

XMLmind XML Editor - Commands

Hussein Shafie
Pixware

`<xmleditor-support@xmlmind.com>`

XMLmind XML Editor - Commands

Hussein Shafie

Pixware

<xmlmind-support@xmlmind.com>

Published December 2, 2005

Abstract

This documents contains the reference of all native XXE commands and explains how to write custom macro-commands.

Table of Contents

I. Guide	1
1. What is a command?	2
2. Writing macro-commands	4
1. How commands are executed	4
2. A sequence of commands	5
3. Alternative commands	5
4. Testing if a command can be executed	6
5. The "%_" variable	6
6. Macro-variables	7
7. Simple use of named variables	8
8. General use of get	9
9. Variables mapped to the selection in XXE	10
10. Contextual commands	10
II. Reference	12
3. Menu commands	13
1. Reference	13
2. Examples	14
4. Macro commands	15
1. Reference	15
1.1. Pass and fail cheat sheet	18
1.2. Macro-variables	18
1.3. XPath variables	19
1.3.1. User variables	19
1.3.2. Predefined variables	19
2. Examples	22
5. Process commands	25
1. Reference	25
1.1. Attributes	28
1.2. Element <code>copyDocument</code>	28
1.2.1. Attributes	29
1.2.2. Element <code>extract</code>	30
1.2.3. Element <code>resources</code>	32
1.3. Element <code>convertImage</code>	33
1.3.1. Parameters supported by the built-in Java image toolkit	34
1.3.2. Parameters supported by the Batik image toolkit	34
1.3.3. Parameters supported by the Jimi image toolkit	35
1.4. Element <code>copyProcessResources</code>	35
1.5. Element <code>transform</code>	35
1.5.1. Using a custom XSLT style sheet	36
1.6. Element <code>processFO</code>	37
1.7. Element <code>upload</code>	39
1.8. Element <code>post</code>	39
1.9. Element <code>print</code>	41
1.10. Element <code>read</code>	41
1.11. Element <code>mkdir</code>	42
1.12. Element <code>rmdir</code>	42
1.13. Element <code>delete</code>	42
1.14. Element <code>copy</code>	42
1.15. Element <code>zip</code>	43
1.16. Element <code>jar</code>	43
1.17. Element <code>shell</code>	44
1.18. Element <code>invoke</code>	45
1.19. Element <code>subProcess</code>	46
1.20. Process variables	47
2. Commented examples	48

2.1. Convert explicitly or implicitly selected <code>para</code> to a <code>simpara</code>	48
2.2. Convert a DocBook document to RTF	49
2.3. Convert ImageDemo document to HTML	51
3. The convertdoc command line tool	52
3.1. Convert a DocBook document to multi-page HTML	53
3.2. Example 2: Convert a DocBook document to PDF	54
3.3. Example 3: Convert a DocBook document to JavaHelp™	56
6. Commands written in the Java™ programming language	57
1. alert	58
2. add	58
3. addAttribute	58
4. addBlockInFlow	59
5. beep	59
6. cancelSelection	59
7. center	60
8. checkValidity	60
9. confirm	60
10. convert	60
11. convertCase	61
12. copy	61
13. copyAsInclusion	61
14. copyChars	62
15. cut	62
16. decrSectionLevel	62
17. declareNamespace	63
18. delete	63
19. deleteChar	63
20. deleteSelectionOrDeleteChar	64
21. deleteSelectionOrJoinOrDeleteChar	64
22. editAttributes	64
23. editMenu	64
24. editPITarget	64
25. execute	65
26. extractImage	65
27. extractObject	66
28. include	66
29. incrSectionLevel	67
30. insert	67
31. insertCharByName	67
32. insertControlChar	68
33. insertControlCharOrSplit	68
34. insertNode	68
35. insertSpecialChars	69
36. insertString	69
37. join	69
38. joinOrDeleteChar	69
39. listBindings	70
40. makeParagraphs	70
41. moveDotTo	71
42. paste	71
43. pasteInclusionEnabled	72
44. pasteSystemSelection	72
45. pick	73
46. prompt	73
47. putAttribute	73
48. recordMacro	74
49. redo	75
50. refresh	75

