XMLmind XML Editor - How to adapt "Paste from Word" to your needs

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

February 22, 2019

Abstract

This document explains how the "Paste from Word" feature works, how to customize the XML it generates, how to integrate it into configurations other than XHTML, DocBook and DITA.

Table of Contents

Please contribute to improving the "Paste from Word" feature

- If you are not satisfied with the results of "Paste from Word", please be kind enough to send your `.doc` or `.docx` file to <xmleditor-info@xmlmind.com> (unlike <xmleditor-support@xmlmind.com>, this email address is not a public mailing list). Please understand that collecting as many difficult cases as possible is absolutely needed to improve this feature.

- If you are using MS-Word *with a locale other than English and French* and find that "Paste from Word" does not work so well with table and figure captions and/or with standard styles such as "Quote", "Subtle Emphasis", "Strong", etc, this probably means that the *`addon_install_dir`*`/xed/aliases.txt` database does not contain entries for your locale yet. In such case, you may want to add style name aliases to this text file and send us your modified `aliases.txt`. Or more simply, you may want to send us the `.doc` or `.docx` file created using your copy of MS-Word, so we can enhance `aliases.txt`.

1. How "Paste from Word" works

1.1. Command `pasteFromWord`

The "Paste from Word" add-on declares a generic command called `pasteFromWord`. The "Paste from Word" feature consists in invoking this command, typically from a menu item, with a parameter which configures the command for the type of the document being edited.

Excerpts from configuration file *addon_install_dir*`/docbook5/docbook5.incl`:

```
<menu label="-" insert="ifNotDefined(os.name*=Linux) _para">
  <item label="Paste from _Word" icon="../common/paste_from_word.png"
    command="pasteFromWord"
    parameter="[xmlns:db=http://docbook.org/ns/docbook]
      [after db:para]

      [xmlns:pfw=java:com.xmlmind.xmleditext.paste_from_word.XPathFunctions]

      -p tables.set-column-number yes

      -p sections.max-level {pfw:docbookSectionMaxLevel(.)}

      -t paste-from-word:xslt/docbook5.xslt

      -p transform.hierarchy-name
      {if($pasting-root, local-name(/*), pfw:docbookHierarchyName(.))}

      -p transform.cals-tables yes"/>
  <separator />
</menu>
```

Command `pasteFromWord` can be executed when the document opened in XMLmind XML Editor (XXE for short) is editable and when the clipboard contains HTML. The fact that the HTML is non-filtered HTML generated by MS-Word 2003+ is checked later, when the command is actually executed.

1.1.1. Engine options

During its execution, command `pasteFromWord` runs *the "Paste from Word" engine*. This engine converts non-filtered HTML generated by MS-Word 2003+ to a valid document having the same type as the document being edited in XXE (e.g. it generates a DocBook 5 `chapter`). Then `pasteFromWord` pastes part or all of the document generated by the engine at the “right” location in the document being edited in XXE, given the current state of the selection. More information in Section 1.2, “The "Paste from Word" engine” [3].

Therefore the parameter of command `pasteFromWord` merely consists in options [10] directly supported by the "Paste from Word" engine. Example taken from *addon_install_dir*`/docbook5/docbook5.incl`:

```
-p tables.set-column-number yes
```

However in order to be able to give these options values adapted to the document being edited in XXE, the option values are often XPath 1.0 expressions (enclosed in curly braces "`{ }`"). These expressions are evaluated in the context of the implicitly or explicitly selected element of the document being edited. Example taken from *addon_install_dir*`/docbook5/docbook5.incl`:

```
[xmlns:pfw=java:com.xmlmind.xmleditext.paste_from_word.XPathFunctions]1
...
-p transform.hierarchy-name2
   {if($pasting-root, local-name(/*), pfw:docbookHierarchyName(.))}3
```

1 This namespace prefix declaration allows XPath to invoke static methods found in Java™ class `com.xmlmind.xmleditext.paste_from_word.XPathFunctions`. More information in Section 2, “Java™ methods as extension functions” in *XMLmind XML Editor - Support of XPath 1.0*.

2 *`transform.hierarchy-name`* is a parameter which is passed to XSLT stylesheet `paste-from-word:xslt/docbook5.xslt` under the name *`hierarchy-name`*. This parameter specifies to the XSLT stylesheet

what kind of DocBook 5 document should be generated. Examples: "`book`" (`book` containing `chapters` and `sections`), "`chapter-sect1`" (`chapter` possibly containing `sect1`, `sect2`, `sect3`, etc), "`section`" (`section` possibly containing sub-`sections`), "`sect3`" (`sect3` possibly containing `sect4`, `sect5`).

