 0 if it fails.

[Long =] list.**Count**

Get the number of items in the list.

[Long =] list.**Del**(startIndex: Long, [endIndex: Long = startIndex])

Remove value(s) from the "startIndex" to "endIndex" from the list. Indexes are counted from zero. If "endIndex" is less than "startIndex" nothing is removed.

Return number of removed items.


```
[Long =] list.IndexOf(findValue: LSC, [startIndex: Long = 0],
[endIndex: Long = list.Count - 1])
```

Find the first occurrence of the "findValue" in the list. Search is case sensitive; indexes are counted from zero. The "startIndex" should be less or equal than "endIndex".

Return index in the list where the "findValue" has been found or -1 if nothing has been found.

```
[Long =] list.LastIndexOf(findValue: LSC, [startIndex: Long =
list.Count - 1], [endIndex: Long = 0])
```

Find the last occurrence of the "findValue" in the list. Search is case sensitive; indexes are counted from zero. The "startIndex" should be greater or equal than "endIndex".

Return index in the list where the "findValue" has been found or -1 if nothing has been found.

```
[LSC =] list.Pop
```

Remove the last value from the list and return that value. If the list is empty, then a zero or an empty string is returned depending on the list type.

```
[List =] list.Range(startIndex: Long, endIndex: Long)
```

Get range of values from the list or an empty list. Indexes are counted from zero. The "startIndex" should be less or equal than "endIndex".

29.9 Miscellaneous functions

```
[Long =] Delay(milliseconds: Long)
```

Wait for specified number of milliseconds. Number must be greater than zero.

Return 1 if it succeeds and 0 if it fails (time is less or equal zero).

```
[Long =] Delete([hardDelete: Long/Boolean])
```

Delete selection in text's or table's view. Parameter "hardDelete" is ignored in the text's view.

Return 1 if it succeeds and 0 if it fails.

```
[Long =] DoCompare()
```

Compare active files. The compare mode must be already set by script before this command and files must be arranged one on the left side and one the right side.

Return 0 if files are equivalent, 1 if files are different and -1 if function fails (e.g. not in compare mode or one file is in text view and the second one is in table's view).

```
[Long =] DoValidate()
```

Validate the active file. The validation mode must be already set by script before this command.

Return: 0 if validation passed without any warnings or errors
1 if there is some warning but without any error
2 if there is some error
-1 if it fails

```
[Long =] Mode([mode: Long = 0])
```

Get or set the editor's mode. Parameter "mode" can be one of these values:
0 = do not change the mode, only get information what is current mode
1 = set standard mode
2 = set compare mode
3 = set validation mode
4 = set script mode

Return mode (1-4) if it succeeds and 0 if it fails.

```
Redraw(enable: Long/Boolean)
```

Enable or disable redrawing. Disabling can increase a speed of the script execution.

```
[Long =] SendKey(key: String, [control: Long/Boolean = 0], [shift:
Long/Boolean = 0], [alt: Long/Boolean = 0])
```

Send one key to the currently active table's or text's view. This can't be used for sending a key to the external application.
Parameter "key" is case sensitive text describing one key (e.g. Enter).
Some frequently used keys:
Back, Delete, Down, Enter, Escape, Home, Insert, Left, PageDown, PageUp, Right, Space, Tab, Up

For more details you can check the Microsoft documentation for Keys enumeration from the System.Windows.Forms namespace.

Return 1 if it succeeds and 0 if it fails.

29.10 Strings

This chapter describes available functions for string manipulations.

```
[Long = ] string.Contains(findWhat: String, [caseInsensitive:
Long/Boolean = 0])
```

Test if the string contains the entered string “findWhat” as a substring.

Return 1 if the string contains the "findWhat" or 0 if it doesn’t contain the substring.

```
[Long = ] string.EndsWith(findWhat: String, [caseInsensitive:
Long/Boolean = 0])
```

Test if the string ends with the entered string “findWhat”.