51. removeAttribute	75
52. repeat	76
53. replace	76
54. resizeSVG	76
55. run	76
56. search	77
57. searchReplace	78
58. selectAt	79
59. selectBlockAtY	79
60. selectById	79
61. selectFile	80
62. selectConvertedFile	80
63. selectPrinter	81
64. selectNode	81
64.1. List of element names or node types	83
64.2. OrNone, OrNode, OrElement modifiers	84
65. selectNodeAt	84
66. selectText	85
67. selectTo	85
68. setEditable	85
69. setImage	86
70. setObject	87
71. showContentModel	87
72. showMatchingChar	87
73. spellCheck	87
74. split	88
75. start	88
76. status	88
77. toggleCollapsed	88
78. undo	89
79. viewHTML	89
80. wrap	90
81. xpathSearch	90
82. XXE.edit	91
83. XXE.open	91
84. XXE.openAsTemplate	92
85. XXE.openCopy	92
86. XXE.close	93
87. XXE.save	93
88. XXE.saveAs	93
89. XXE.saveCopy	94
7. XPath functions	95
1. Extension functions	95
2. Java™ methods as extension functions	97

Part I. Guide

Chapter 1. What is a command?

A command is an action which occurs in the view of a document, styled or not. This action is triggered by a key-stroke, mouse click, custom tool bar button (example: the XHTML tool bar) or custom menu entry (example: the DocBook menu).

Some menu entries of XMLmind XML Editor such as File|Open have been made available as commands. For example, the command corresponding to menu entry File|Open is called `XXE.open`. But other menu entries such as File|Print are not (yet) available as commands. For example, you cannot invoke File|Print from a custom tool bar and you cannot invoke File|Print from a macro-command.

Almost all commands can be passed a *parameter string* which is used to parametrize the behavior of the command. The syntax of this parameter string and its exact effects are totally command specific. Therefore there is nothing more to say about these parameter strings except that you'll need to read the reference manual of all native commands to check what is supported and what is not.

There are four types of commands:

Commands written in the Java programming language

All generic commands written in the Java™ programming language are predefined: you don't need to declare them.

All XML application specific commands written in the Java™ programming language (XMLmind XML Editor - Developer's Guide describes how to write such command) need to be declared in an XXE configuration file (see XMLmind XML Editor - Configuration and Deployment). Example:

```
<command name="xhtml.preview">
  <class>com.xmlmind.xmlmleditapp.xhtml.Preview</class>
</command>
```

Menu commands

A "menu command" is a popup menu of commands. This special type of command, typically invoked from contextual macro-commands, is intended to be used to specify contextual popup menus, redefining or extending the standard right-click popup menu.

Menu of commands need to be specified in an need to be declared in an XXE configuration file (see XMLmind XML Editor - Configuration and Deployment). Example:

```
<command name="contextualMenu">
  <menu>
    <item label="To Upper Case"
      command="convertCase" parameter="upper" />
    <item label="To Lower Case"
      command="convertCase" parameter="lower" />
    <item label="Capitalize Words"
      command="convertCase" parameter="capital" />

    <editMenu />
  </menu>
</command>
```

Macro-commands

A macro-command is, to make it simple, a sequence of native commands, menu commands, process commands or other macro-commands.

Macro-commands need to be specified in an XXE configuration file (see XMLmind XML Editor - Configuration and Deployment). Example:

```
<command name="xhtml.convertToLink">
  <macro>
    <sequence>
      <command name="convert" parameter="a" />
    </sequence>
  </macro>
</command>
```

```
<command name="putAttribute" parameter="%*" />
  </sequence>
</macro>
</command>
```

Process commands

A process command is an arbitrarily complex transformation of part or all of the document being edited.