3 This XPath expression reads: if command `pasteFromWord` is used to replace the root element of the document being edited, then generate the same element as the root element, otherwise lookup the ancestors of the context node ("`.`", which is the implicitly or explicitly selected element of the document being edited) to determine the name of the hierarchy to be generated ("`book`", "`chapter-sect1`", etc).

1.1.2. Other options

The only option which is interpreted by command `pasteFromWord` itself and not by the "Paste from Word" engine is `[after` *`element_qname`*`]`. Excerpts from *`addon_install_dir`*`/docbook5/docbook5.incl`:

```
<item label="Paste from _Word" icon="../common/paste_from_word.png"
    command="pasteFromWord"
    parameter="[xmlns:db=http://docbook.org/ns/docbook]
      [after db:para]
  ...
```

Excerpts from *`addon_install_dir`*`/dita/topic.incl`:

```
<item label="Paste from _Word" icon="../common/paste_from_word.png"
      command="pasteFromWord"
      parameter="[after p]
  ...
```

DocBook element `para` and DITA topic element `p` are both plain paragraphs. However these elements may contain blocks such as lists and tables. Inserting a list or table in a plain paragraph makes the structure of the document hard to understand. That's why an option like `[after db:para]` instructs command `pasteFromWord` to paste blocks *after* `db:para`, and *not inside* `db:para`, even if this is allowed by the schema of the document.

1.2. The "Paste from Word" engine

The "Paste from Word" engine converts the non-filtered HTML generated by MS-Word 2003+ to XML. This engine is embedded in command `pasteFromWord`. It is also available as a command-line utility [16].

The conversion to XML comprises 3 phases:

1. **Parse phase:** parse the non-filtered HTML generated by MS-Word 2003+ and convert it to well-formed XHTML (embedding a CSS stylesheet by the means of the `style` XHTML element). The XHTML document obtained this way is completely invalid (e.g. it contain many foreign elements and attributes), highly redundant and non-structured (e.g. no lists; just styled paragraphs).

2. **Edit phase:** modify the XHTML document in place in order to clean it up and to structure it. The XHTML document obtained after this phase is a clean, almost completely structured, valid "XHTML 1.0 Transitional" document.

   This phase is implemented by a elaborate sequence of *edit steps*. Most edit steps are implemented using XED scripts in *XMLmind XML Editor - Support of XPath 1.0*. However a small number of edit steps are still implemented in Java™.

   The edit steps and their parameters are documented in Section 1.2.1, “Edit steps” [4].

3. **Transform phase:** transform the "XHTML 1.0 Transitional" document to the target document type using XSLT 1.0 stylesheets.

   The XSLT stylesheets and their parameters are documented in Section 1.2.3, “Transform stylesheets” [9].

1.2.1. Edit steps

Note

The documentation found in this section is currently insufficient to be able to parameterize and/or customize the edit steps. For now, you'll have to read the XED source of these steps. All these XED scripts are found in *addon_install_dir*/xed/.

The edit steps are invoked in the following order by the main XED script (*addon_install_dir*/xed/main.xed):

after-parse

XED script *addon_install_dir*/xed/after-parse.xed. Delete some foreign elements (e.g. w:Sdt[@docpart-type="Table of Contents"]).

No parameters.

styles

Compiled step. This step processes head/style elements as well as class and style attributes:

- Remove all the elements found in head, except title.
- Parse the contents of style elements found in head.

  Parsed styles are saved as a <!--styles--> comments for reference by the developer of XED scripts.
- Attributes class and style are moved to urn:x-mlmind:namespace:style, the style namespace.

  The value of a style attribute is parsed and possibly split into several attributes belonging to the style namespace.

  The CSS cascade (inheritance, selectors, computed property value, etc) is applied to these style attributes.

No parameters.

prune

XED script *addon_install_dir*/xed/prune.xed. Delete useless elements (e.g. divs which are only used to style their contents). Replace each non-empty span containing only whitespace and/or non-breaking spaces by a single space character.

No parameters.

lang

Compiled step. This step simplifies the lang attributes found in the document.

Parameter	Value	Default Value	Description
lang.lang	remove | simple | simplify | ISO 639-1 two-letter language code	simplify	remove Remove all lang attributes. simple Move the lang attribute from the body element to the html root element. Remove all other lang attributes. simplify Minimize the number of lang attributes found in the document.

Parameter	Value	Default Value	Description
			A user-specified language code such as `de`, `fr-CA`, etc Remove all `lang` attributes. Set the `lang` attribute of the `html` root element to this user-specified language code. In all cases, remove all `s:mso-*-language` attributes.

`title`

XED script *addon_install_dir*`/xed/title.xed`. Process the title of the document.

No parameters.

`biblio`

XED script *addon_install_dir*`/xed/biblio.xed`. Process bibliography entries.

No parameters.