Return 1 if it ends with the "findWhat" or 0 if it doesn't end with the text,

```
[Long = ] string.FirstFind (findWhat: String, [startIndex: Long = 0],
[endIndex: Long = string.Length - 1], [caseInsensitive: Long/Boolean
= 0])
```

Find the first occurrence of the “findWhat” in the string. Search is case sensitive; indexes are counted from zero. The “startIndex” should be greater or equal than “endIndex”.

Return index in the string where the “findWhat” has been found or -1 if nothing has been found.

```
[Long = ] string.LastFind (findWhat: String, [startIndex: Long =
string.Length - 1], [endIndex: Long = 0], [caseInsensitive:
Long/Boolean = 0])
```

Find the last occurrence of the “findWhat” in the string. Search is case sensitive; indexes are counted from zero. The “startIndex” should be lesser or equal than “endIndex”.

Return index in the string where the “findWhat” has been found or -1 if nothing has been found.

```
[Long = ] string.Length
```

Get the length of the string variable or of the string builder.

```
[String = ] string.LowerCase
```

Convert characters of the string to their lower case representation, invariantly of the current culture. This function might be use if you want compare two strings case insensitive.


```
[List string = ] string.Split(separator: String, [removeEmptyEntries:
Long/Boolean = 0])
```

Split string into pieces divided by a separator string. If parameter "removeEmptyEntries" is 1(true) the empty fragments are removed from the result.
For instance, split with a separator string "/" applied on the string "/root/" creates list { "", "root", "" } if "removeEmptyEntries" = 0.

Return list of strings.

```
[String = ] string.StrReplace(findWhat: String, newText: String)
```

Replace all occurrences of the "findWhat" string with "newText" string.

```
[Long = ] string.StartWith(findWhat: String, [caseInsensitive:
Long/Boolean = 0])
```

Test if the string starts with the entered string “findWhat”.

Return 1 if it starts with the "findWhat" or 0 if it doesn't start with the text,

```
[String = ] string.SubString(startIndex: Long, endIndex: Long)
```

Get the substring from the string variable. Indexes are counted from zero.

```
[String = ] string.UpperCase
```

Convert characters of the string to their upper case representation, invariantly of the current culture. This function might be use if you want compare two strings case insensitive.

29.11 StringBuilder

String builder variable SX is dedicated for building a long string by appending strings (e.g. for creating a new file content from string lines).

```
[Long = ] SX.Append(value: String builder/String/Long /Boolean/Double)
```

Append text (numeric values are converted to string) to the string builder.

Return 1 if it succeeds or 0 if it fails.

```
[Long = ] SX.Clear()
```

Clear the string builder. It will contain empty string.

Return 1 if it succeeds or 0 if it fails.


```
[Long = ] SX.ReadFromFile(fileName: String, [encodingType: Long = -1])
```

Read the file to the string builder. Parameter "fileName" can be a relative path – in such case the path "DirFiles" is used as the parent folder.
The “encodingType” can be one of these values:
-1 = automatically determine the encoding, 0 = UTF8, 1 = Unicode, 2 = UTF32, 3 = ASCII, 4 = UTF7

Return number of characters read or -1 if it fails.

```
[Long = ] SX.WriteToFile(fileName: String, [encodingType: Long = -1])
```

Write the file from the string builder. If the file exists, it's overwritten. Parameter "fileName" can be a relative path – in such case the path "DirFiles" is used as the parent folder.
The “encodingType” can be one of these values:
-1 = not used, 0 = UTF8, 1 = Unicode, 2 = UTF32, 3 = ASCII, 4 = UTF7

Return 1 if it succeeds or 0 if it fails.

29.12 Xml functions

This includes some of the functions which are available in the table's view. They are working in the currently active table's view.

```
[Long = ] Collapse(action: Long, [cell: Cell = 0])
```

Collapse or expand selected or specified cell or selected area in the currently active table's view.
Parameter "action":
-3=expand selection, -2=expand level, -1=expand last selected cell,
1=collapse last selected cell, 2=collapse level, 3=collapse selection, 4=collapse unselected
Parameter "cell" can specify a target cell, which should be expanded or collapsed. It can be used only for actions -2, -1, 1 or 2.
If the cell is used for actions 1,-1, the selected area doesn't change (it corresponds to mouse clicking on the expand/collapse button).
If the cell is not used for actions 1,-1 the focus is changed to the specified switched cell (it corresponds to press Enter over the cell).