Process commands need to be specified in an XXE configuration file (see XMLmind XML Editor - Configuration and Deployment). Example:

```
<command name="toSimpara">
  <process>
    <copyDocument selection="true" to="in.xml" />
    <transform stylesheet="simpara.xslt" cacheStylesheet="true"
      file="in.xml" to="out.xml" />
    <read file="out.xml" encoding="UTF-8" />
  </process>
</command>
```

Chapter 2. Writing macro-commands

The macro-command examples you'll find in this tutorial can be tested by creating a file called `customize.xxe` in `XXE_user_preferences_dir/addon/` (`XXE_user_preferences_dir` is `$HOME/.xxe/` on Unix and `%System-Drive%\Documents and Settings\%USERNAME%\Application Data\XMLmind\XMLeditor\` on Windows) and binding the command to be tested to a keystroke.

Example: this `customize.xxe` file binds a macro-command named `convertToBold` to keystroke **F2**.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<configuration
  xmlns="http://www.xmlmind.com/xmleditor/schema/configuration"
  xmlns:cfg="http://www.xmlmind.com/xmleditor/schema/configuration">

  <binding>
    <keyPressed code="F2" />
    <command name="convertToBold" />
  </binding>

  <command name="convertToBold">
    <macro>
      <sequence>
        <command name="convert"
          parameter="[implicitElement] emphasis" />
        <command name="putAttribute" parameter="role bold" />
      </sequence>
    </macro>
  </command>
</configuration>
```

More info. about customizing XMLmind XML Editor in XMLmind XML Editor - Configuration and Deployment.

The examples used in this tutorial are found in `XXE_install_dir/doc/commands/tutorial/customize.xxe`.

1. How commands are executed

Before attempting to write a macro-command, it is important to understand how commands (native or not) are executed.

The execution of a command can be described as a sequence of two steps:

1. The command tests if it can be successfully executed. If this test fails, the command will not attempt to execute itself: step #2 is silently skipped (that is, no warning or error messages are reported).

For this test to pass:

- a. The command must successfully parse its parameter string (if any).
 - b. The current text or node selection (if any) must be compatible with the command. For example, command `replace` cannot be applied to text selection.
 - c. The grammar constraining the document (if any) must allow the operation.
2. The command is actually executed. It may return a result (a Java object) to its invocation environment.

Very few commands return a result. Unless explicitly documented, one must assume that commands do not return a result.

Some commands such as `selectFile` return an actual result (a file name or an URL in the case of command `selectFile`) or a special result understood by the invocation environment as ``command has failed or has been

canceled by user" (in the case of command `selectFile`, user has clicked on the Cancel button of the file chooser dialog box).

2. A sequence of commands

DocBook has no `bold` tag" but it is customary to use the `emphasis` element with attribute `role` equals to `bold`. The following macro automates this:

```
<command name="convertToBold">
  <macro>
    <sequence>
      <command name="convert"
        parameter="[implicitElement] emphasis" />
      <command name="putAttribute" parameter="role bold" />
    </sequence>
  </macro>
</command>
```

Using a sequence element:

1. The macro converts anything convertible to an `emphasis` element (generally text selection, but not only text selection) to an `emphasis` element.
2. If step #1 is successful, the macro adds attribute `role` with value `bold` to the newly created `emphasis` element.

Note that if the first step of a sequence cannot be executed (this is tested before attempting to actually execute the sequence construct), the whole sequence construct cannot be executed.

Step #2 works without an `"[implicitElement]"` parameter for command `putAttribute` because the newly created `emphasis` element has been automatically selected by the `convert` command of step #1.

This is often the case. A quick way to learn this is to first perform interactively what needs to be automated by the macro.

3. Alternative commands

The above macro needs to be refined. If an `emphasis` element is implicitly or explicitly selected and this element has no `role` attribute or a `role` different from `bold`, we would like to add to it attribute `role` with value `bold`.

The following macro uses a choice element to do this:

```
<command name="convertToBold2">
  <macro>
    <sequence>
      <choice>
        <command name="selectNode"
          parameter="self[implicitElement] emphasis" />
        <command name="convert"
          parameter="[implicitElement] emphasis" />
      </choice>

      <command name="putAttribute" parameter="role bold" />
    </sequence>
  </macro>
</command>
```

The choice element will execute the first alternative which can be executed:

- Explicitly selecting (using command `selectNode`) the `emphasis` element if such element is implicitly (or explicitly) selected;
- OR converting anything else to an `emphasis` element, using command `convert`.

If all alternatives cannot be executed (this is tested before attempting to actually execute the choice construct), the whole choice construct cannot be executed.

4. Testing if a command can be executed

The following macro inserts a `br`, an XHTML line break element, at caret position. If there is no text node after the newly inserted `br`, the macro inserts a new text node in order to let the user continue to type text. This macro is typically bound to a keystroke such as **Shift-Enter**.

The problem is that we don't want to insert a new text node after a `br` inserted in text span elements such as `b`, `i`, `em`, `strong`, `a`, etc, but only after a `br` inserted in a text block such as `p`, `li`, etc.

Command `selectNode` can, not only blindly select nodes, but it can also select nodes conditionally, if these nodes match a list of elements passed as a parameter string.

Therefore the idea is to use `selectNode`, not for its ability to select nodes, but for its ability to test where the caret is.

The `pass` and `fail` constructs have designed to do this: test if a command can be executed without actually executing it.

```
<command name="insertLineBreak">
  <macro>
    <sequence>
      <command name="insert" parameter="into br" />

      <sequence>
        <pass><command name="selectNode"
          parameter="parent p li dt dd th td"/></pass>
        <command name="insertNode" parameter="textAfter" />
        <command name="cancelSelection" />
      </sequence>
    </sequence>
  </macro>
</command>
```

If `selectNode` can be executed, then the `pass` construct can be executed, then the whole sequence can be executed.

Note that when the `pass` construct is actually executed, *it does nothing at all*. This is good because, in our example, if `selectNode` was actually executed, it would have selected, say a `p` or a `li`, after which you generally cannot add a text node (moreover adding a text node after the `p` or `li` is *not* what we want to do).

The last step of the sequence, `cancelSelection`, is just a refinement which removes the "red border" around the newly inserted text node.

5. The "%_" variable

Few commands return a result to their invocation environment (here the invocation environment is the macro).

Command `run` is one of the few commands really designed to return a value: it executes an external command, for example `dir` on Windows (`ls` on Unix), and it captures what is printed on the console to return it as its result.

The following macro is used to run an external command (user is prompted to specify it) and then, to insert at caret position the text which is the result of the external command.

```
<command name="insertCommandOutput">
  <macro>
    <sequence>
      <command name="run" />
      <command name="insertString" parameter="%_" />
    </sequence>
  </macro>
</command>
```

Command `insertString` can insert text at caret position. But how to pass to command `insertString` what has been returned by command `run`? The answer is: use variable `"%_"`.

Each time a command (or a sequence, or a choice) is executed inside a macro, the result of the executed command (or construct) is used to assign a predefined variable which referenced as `"%_"` in command parameters.

When executed command does not return a result, variable `"%_"` is cleared. A reference in a command parameter to a cleared `"%_"` is replaced by the empty string.

The sequence and choice constructs, which can be considered as being pseudo-commands, can return results too:

sequence

Returns the result of its last step.

choice

Returns the result of its executed alternative.

The pass and fail constructs are just tests. They have no effect on `"%_"`. That is, they return the result of the last executed command or construct.

6. Macro-variables

Variables which are referenced as `"%variable_name"` are *macro-variables*. They are referenced in the parameter of commands. They are substituted with their values before the command (or construct) is tested for execution and before the command (or construct) is actually executed.

We have already studied the `"%_"` variable. There are other macro-variables: `"%0"`, `"%1"`, `"%*"`, `"%d"`, etc.

Note that all macro-variables are predefined, which means that there is no way for a user to define its own macro-variables in its custom macros.

The following macro pastes after explicitly or implicitly element, the content of the clipboard after parsing this content as paragraphs. For example, if the clipboard contains several lines of text, each line can be converted to a paragraph. Such macro is useful to convert legacy documents to XML documents.

This macro is built using a sequence of commands `makeParagraphs` and `paste`.