`index`

XED script *addon_install_dir*`/xed/index.xed`. Process index entries.

No parameters.

`xrefs`

XED script *addon_install_dir*`/xed/xrefs.xed`. Process anchors and links.

No parameters.

`inlines`

XED script *addon_install_dir*`/xed/inlines.xed`. Process styled spans.

Parameter	Value	Default Value	Description
`inlines.generate-big-small`	`no`|`yes`	`yes`	if `yes`, convert `span` to `big` or `small` depending on the font size of the style of the `span`.

`tables`

Compiled step. This step mainly reduces the number of `align`, `valign` and `width` attributes found inside tables.

- `align` attributes are removed from list items and tables found in `td`.
- `align` attributes which are common to all `td`/`p` are moved to the `td`.
- `align` attributes which are common to all `td` belonging to the same column are moved to the corresponding `colgroup`.
- `valign` attributes which are common to all `tr`/`td` are moved to the `tr`.
- `width` attributes are removed from all `td` elements. `Colgroup` elements are are added to specify the width of all columns. A width is specified as a percentage.

Moreover this step:

- Groups in a `tbody` all `tr` found directly in a table.
- Sets the table width to 100%.
- Adds attribute `style="-cell-rotate:`*`NNN`*`;"`, where *`NNN`* is 270 or 90, to all rotated `td` elements.

Parameter	Value	Default Value	Description
tables.set-column-number	no|yes	no	if yes, insert before the first child of each table cell <?column-number *N*?>, where *N* is the column number of the cell. First column is column #1.

captions

XED script *addon_install_dir*/xed/captions.xed. Process table and figure captions.

No parameters.

headings

XED script *addon_install_dir*/xed/headings.xed. Convert paragraphs having an outline level to headings (h1, h2, ..., h6). Simplify headings.

No parameters.

lists

XED script *addon_install_dir*/xed/lists.xed. Convert sequences of paragraphs styled as list items to proper lists.

No parameters.

footnotes

XED script *addon_install_dir*/xed/footnotes.xed. Process footnotes and endnotes.

No parameters.

sections

XED script *addon_install_dir*/xed/sections.xed. Leverage headings (h1, h2, ..., h6) to create sections ( )

Parameter	Value	Default Value	Description
sections.max-level	integer; negative or null means no limit.	-1	Specifies how deeply sections can nest.

ids

XED script *addon_install_dir*/xed/ids.xed. Move id attributes from headings and captions to their parent containers (section, table, figure, etc).

No parameters.

finish

XED script *addon_install_dir*/xed/finish.xed. Delete empty elements. Optionally, set <!DOCTYPE>.

Parameter	Value	Default Value	Description
finish.set-doctype	no|yes	no	if yes, add a "XHTML 1.0 Transitional" <!DOCTYPE> to the document being edited.

`before-save`

Compiled step. This step performs the final clean-up needed before saving the XHTML result document to disk.

- It removes the `<!--styles-->` comments found in the `head`.
- It removes all the "`s:`" and "`g:`" prefixed attributes.
- It removes the "`s:`" and "`g:`" prefixes.
- It removes all foreign elements and attributes created by TagSoup, the HTML parser (their namespace starts with "`urn:x-prefix:`").

Parameter	Value	Default Value	Description
`before-save.allow-flow`	`no`|`yes`	`no`	if `yes`, allow flow elements (e.g. `li`, `td`) to contain text and inline elements (e.g. `b`, `i`) in addition to block elements (e.g. `p`, `pre`, `table`). if `no`, do not allow flow elements to contain text and inline elements. In order to implement this, wrap the text and inline elements into `<p class="role-inline-wrapper">`.

1.2.2. XPath extension functions for use by the edit steps

In the following reference, prefix "`f:`" is bound to namespace "urn:x-mlmind:namespace:function".

`string` f:alias(*`style_name`*)

Returns the reference, English, style name corresponding to specified style name. Uses the alias declarations found in text file *`addon_install_dir`*`/xed/aliases.txt` to determine this. Example: `alias("TitelZchn")` returns `"TitleChar"`. Returns *`style_name`* when the reference style name is not found.

`number` f:cm(*`number`*)

Converts *`number`*, a number expressed in centimeters, to points. Example: `cm(2.54)` returns 72. Returns `NaN` when the conversion fails.

`string` f:color(*`color`*)

Converts *`color`* to its 6 hexadecimal digit, upper-case, representation. Examples: `color("red")` returns `"#FF0000"`, `color("rgb(255,0,0)")` returns `"#FF0000"`. Returns `""` when the conversion fails.

`boolean` f:contains-font-family(*`style`*, *`family`*, ..., *`family`*)

Returns `true()` if specified `font-family` style property contains any of the specified typefaces (case insensitive). Example: `contains-font-family(../@s:font-family, "Times New Roman", "Courier New", "Menlo")` returns `true()`. Returns `false()` when this cannot be determined.