Return 0 if something fails, otherwise return 1 (but it doesn't necessarily mean that something has been collapsed or expanded, only that the command was executed without any issue).

```
[Long = ] Comment([action: Long = 1])
```

Comment (action = 1) or uncomment (action = 0) elements or values in the selected area of the currently active table's view.

Comment moves the element(s) or value(s) into an xml comment block. It tries to group selected elements in one block if it's possible. Otherwise it creates multiple comments (for instance if a table's column is selected).

If only a value of an element is selected, it creates a comment inside such element.
Attributes can't be commented.

Uncomment does the reverse function. The xml structure contained in the comment value is extracted (if the xml structure is correct) and placed instead of the comment. This function ignores all non-comment cells in the selected area.
If a value in the comment is a text, which can't be converted to xml structure, it tries to assign it to the element's owner as a new value if that element doesn't have a value yet or if it has empty value. Otherwise it does nothing.

Return number of changes or zero if nothing has been changed.

```
[Long = ] ConvertValues(action: Long)
```

Do "Convert values" action in the selected area in the currently active table's view. See a section [10.1 `Convert values (in Selection)`] for additional information.
Parameter "action":
0=convert no values to blank, 1=convert blank values to no values,
2=convert base64 text to text, 3=convert text to base64 text,
4=fill blank cells by "{GU-ID}", 5=fill blank cells by "{gu-id}",
6=fill blank cells by "GUID", 7=fill blank cells by "guid",
8=convert json text to value, 9=convert json value to text
10=decompress text, which was compressed by Gzip, 11=compress text by Gzip (Gzip compresses text and the result (in general binary result) is encoded as base64 text
12=xml beautify (if value contains xml structure, normalize the spaces)

Return 0 if something is not correct, otherwise return 1 (but it doesn't necessarily mean that something has been changed).

```
[Long = ] Duplicates([action: Long = 0], [ignoreOrder: Long/Boolean
= 1])
```

Do "Duplicates" action in the selected column(s) in the currently active table's view. See a section [5.1 point 14. Find duplicates in a table by selected columns] for additional information.
Parameter "action":
0=count number of duplicates
1=group duplicates together and move them to the end of the table
2=remove duplicates from the table
Options 1 and 2 are not allowed in the main sub-table table.

Return number of duplicates found or -1 if something fails.

Note: The function fails while the sub-table is opened for "action" not equal 0.

```
[Long = ] FillColumn([startValue: Long/Double = 1], [stepValue:
Long/Double = 1])
```

Fill the selected table's column in the currently active table's view by numbers. It's possible to enter also real values (e.g. start from value 0.9 with the step -0.01).

Return number of table rows or 0 if it fails.

`[Cell = ] GetSelectedCell`

Get the selected cell (top left corner if the area is selected) in the currently active table's view.

`[Cell = ] Insert([append: Long/Boolean = 0], [type: Long = 0])`

Insert or append an element or attribute or comment or table's row or table's column depending on what is selected in the currently active table's view. By default it inserts an element (if table's row or table's column is not selected).
Parameter "type" equals 0 for an element, 1 for an attribute and 2 for a comment.

Return the added cell or 0 (null) if it fails. The cell could be the added element or attribute or comment or table's row cell or for added table's column it returns the cell in the new column from the first row of the table. By the function cell.SetValue we can set a new name of the element or attribute. For a table rename or a column rename there are special functions.

`[Cell = ] InsertTable([attributeColumns: Long = 0], [elementColumns: Long = 1], [append: Long/Boolean = 0], [tableName: String])`

Insert or append a new table in the currently active table's view to the place with the selected cell.
See also a section [16.1 `Insert / Append Table`] for additional information.

Return the first row cell of the new table or 0 (null) if it fails.

`[Long = ] MoveSelRowCol(shift: Long)`

Move selected row(s) or column(s) in the currently active table's view by "shift" position. Parameter can be positive (move down or to the right) or negative (move up or to the left).