```
<command name="insertAfterAsParagraphs">
  <macro>
    <sequence>
      <command name="makeParagraphs" parameter="%0" />
      <command name="paste" parameter="after[implicitElement] %_" />
    </sequence>
  </macro>
</command>

<binding>
  <keyPressed code="F3" />
  <command name="insertAfterAsParagraphs" parameter="para" />
</binding>
```

`MakeParagraph` has a mandatory parameter string which must be used to specify which paragraph element to create: is it XHTML `p`? Is it DocBook `para`? Is it DocBook `simpara`? Etc.

Macro `insertAfterAsParagraphs` has been made as generic as command `makeParagraphs` because it must be passed a parameter string specifying which paragraph elements to create. This question is simply: how to reference the parameter string passed to a macro inside this macro? The answer is: use following macro-variables:

`%*`

is the value of the whole parameter string.

%0, %1, %2, ..., %9

are parts of the parameter string, split like what is done for command line arguments. For example, if parameter string is:

```
foo 'bar is a gee' "gee is a wiz"
```

%0 is "foo", %1 is "bar is a gee", %2 is "gee is a wiz" and %3, ..., %9 are substituted with the empty string.

7. Simple use of named variables

The following macro is used to insert a DocBook `ulink` element at caret position, the URL referenced by the inserted `ulink` being chosen from a predefined list.

Command `pick` has been created to display a dialog box which lets the user choose one item from a list. This command returns the selected item (a string) to its invocation environment.

Command `pick` can always be executed, but it returns a special value when the user has canceled its execution by clicking on the Cancel button of its dialog box.

```
<command name="insertFamousUlink">
  <macro>
    <sequence>
      <command name="pick"
        parameter="Favorites true
          W3C
          http://www.w3.org/TR/
          'DocBook Oasis'
          http://www.oasis-open.org/docbook/xml/
          Java
          http://java.sun.com/" />
      <set variable="url" expression="%_" plainString="true" />

      <command name="insert" parameter="into ulink" />

      <get expression="$url" />
      <command name="putAttribute" parameter="url %_" />

      <get expression="$url" />
      <command name="insertString" parameter="%_" />
    </sequence>
  </macro>
</command>
```

The above macro stores the result returned by command `pick` in a user variable called `url`. The value of the `url` variable is then used twice: one time to set the value of attribute `url` of element `ulink`, a second time to specify the text of element `ulink`.

Macro variable `"%_"` is extremely volatile. For example, the following sequence cannot be used to add attribute `url` to newly inserted element `ulink`, because command `insert`, which does not return a result, clears `"%_"`.

```
<sequence>
  <command name="pick"
    parameter="Favorites true
      W3C
      http://www.w3.org/TR/
      'DocBook Oasis'
      http://www.oasis-open.org/docbook/xml/
      Java
      http://java.sun.com/" />

  <command name="insert" parameter="into ulink" />

  <command name="putAttribute" parameter="url %_" />
</sequence>
```

The only easy way to reuse what has been returned by command `pick` is to immediately save the value of `"%_"` in a user-defined variable.

User-defined variables are not related to macro-variables. They are set using special construct `set` and are read using special construct `get`. These constructs have `expression` attributes which have been designed to contain arbitrarily complex XPath expressions (more info. about this in following sections).

The above macro illustrates a trivial use of `set` and `get`. This means that you don't need to learn XPath to use `set` and `get` to do simple things. However, it is important to remember this:

- Do not forget to add `plainString=true` to element `set`. Otherwise, the value of attribute `expression` is understood as being an XPath expression.
- User-defined variables cannot be referenced in constructs other than `set`, `get`, `test` and `match`. For example, it is not possible to directly write:

```
<command name="putAttribute" parameter="url $url" />
<command name="insertString" parameter="$url" />
```

In such case, the `get` construct must be used because it is the only way to return in `"%_"` the value of `"$url"`.

- Do not use the following names for your variables because they have a special meaning (more info. about this in following sections): `implicitElement`, `selectedElement`, `implicitNode`, `selectedNode`, `selectedChars`, `selectedNodes`, `selected`, `selected2`, `dot`, `dotOffset`, `mark`, `markOffset`.

8. General use of `get`

At this point of the tutorial, you'll need to know the XPath standard to understand what follows.

The following macro is used to display in an external image viewer, the image referenced in the `fileref` attribute of explicitly or implicitly selected DocBook elements `graphic` or `imagedata`.

The image viewer used by this macro is an external program called `xv`. It is launched using command `start`.

```
<command name="startImageViewer">
  <macro>
    <sequence>
      <get context="$implicitElement/@fileref"
          expression="uri-to-file-name(resolve-uri(.))" />
      <command name="start" parameter='xv "%_"' />
    </sequence>
  </macro>
</command>
```

The above macro shows how to use `get` at its best:

1. First the `context` attribute, common to all XPath-based constructs `get`, `set`, `test` and `match`, is evaluated as a node set, using the document as a context node.
2. The `expression` attribute, common to `get`, `set` and `test`, is evaluated as a string using the context node found in previous step.

If `get`, `set`, `test` or `match` have no `context` attribute, the context node used to evaluate `expression` is the document node itself (that is, XPath `"/`).

The `context` attribute contains `"$implicitElement/@fileref"` which means attribute `fileref` of explicitly or implicitly selected element, because `implicitElement` is a predefined variable mapped to explicitly or implicitly selected element (more info. about this in next section).

The `expression` attribute contains `"uri-to-file-name(resolve-uri(.))"`. `"."` is the `fileref` attribute node. `resolve-uri()` and `uri-to-file-name()` are two non-standard XPath functions which are used to resolve a relative URL and then to convert this URL to a file name (`xv` will not work if passed an URL).

9. Variables mapped to the selection in XXE

The following macro can be used to move a DocBook list item (`listitem`, `callout` or `step`) down in the list. How to move a DocBook list item up in the list can be found in "Using the XPath-based constructs `match` and `set`".

```
<command name="moveListItemDown">
  <macro>
    <sequence>
      <command name="selectNode"
        parameter="ancestorOrSelf[implicitElement] listitem callout step" />1
      <match context="$selected" pattern="*[position() < last()]" />2
      <set variable="anchor" context="$selected"
        expression="./following-sibling:*[1]" />3
      <command name="cut" />4
      <set variable="selected" expression="$anchor" />5
      <command name="paste" parameter="after" />6
    </sequence>
  </macro>
</command>
```

- 1** This step ensures that the macro can be executed only inside a list item.
- 2** This step ensures that the macro cannot be executed for last list item.

It uses the XPath-based construct `match`. As a pseudo-command of a macro, it can be executed only if the context node specified in its `context` attribute matches the XSLT pattern specified in its `pattern` attribute.

This construct like `pass`, `fail` and `test`, is only a test. When `match` is actually executed, it does nothing at all.

- 3** This step saves in user variable named `anchor`, the list item which follows the selected list item.

Variable `selected` referenced from the `context` attribute of this `set` construct is, like `implicitElement` seen in previous example, one of the many predefined variables mapped to the selection in XXE:

- Reading variable `selected` returns the node, first selected in the node selection, whatever is its type.
- Writing variable `selected` clears current node selection or current text selection if any, and then, explicitly selects specified value (which must be a node set).

- 4** Cut selected list item.
- 5** Select the list item saved in variable `anchor`: ``the following one".

The selection is changed by assigning a node value to predefined variable `selected`, as explained above.

- 6** Paste the list item found in the clipboard after last selected list item.

10. Contextual commands

The following macro swaps the character before the caret with the character after the caret. It is useful if, like everybody, you are a bit dyslexic.