`number` f:content-type(*`node`*?)

Returns a numeric code indicating the type of contents of specified element (or parent element of specified node in case specified node is not an element). Parameter *`node`* defaults to the context element (or the parent element of the context node if the context node is not an element). Returns -1 when the content type cannot be determined.

Code	Description
0	Empty.
1	Whitespace only.

Code	Description
	Important Non-breaking space characters (` `) are considered to be whitespace.
2	Element only.
3	Elements and whitespace.
4	Text other than whitespace (words) but no elements.
6	Words and elements.

number f:em(*number*)

Converts *number*, a number expressed in `em`, to points, using the font size of the styled element containing the context node. Example: `em(1)` returns 10. Returns `NaN` when the conversion fails.

number f:ex(*number*)

Converts *number*, a number expressed in `ex`, to points, using the font size of the styled element containing the context node. Example: `ex(1)` returns 5. Returns `NaN` when the conversion fails.

boolean f:font-family(*family*, *node*?)

Returns `true()` if the styled element containing containing specified node uses specified typeface (case insensitive). Parameter *node* defaults to the styled element containing the context node. Examples: `font-family("Times New Roman")` returns `true()`, `font-family("Times New Roman", .//html:tt)` returns `true()`. Returns `false()` when this cannot be determined.

number f:font-size(*node*?)

Returns the font size, expressed in points, of the styled element containing containing specified node. Parameter *node* defaults to the styled element containing the context node. Examples: `font-size()` returns 10, `font-size(./html:b)` returns 10. Returns `NaN` when the font size cannot be determined.

number f:length(*length*)

Converts *length* to points. Example: `length("1in")` returns 72. Returns `NaN` when the conversion fails.

For some units, function `length()` has to use the font size of the styled element containing the context node in order to perform the conversion. Example: `length("1em")` return 10.

number f:line-height(*node*?)

Returns the line height, expressed in points, of the styled element containing containing specified node. Parameter node defaults to the styled element containing the context node. Examples: `line-height()` returns 12, `line-height(.//html:p)` returns 12. Returns `NaN` when the line height cannot be determined.

number f:in(*number*)

Converts *number*, a number expressed in inches, to points. Example: `in(1)` returns 72. Returns `NaN` when the conversion fails.

number f:mm(*number*)

Converts *number*, a number expressed in millimeters, to points. Example: `mm(25.4)` returns 72. Returns `NaN` when the conversion fails.

boolean f:monospaced-font-family(*node*?)

Returns `true()` if the styled element containing containing specified node uses a monospaced font. Parameter *node* defaults to the styled element containing the context node. Examples: `monospaced-font-family()` returns `false()`, `monospaced-font-family(.//html:b)` returns `false()`. Returns `false()` when this cannot be determined.

number f:parse-list-value(*ol_type*, *pattern*, *value*, *start*)

Parses label *value* (e.g. "`III.D.`") of a list item using the format specified by the combination of *ol_type* ("`1`", "`a`", "`A`", "`i`", "`II`") and *pattern* (e.g. "`%1)`", "`%1.%2.`") Returns the number corresponding to the label. Returns *start* (e.g. 1) when the label cannot be parsed. Examples: `parse-list-value("A", "%1.%2.", "III.D.", 1)` returns 4; `parse-list-value("1", "%1)", "(two)", 0)` returns 0.

number f:percent(*percent*)

Converts *percent* to a number. Example: `percent("10%")` returns 10. Returns `NaN` when the conversion fails.

number f:pc(*number*)

Converts *number*, a number expressed in pica, to points. Example: `pc(10)` returns 120. Returns `NaN` when the conversion fails.

number f:pt(*number*)

Converts *number*, a number expressed in points, to points. Example: `pt(10)` returns 10. Returns `NaN` when the conversion fails.

number f:px(*number*)

Converts *number*, a number expressed in pixels, to points, using a 96DPI resolution. Example: `px(100)` returns 75. Returns `NaN` when the conversion fails.

number f:vertical-align(*node*?)

Returns the vertical align, an offset from the baseline expressed in points, of the styled element containing containing specified node. Parameter *node* defaults to the styled element containing the context node. Examples: `vertical-align()` returns -3, `vertical-align(.//html:p)` returns -3. Returns `NaN` when the vertical align cannot be determined or is a keyword (`sub`, `super`) and not a length or percentage.