Return 1 if it succeeds or 0 if it fails.

`[Long = ] NormalizeSpaces(numberOfSpaces: Long, [numberOfTabs: Long = 0], [endLineType: Long = 0])`

Normalize spaces in the currently active table's view. If "numberOfTabs" is non-zero, then "numberOfSpaces" is ignored. If both parameters are zero or less than zero, then 1 space is used.
Parameter "endLineType" can be 0=CRLF, 1=LF, 2=CR.
See also a section [20.8 `Normalize spaces`] for additional information.

Return 1 if it succeeds or 0 if it fails.

`[Long = ] SetSelCellValue(newValue: String/Long/Boolean/Double)`

Set a new value (if it's possible) for currently selected top left cell in the currently active table's view.

Return 1 if it succeeds or 0 if it fails (e.g. newValue is not valid value for element name or cell is not editable or not in table's view and so on).

`[Long = ] SortColumn(sortType: Long)`

Sort the selected table column in the currently active table's view.
Parameter "sortType":
1 = sort as text ascending, -1 = sort as text descending,
2 = sort as numbers ascending, -2 = sort as numbers descending

Return 1 if it succeeds or 0 if it fails.

Note: The function fails while the sub-table is opened.

`[Long = ] SubtableClose()`

Close the top sub-table window.

Return 1 if sub-table has been closed or 0 if nothing has been closed.

`[Cell = ] SubtableOpen()`

Open a sub-table for a selected table's column in the currently active table's view.

Return the top table's cell (the same which you can get by the function XmlRoot after the sub-table is opened) from the new open sub-table or 0 (null) if something fails.

`[Cell = ] XmlRoot`

Get the root element from the currently active table's view or 0 (null) if it fails.

`[Cell list/Cell = ] XpathQuery(xPath: String)`

Evaluate XPath query in the currently active table's view. If we want to target text value we can use the directive "/text()". It can be used for an element and for an attribute as well, although for attribute it's not a part of the xml specification (it's for internal purposes of the program only).

Result is returned in the cell list, but it's also possible to assign the result to the cell. In such case the first cell from the result list is taken (if the list is empty the 0 (null) is assigned to the cell.
Example: *LC0 = XpathQuery("/Root/Element1/*") // in this case we expect more than 1 result*
C1= XpathQuery("/Root/Element1") // now we expect only one result,
// but if the Element1 is table, then the result can have more records
// but using this assignment will throw away all others except the first one

29.13 Script examples

29.13.1 Demo example

On the site http://ximple.cz/files/ximple_video.mp4 or here https://www.youtube.com/watch?v=ceXLC02d-rQ there is a video, which demonstrates some tasks.
In the next chapter there is presented a script which does the same example tasks.
The file "Demo.xml" could be downloaded as a zip file here: http://ximple.cz/files/Demo.zip