1.2.3. Transform stylesheets

Note

The documentation found in this section is currently insufficient to be able to parameterize and/or customize the transform stylesheets. For now, you'll have to read the XSLT 1.0 source of these stylesheets. All these stylesheets are found in *addon_install_dir*`/xslt/`.

addon_install_dir`/xslt/docbook5.xslt`

Converts the "XHTML 1.0 Transitional" document created during phase #2 [3] to a DocBook 5 document.

addon_install_dir`/xslt/docbook.xslt`

Converts the "XHTML 1.0 Transitional" document created during phase #2 [3] to a DocBook 4 document.

addon_install_dir`/xslt/topic.xslt`

Converts the "XHTML 1.0 Transitional" document created during phase #2 [3] to a DITA topic.

addon_install_dir/xslt/xhtml1_1.xslt

Converts the "XHTML 1.0 Transitional" document created during phase #2 [3] to an XHTML 1.1 document.

addon_install_dir/xslt/xhtml5.xslt

Converts the "XHTML 1.0 Transitional" document created during phase #2 [3] to an XHTML 5 document.

addon_install_dir/xslt/xhtml_loose.xslt

Converts the "XHTML 1.0 Transitional" document created during phase #2 [3] to a completely structured "XHTML 1.0 Transitional" document.

addon_install_dir/xslt/xhtml_strict.xslt

Converts the "XHTML 1.0 Transitional" document created during phase #2 [3] to an "XHTML 1.0 Strict" document.

1.2.4. Engine options

Process options:

-p *name value*

Set parameter *name* to *value*.

Parameters starting with "transform." are passed to the XSLT stylesheet, if any, after removing the "transform." prefix. All the other parameters are passed as is to the main .xed script, if any.

-pu *name URL_or_file*

Same as "-p", except that parameter value *URL_or_file* is first converted to an URL.

URL_or_file is an URL or an absolute or relative (to current working directory) filename.

-s *xed_URL_or_file*

Specifies which main .xed script to use to modify the document.

Specify an empty string ("") to suppress the edit phase [3].

Default script: paste-from-word:xed/main.xed.

-t *xslt_URL_or_file*

Specifies which XSLT 1.0 stylesheet to use to transform the document.

Specify an empty string ("") to suppress the transform phase [3].

Process options modifying default script "paste-from-word:xed/main.xed":

-parse

Save XHTML without fully processing it. (Stop processing after edit step "styles".)

-i *step xed_URL_or_file*

Insert script before .xed *step*.

-a *step xed_URL_or_file*

Add script after .xed *step*.

`-r` *`step xed_URL_or_file`*

Replace *`step`* by `.xed` script.

`Step` may be a single step name or a range: "*`first`*..*`last`*" or "..*`last`*" or "*`first`*..".

`-d` *`step`*

Delete *`step`*.

`Step` may be a single step name or a range.

2. Customizing the XML generated by "Paste from Word"

In order to customize the XML generated by "Paste from Word", you first need to create a customization of the XHTML, DocBook or DITA configurations as explained in Chapter 5, *Customizing an existing configuration* in *XMLmind XML Editor - Configuration and Deployment*. In this section, we'll use a customization of the DocBook 5 configuration as an example. All the files comprising this customization are found in folder `samples/custom_docbook5/`.

2.1. Custom engine options

2.1.1. Using property *`configuration_name`*.pasteFromWord.parameter

Command `pasteFromWord` is passed a parameter mainly containing engine options [10]. This command is generally invoked from a menu item. Rather than replace the menu item by another one invoking Command pasteFromWord with a different parameter, suffice to define property *`configuration_name`*.pasteFromWord.parameter containing this different parameter. When property *`configuration_name`*.pasteFromWord.parameter is defined, command `pasteFromWord` ignores its normal parameter and uses the value of this property instead.

For example, let's suppose we want "Paste from Word" to generate HTML tables rather than CALS tables. This is done by suppressing the following two engine options from the original parameter of command `pasteFromWord`:

```
-p tables.set-column-number yes
-p transform.cals-tables yes
```

See original parameter in *`addon_install_dir`*`/docbook5/docbook5.incl`.

Hence your customization file should contain (excerpts from `samples/custom_docbook5/0docbook5.xxe`):

```
<property name="$c.pasteFromWord.parameter">1
  [xmlns:db=http://docbook.org/ns/docbook]
  [after db:para]

  [xmlns:pfw=java:com.xmlmind.xmleditext.paste_from_word.XPathFunctions]

  -p sections.max-level {pfw:docbookSectionMaxLevel(.)}

  -t paste-from-word:xslt/docbook5.xslt

  -p transform.hierarchy-name
  {if($pasting-root, local-name(/*), pfw:docbookHierarchyName(.))}
</property>
```

1 Notice the use of the `$c` variable. This variable is automatically substituted with the name of the configuration containing the configuration element.

2.2. Custom edit steps

2.2.1. Using property *configuration_name*.pasteFromWord.parameter.base

Edit steps are implemented by XED scripts in *XMLmind XML Editor - Support of XPath 1.0*. The `-s` engine option [10] specifies which main XED script is to be run to implement phase #2 [3]. Therefore custom edit steps could be implemented as follows:

1. Create your custom main XED script. Let's call this file `custom_main.xed`. This file is found in the folder containing your DocBook 5 customization.

   ```
   include "paste-from-word:xed/main.xed";

   ...YOUR CUSTOM CODE HERE...
   ```

2. Use property *configuration_name*.pasteFromWord.parameter to run it.

   ```
   <property name="$c.pasteFromWord.parameter">
     ...
     -s custom_main.xed
     ...
   </property>
   ```

3. As is, option `-s custom_main.xed` will *not* work because a relative filename is resolved against the current working directory. For this `-s` option to work, your DocBook 5 customization file must additionally define property *configuration_name*.pasteFromWord.parameter.base as follows:

   ```
   <property name="$c.pasteFromWord.parameter.base" url="true">.</property>
   ```

   When this property is defined, command `pasteFromWord` uses it to resolve any relative URL found in its parameters.

2.2.2. Inserting, replacing and removing edit steps in stock `main.xed`

The procedure explained above has been introduced mainly to explain the use of property *configuration_name*.pasteFromWord.parameter.base. However we don't recommend to use it because firstly, there is a more convenient way to customize stock `main.xed` and secondly, you'll rarely want to perform your custom editing *after* stock `main.xed` finishes its work.

Script *addon_install_dir*`/xed/main.xed` looks like this:

```
script(defined[1]("before.after-parse", ""));[2]
script(defined("do.after-parse", "after-parse.xed"));[3]
script(defined("after.after-parse", ""));[4]

script(defined("before.styles", ""));
invoke(defined("do.styles",
               "com.xmlmind.xmleditext.paste_from_word.engine.SetStyles"));[5]
script(defined("after.styles", ""));
...
```

1. XPath extension `defined()` is documented here object defined(string *variable-name*, *default-value*?) in *XMLmind XML Editor - Support of XPath 1.0*.
2. Parameter *before.after-parse* defaults to `""`. Therefore, by default, there is no script which is run before `after-parse.xed`.

   This also means that if you want to run a custom script before step `after-parse`, then pass option "`-pu before.after-parse my_script.xed`" to the engine.
3. If you want to suppress step `after-parse`, pass option "`-p do.after-parse ''`" to the engine.

   If you want to replace step `after-parse`, pass option "`-pu do.after-parse my_script.xed`" to the engine.

4 Parameter *after.after-parse* defaults to "". Therefore, by default, there is no script which is run after after-parse.xed.

This also means that if you want to run a custom script after step after-parse, then pass option "-pu after.after-parse my_script.xed" to the engine.

5 If you want to suppress *compiled step* styles, pass option "-p do.styles ''" to the engine.

There is no direct way to replace a compiled step. You must first suppress it and then specify the corresponding *after.step_name* parameter.

Convenience engine options [10] -i, -a, -d, and -r makes what explained above even easier to use. For example, "-i after-parse my_script.xed" is equivalent to "-pu before.after-parse my_script.xed".

2.2.3. A real world example

The problem to be solved is converting paragraphs styled using MS-Word user-defined style ProgramListing to an XHTML pre element.

Style ProgramListing has the following characteristics:

- Font: "Times New Roman", 10pt.
- Line height: 10pt.
- Space after: 10pt, but do not add space between contiguous paragraphs having a ProgramListing style.

Pasting a few lines of indented source code into a paragraph having a ProgramListing style causes MS-Word to create several contiguous ProgramListing paragraphs, one for each line of source code. Moreover, when saving the document to non-filtered HTML, MS-Word replaces the leading space characters (that is, the indentation of the source code) by non-breaking space characters ( ).

In order to solve this problem, it is recommended to proceed as follows:

1. Use MS-Word to save a sample document making use of your user-defined styles as non-filtered HTML. Example: samples/custom_docbook5/program_listing.docx saved as samples/custom_docbook5/program_listing.htm.

2. Browse the non-filtered HTML file to see what did MS-Word with your user-defined styles.

3. Write your XED script. Example: samples/custom_docbook5/program_listing.xed.

4. Test it using the **toxml** command-line utility [16], and not in XMLmind XML Editor. Example:

```
C:\...\custom_docbook5> toxml -i prune program_listing.xed¬
 program_listing.htm program_listing.xml
```

5. After your XED script passes all tests, integrate it into your DocBook 5 customization. Example: samples/custom_docbook5/0docbook5.xxe.