Hint: You can insert a Msg("") command or stop command to any place in a code to see what happened until the place where you stop it.

```
DirFiles(@"D:\Scripts")             // set default path for files
s1 = "Demo.xml"                     // name of the file

//Redraw(0)               // this command can be enabled for better performance



/* Task 1: > Identify the missing variables < */

if (Open(s1))                       // is file opened?
  lc0 = XPathQuery("/Demo/Private/Variables[1]") // target cell in the xml structure
  if (lc0.Count > 0)                // does x-path query return any result?
      c0 = lc0(0)                   // get the first item from the list
      if (c0.IsTable)               // is C0 a table's row?
         if (c0.TableColsSelect("Key")!=0) // select the column 'Key' from the table
           c1 = SubTableOpen()      // open the sub-table on the column
           if (c1 <> 0)             // is sub-table opened?
              c1.TableColsSelect("value") // select column 'value' in the sub-table
              Copy()                // copy selected values to the clipboard
              SubTableClose()       // close the sub-table
           else
              Msg("Opening a sub-table failed.")
              stop
           endif
         else
            Msg("Not found the column 'Key' in the table.")
            stop
         endif
      else
         Msg("Cell on the path '/Demo/Private/Variables[1]' is not a table.")
         stop
      endif
  else
      Msg("XpathQuery '/Demo/Private/Variables[1]' fails.")
      stop
  endif

  s2 = NewFile()                    // create a new xml file and get its name
  c0 = XmlRoot()                    // get the root cell
  c0 = c0.FirstChild                // get the first child
  if (c0 <> 0)                      // does the child exist?
      c0.Select()                   // select the first child
      // insert "MyTable" with 3 element columns before the selected cell
```



```
      c9 = InsertTable(0, 3, 0, "MyTable")
      // c9 should be the first table's row - remember it
      ls0 = c9.TableColsNames       // get 3 column names
      s8 = ls0(0)                   // get the name of the first column
      c0 = c9.TableRowCell(c9.TableRowsCount - 1)  // get the last row of the table
      c0.Select()                   // select the last row of the table
      Paste()                       // paste the text from the clipboard
   else
      Msg("Missing the first child in the new file.")
      stop
   endif

   Open(s1)                         // select original file
   c0 = XPathQuery("/Demo/Public/Variables[1]") // target cell in the xml structure
   if (c0 <> 0)                     // does x-path query return the result?
      if (c0.IsTable)               // is C0 a table's row?
         c0.TableColsSelect("Name")  // select the column 'Name' from the table
         Copy()                     // copy selected values to the clipboard
      else
         Msg("Cell on the path '/Demo/Public/Variables[1]' is not a table.")
         stop
      endif
   else
      Msg("XpathQuery '/Demo/Public/Variables[1]' fails.")
      stop
   endif

   Open(s2)                         // select the created new file
   c9.Select()                      // select the first row of the "MyTable" table
   i0 = PasteMerge(s8)
   /*
      Paste with merge and use the first column of the table for merge;
      other parameters are default: paste only different values, case sensitive
      and disable messages;
      I0 is the number of added rows at the top of the table (because the first
      row is selected)
   */
   c0 = c9.TableRowCell(i0 + 2)  // get the row i0 + 2 (counted from zero) of the
table
                     // there are two empty rows which we keep in case that I0 <= 1
   c0.Select()                      // select this row
   i1 = c9.TableRowsCount           // get number of rows of the table
   c1 = c0.TableRowCell(i1 - 1)                    // get the last table's row
   c1.Select(1)                     // append the last row to the selection
                  // the rows from I0 till the end of the table should be selected
   Delete()                         // delete all selected rows
   // only the first I0 + 2 rows remain - the I0 rows were pasted with merge
   // these are the missing values we are looking them for

   // cell c9 is no more the first row, because it was shifted by paste
   // update c9 to be again the first row
   c9 = c9.TableRowCell(0)

   /* Task 2: > Add missing variables with respect to existed xml structure < */

   Open(s1)                         // select original file
   lc0 = XPathQuery("/Demo/Private/Variables[1]") // target cell in the xml structure
   if (lc0.Count > 0)               // does x-path query return any result?
```



```
   c8 = lc0(0)                    // get the first item from the list
   ls2 = c8.TableColsNames()      // remember column names
   c8.Select()                    // select the row
   Copy()          // copy one row (we know there are two columns) as a pattern
else
   Msg("XpathQuery '/Demo/Private/Variables[1]' fails.")
   stop
endif

Open(s2)                          // select the created new file
// select 2nd and 3rd column of the "MyTable" for pasting the pattern
c9.TableColsSelect(1, 2)
Paste()        // paste two columns from /Demo/Private/Variables[1] to all rows
c9.TableColsSelect(0)             // select the first column (name is s8)
Copy()                            // copy content of the first column
c9.TableColsSelect(1)             // and select the second column

c0 = SubTableOpen()               // open sub-table on the selected 2nd column
                                  // c0 is the root cell from the sub-table
/*
  While in a sub-table we are in a different "active file";
  that's why the operations targeting for instance c9 will not do anything,
  because c9 is not from the current content
*/
c0.TableColsSelect(1)             // select the second column of the sub-table
Paste()                           // paste the values
SubTableClose()                   // and close the sub-table
// now we are back in the content where c9 can be used
ls0 = c9.TableColsNames()         // get column names of "MyTable"
c9.TableColRename(ls0(1), ls2(0)) // rename 2nd column to "Key"
c9.TableColRename(ls0(2), ls2(1)) // rename 3rd column to "Value"
// select the second and third column of the table without the last two rows
i0 = c9.TableRowsCount - 3
if (i0 >= 0)                      // is there at least one row to copy?
  c0 = c9.TableCell(0, 1)         // get cell from the 1st row and 2nd column
  // get cell of the third row from the end and the third column
  c1 = c9.TableCell(c9.TableRowsCount - 3, 2)
  c0.Select()                     // select c0
  c1.Select(1)                    // to the selection append the cell c1
  Copy()                          // copy selected two columns

  Open(s1)                        // now back to original file
  i0 = c8.TableRowsCount          // get number of rows
  c0 = c8.TableRowCell(i0 - 1)    // because we want to get the last row
  c0.Select()                     // and select it
  Paste()           // to append the clipboard data to the end of the table
else
  Open(s1)                        // nothing to paste, go back to original file
endif

/* Task 3: > Create 'IsTorque' column < */

c8.TableColsSelect(0) // select the first column of /Demo/Private/Variables table
// we want to copy values 'Demo/Private/Variables/Key/value' for later use
c0 = SubTableOpen()               // open sub-table
i1 = c0.TableRowsCount            // get number of rows of the sub-table
ls0.Clear()                       // clear string list to collect values
i0 = 0                            // i0 is used as a loop variable
```



```
  Loop (i0 < i1)                    // go through all rows
     // collect values of the element "Key" for each row
     ls0.Add(c0.TableCellTextValue(i0, "value"))
     i0 = i0 + 1                    // increase row index
  EndLoop
  SubTableClose()      // we have collected values, we can close sub-table

  c8.TableColsSelect(1)
  // select the second column where we want to add a 'IsTorque' column
  c0 = SubTableOpen()               // open sub-table for that second column
  c0.TableColsSelect(1)   // select the second (coincidence) column in the sub-table
  c0 = SubTableOpen()               // and open another sub-table again
  c0.TableColsSelect(0)   // in this second sub-table select the first column
  c1 = Insert(1)                    // function Insert depends on what is selected
  // because the column is selected we are appending a new column
  c0.TableColRename(c1.Value, "IsTorque") // rename the created column to 'IsTorque'

  // fill whole column ‘IsTorque’ by values true or false
  i1 = c0.TableRowsCount            // get number of rows of the sub-table
  i0 = 0                            // reset loop variable
  Loop (i0 < i1)                    // go through all rows
     // get 'IsTorque' cell from the table of the I0-th row
     c1 = c0.TableCell(i0, "IsTorque")
     c2 = c1.FirstChild             // we know that this new column has only one
                                    // child and this child is the text value
     If (ls0(i0).Contains("torque", 1))  // use the list ls0 with prepared values
                                    // and check if the value contains substring
                                    // 'torque' (case insensitive)
        c2.SetValue("true")         // if the value contains 'torque' set the value
                                    // of the column 'IsTorque' to true
     Else
        c2.SetValue("false")        // otherwise set it to 'false'
     EndIf
     i0 = i0 + 1                    // increase index of the row
  EndLoop                           // repeat loop and go to Loop instruction

  // we are done, close both sub-tables and also temporary file we have created
  SubTableClose()
  SubTableClose()
  Close(s2)

  /* go to the first table and collapse it to be able to see that also second part
  (private) has now the same number of rows */
  c0 = XPathQuery("/Demo/Public/Variables[1]")
  c0.FirstChild.Select              // in case the table is collapsed, we expand it
  SendKey("Left")                   // jump from first column to the row
  SendKey("Left")                   // jump from first row to whole table
  SendKey("Enter")                  // collapse the table
else
  Msg("Can't open the file " + DirFiles + "\\" + s1)
endif

//Redraw(1)              // an optional pair command for Redraw(0)
```