XED script samples/custom_docbook5/program_listing.xed contains:

```
for-each /html/body//p[starts-with(@s:class, "ProgramListing")] {
    set-element-name("span");
    set-attribute("class", "programlisting");

    set-attribute("g:id", "pre");
    set-attribute("g:container", "pre class='programlisting'");
}

group();

for-each /html/body//span[@class='programlisting'] {
     (: Get rid of inner spans :)
     for-each .//span {
         unwrap-element();
```

```
    }

    (: Get rid of this span by replacing it by a text node
       where nbsp, \n, \r characters have been processed. :)
    replace(<g:envelope>{translate(normalize-space(.),
                           " &#xA;&#xD;", "  ")}&#xA;</g:envelope>);
}
```

It is inserted *before* the `prune` edit step because this step simplifies `span` elements only containing whitespace and/or non-breaking space characters.

When the above **toxml** command is applied to `samples/custom_docbook5/program_listing.htm`, it gives DocBook 5 document `samples/custom_docbook5/program_listing.xml`.

2.3. Custom transform stylesheets

The transform phase [3] is implemented by the means of XSLT 1.0 stylesheets. The `-t` engine option [10] specifies which XSLT stylesheet is to be used to implement the transform phase. Therefore a custom transform phase is implemented as follows:

1. Create your XSLT 1.0 stylesheet. Let's call this file `custom_docbook5.xslt`. This file is found in the folder containing your DocBook 5 customization.

   ```
   <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
                   xmlns:h="http://www.w3.org/1999/xhtml"
                   xmlns="http://docbook.org/ns/docbook"
                   exclude-result-prefixes="h">

     <xsl:import href="paste-from-word:xslt/docbook5.xslt"/>

   ...YOUR CUSTOMIZATION HERE...
   ```

2. Use property *`configuration_name`*.pasteFromWord.parameter to run it.

   ```
   <property name="$c.pasteFromWord.parameter">
     ...
     -t custom_docbook5.xslt
     ...
   </property>
   ```

3. As is, option `-t custom_docbook5.xslt` will *not* work because a relative filename is resolved against the current working directory. For this `-t` option to work, your DocBook 5 customization file must additionally define property *`configuration_name`*.pasteFromWord.parameter.base as follows:

   ```
   <property name="$c.pasteFromWord.parameter.base" url="true">.</property>
   ```

   When this property is defined, command `pasteFromWord` uses it to resolve any relative URL found in its parameters.

2.3.1. A real world example

The problem to be solved is transforming XHTML `pre` elements having attribute `class="programlisting"` to DocBook 5 `programlisting` elements. (The stock XSLT stylesheet *`addon_install_dir`*`/xslt/docbook5.xslt` generates `literallayout` elements.)

In order to solve this problem, it is recommended to proceed as follows:

1. Use MS-Word to save a sample document making use of your user-defined styles as non-filtered HTML. Example: `samples/custom_docbook5/program_listing.docx` saved as `samples/custom_docbook5/program_listing.htm`.

2. Write your XSLT stylesheet. Example: `samples/custom_docbook5/custom_docbook5.xslt`.

3. Test it using the **toxml** command-line utility [16], and not in XMLmind XML Editor. Example:

```
C:\...\custom_docbook5> toxml -i prune program_listing.xed¬
 -t custom_docbook5.xslt program_listing.htm program_listing.xml
```

   In the above **toxml** command, option "`-i prune program_listing.xed`" is used to generate XHTML `pre` elements having attribute `class="programlisting". See Section 2.2, “Custom edit steps” [12].`

4. After your XSLT stylesheet passes all tests, integrate it into your DocBook 5 customization. Example: `samples/custom_docbook5/0docbook5.xxe`.

XSLT stylesheet `samples/custom_docbook5/custom_docbook5.xslt` contains:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
                xmlns:h="http://www.w3.org/1999/xhtml"
                xmlns="http://docbook.org/ns/docbook"
                exclude-result-prefixes="h">

  <xsl:import href="paste-from-word:xslt/docbook5.xslt"/>

  <xsl:template match="h:pre[@class='programlisting']">
    <programlisting>
      <xsl:call-template name="processCommonAttributes"/>
      <xsl:apply-templates/>
    </programlisting>
  </xsl:template>

</xsl:stylesheet>
```

When the above **toxml** command is applied to `samples/custom_docbook5/program_listing.htm`, this gives DocBook 5 document `samples/custom_docbook5/program_listing.xml`.

3. Integrating "Paste from Word" into configurations other than XHTML, DocBook and DITA

We'll explain how to integrate "Paste from Word" into a configuration by using an example. The configuration we'll modify is called "`Simple Section (XML Schema)`". It is found in `xsd_section_config/`. This sample configuration is described in Section 2, “A configuration for the "Simple Section" document type.” in *XMLmind XML Editor - Configuration and Deployment*.

Procedure:

1. Add a menu item invoking command `pasteFromWord`.

   Excerpts from `samples/xsd_section_config/pfw/pfw.incl`:

```
<menu label="-" insert="##first">
  <item label="Paste from _Word"
        icon="paste-from-word:common/paste_from_word.png"
        command="pasteFromWord"
        parameter="-t section-config:pfw/pfw.xslt"/>
  <separator />
</menu>
```

2. Notice in the above configuration file the parameter of command `pasteFromWord` which specifies that the "XHTML 1.0 Transitional" document created during phase #2 [3] is to be converted to a "Simple Section" using XSLT 1.0 stylesheet "`section-config:pfw/pfw.xslt`"[1].

[1] Location "`section-config:pfw/pfw.xslt`" can be successfully converted to an URL because `xsd_section_config/` contains XML catalog `catalog.xml`:

```
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <rewriteURI uriStartString="section-config:" rewritePrefix="." />
  ...
```

As you can see, most of the integration effort consists in developing XSLT 1.0 stylesheet "`section-config:pfw/pfw.xslt`".

There are few things to say about how to develop this XSLT 1.0 stylesheet:

- Run **toxml** [16] with a `-t ""` option to learn about the XHTML input of the XSLT stylesheet:

```
C:\…\paste_from_word> toxml -t "" test1.htm test1.xhtml
```

- Read the source of other similar XSLT stylesheets. For example: `paste-from-word:xslt/docbook5.xslt`, `paste-from-word:xslt/topic.xslt`, etc.

- Notice that the XHTML body element is always processed as follows:

```
<xsl:template match="h:body">
  <xsl:processing-instruction name="pfw-begin-body"/>

  ...

  <xsl:processing-instruction name="pfw-end-body"/>
</xsl:template>
```

  This is needed for the following reasons:

  - When the user selects menu item "Paste from Word" to replace the root element of her/his document, suffice for command `pasteFromWord` to paste the root element of the document generated by the XSLT stylesheet.

  - In any other case, command `pasteFromWord` pastes all the nodes found between `<?pfw-begin-body?>` and `<?pfw-end-body?>`.

A. The toxml command-line utility

The **toxml** command-line utility is the "Paste from Word" engine [3] in the form a command-line utility. It allows to convert the non-filtered HTML generated by MS-Word 2003+ to XML.

The **toxml** command-line utility is available as `toxml` (shell script; Mac OS X, Linux) and as `toxml.bat` (Windows). Both files are found in the directory where the "Paste from Word" add-on has been installed.

Note

Depending on where the "Paste from Word" add-on has been installed, you may have to edit `toxml` or `toxml.bat` using a text editor in order to modify variable `xxeHome`.

Command-line usage:

toxml [-v|-vv] [Process options] [Format options] *in_html_file out_xml_file*

Process options:

`-p` *name value*

Set parameter *name* to *value*.

Parameters starting with "`transform.`" are passed to the XSLT stylesheet, if any, after removing the "`transform.`" prefix. All the other parameters are passed as is to the main `.xed` script, if any.

`-pu` *`name URL_or_file`*

Same as "`-p`", except that parameter value *`URL_or_file`* is first converted to an URL.

`URL_or_file` is an URL or an absolute or relative (to current working directory) filename.

`-s` *`xed_URL_or_file`*

Specifies which main `.xed` script to use to modify the document.

Specify an empty string (`""`) to suppress the edit phase [3].

Default script: `paste-from-word:xed/main.xed`.

`-t` *`xslt_URL_or_file`*

Specifies which XSLT 1.0 stylesheet to use to transform the document.

Specify an empty string (`""`) to suppress the transform phase [3].

Process options modifying default script "`paste-from-word:xed/main.xed`":

`-parse`

Save XHTML without fully processing it. (Stop processing after edit step "`styles`".)

`-i` *`step xed_URL_or_file`*

Insert script before `.xed` *`step`*.

`-a` *`step xed_URL_or_file`*

Add script after `.xed` *`step`*.

`-r` *`step xed_URL_or_file`*

Replace *`step`* by `.xed` script.

`Step` may be a single step name or a range: "*`first`*..*`last`*" or "..*`last`*" or "*`first`*..".

`-d` *`step`*

Delete *`step`*.

`Step` may be a single step name or a range.

The steps of default script are: `after-parse`, `styles`, `prune`, `lang`, `title`, `biblio`, `index`, `xrefs`, `inlines`, `tables`, `captions`, `headings`, `lists`, `footnotes`, `sections`, `ids`, `finish`, `before-save`.

Format options:

`-out` *`format`*

Specifies the output format in case it cannot be determined using the extension of the output file.

Formats: `docbook`, `docbook5`, `topic`, `xhtml_strict`, `xhtml_loose`, `xhtml1_1`, `xhtml5`, `xhtml` (default).

Other options:

`-v`, `-vv`

Verbose.