29.13.2 Go through xml structure – recursion example

The script provides an additional option to look in the xml structure on lower level than we have on graphical interface. This example provides some idea how to do it.
We are using two script files: one is the routine, which is recursively called, the second is the main block which starts the recursion and handle the result.

Main file content:

```
DirFiles(@"D:\Scripts")             // set default path for files
s1 = "Demo.xml"                     // name of the file

WriteToClipboard("")                // clear the clipboard
if (Open(s1))
  // prepare variables c0 and i9 for the subroutine
  c0 = XmlRoot                      // get the root element
  i9 = 0                            // depth in the xml structure
  RunScript("Recurse.txt")          // call another script
  WriteToClipboard(sx)              // write the string builder result to clipboard
else
  Msg("Can't open the file " + DirFiles + "\\" + s1)
Endif
```

Recurse.txt content:

```
// Subroutine expects these parameters:
// c0 = parent cell
// i9 = depth structure level

// create spaces according to the level depth
i1 = 0
s0 = ""
loop (i1 < i9)
   sx.Append("  ")
   i1 = i1 + 1
endloop

// create text for values (special handling for text values and attributes)
s1 = c0.Value
if (c0.IsAttribute And Not c0.IsValue)
  s1 = "@" + s1
endif
if (c0.IsValue)
  s1 = "\"" + s1 + "\""
endif
sx.Append(s1 + "\r\n")              // collect values in one big string
// you can try how much slower it will be if you use a string variable
// instead of the string builder

c1 = c0.FirstChild                  // go one level down to the first child
i9 = i9 + 1

loop (c1 <> 0)                      // go through all siblings
  lc0.Add(c0)                       // push c0; the variable context doesn't
  // protect variables when running a sub-script, that's why we protect important
  // variables in the list
```



```
  lc0.Add(c1)                         // push c1
  c0 = c1                             // assign new parent cell
  RunScript("Recurse.txt")            // call this routine again recursively with
                                      // new parameters c0 and i9
  c1 = lc0.Pop()                      // pop c1; pick up the stored values back
  c0 = lc0.Pop()                      // pop c0
  c1 = c1.NextSibling     // go to next sibling (next child of the parent cell c0)
endloop

i9 = i9 - 1               // go one level up in the depth – return from sub-script
```

29.13.3 Go through xml structure – loop example

This example should produce the same results as the previous one, but the approach is different. Instead of recursive calling we are going through the cells by the function NextCell. This approach is faster but maybe not so intuitive and comfortable.

```
DirFiles(@"D:\Scripts")             // set default path for files
s1 = "Demo.xml"                     // name of the file

WriteToClipboard("")                // clear the clipboard


if (Open(s1))
  c0 = XmlRoot                      // start with the root element
  i9 = 0                            // depth in the xml structure

  loop (c0 <> 0)                    // while we are not at the end of the structure
    // create spaces according to the level depth
    i1 = 0
    loop (i1 < i9)
      sx.Append("  ")
      i1 = i1 + 1
    endloop

    // create text for values (special handling for text values and attributes)
    s1 = c0.Value
    if (c0.IsAttribute And Not c0.IsValue)
      s1 = "@" + s1
    endif
    if (c0.IsValue)
      s1 = "\"" + s1 + "\""
    endif
    sx.Append(s1 + "\r\n")

    c1 = c0
    c0 = c0.NextCell
    if (c0.Parent == c1)
      // go 1 level down
      i9 = i9 + 1
      lc0.Add(c1)                   // store the parent cell
    elseif (c1.Parent == c0)
      // go 1 level up
      i9 = i9 - 1
      lc0.Pop()      // in this case the same could do also lc0.Del(lc0.Count-1)
    elseif (c1.Parent == c0.Parent)
      // we are on the same level
```



```
    else
      // we have skipped several levels up
      i9 = lc0.IndexOf(c0.Parent)   // find the correct level by parent
      i9 = i9 + 1
      lc0.Del(i9, lc0.Count - i9)   // throw away what we don’t need
    endif
  endloop
  WriteToClipboard(sx)              // write the string builder result to clipboard
else
  Msg("Can't open the file " + DirFiles + "\\" + s1)
endif
```