```
<command name="transposeChars">
  <macro>
    <sequence>
      <test expression="not($selected) and not($mark) and
        $dotOffset > 0 and
        $dotOffset < string-length($dot)"/>
      <command name="selectTo" parameter="previousChar" />
      <command name="cut" />
      <command name="moveDotTo" parameter="nextChar" />
      <command name="paste" parameter="into" />
    </sequence>
  </macro>
</command>
```

The above macro uses basic commands `selectTo`, `moveDotTo`, `cut` and `paste`, but also XPath-based construct `test`.

As a pseudo-command of a macro, `test` can be executed only if its `expression` attribute evaluated as a boolean in the context specified by its `context` attribute returns true.

This construct like `pass`, `fail` and `match`, is only a test. When `test` is actually executed, it does nothing at all.

`Test` is used in the above macro to ensure that the macro can be executed only if:

- There is no node selection: `not($selected)`.
- There is no text selection: `not($mark)`.
- The caret is not before first character of a textual node: `$dotOffset > 0`.
- The caret is not after last character of a textual node: `$dotOffset < string-length($dot)`.

Like `selected` and `implicitElement` seen in previous examples, `mark`, `dot` and `dotOffset` are predefined variables mapped to the selection in `XXE`.

Part II. Reference

Chapter 3. Menu commands

1. Reference

```
<command
  name = NMTOKEN
>
  Content: class | menu | macro | process
</command>

<menu
  label = non empty token
>
  Content: editMenu? [ menu | separator | item ]+ editMenu?
</menu>

<separator
/>

<item
  label = non empty token
  command = NMTOKEN
  parameter = string
/>

<editMenu
/>
```

Define a popup menu of commands. This special type of command, typically invoked from contextual macro-commands, is intended to be used to specify contextual popup menus, redefining or extending the standard "right-click" popup menu.

The `editMenu` element can be used to specify contextual menus that extend the standard "right-click" popup menu. By the way, the standard "right-click" popup menu is a predefined menu command called `editMenu`.

This element must only be added once: before all top level items or after all top level items.

A separator is automatically added before or after `editMenu`. Therefore there is no need to specify a `separator` element in this case.

Note that the `binding` configuration element (see XMLmind XML Editor - Configuration and Deployment) can also contain a popup menu child element. But menu commands are more powerful because:

- They can be used to extend the standard "right-click" popup menu.
- They can be bound to a keystroke (popup menus in `binding` can only be bound to a mouse click).

2. Examples

Example 3.1. Contextual menus

The example below is pretty useless however it shows how contextual menus can be implemented. It displays a special menu when text is selected and another special menu when root element is selected.

This example can be found in `XXE_install_dir/doc/configure/samples2/contextualmenus.xxe`.

```
<command name="contextualMenu">
  <macro>
    <choice>
      <sequence>
        <test expression="$selectedChars != ''" />
        <command name="contextualMenu1" />
      </sequence>

      <sequence>
        <match context="$selectedElement" pattern="/*" />
        <command name="contextualMenu2" />
      </sequence>

      <command name="editMenu" />
    </choice>
  </macro>
</command>

<command name="contextualMenu1">
  <menu>
    <item label="To Upper Case"
      command="convertCase" parameter="upper" />
    <item label="To Lower Case"
      command="convertCase" parameter="lower" />
    <item label="Capitalize Words"
      command="convertCase" parameter="capital" />

    <editMenu />
  </menu>
</command>

<command name="contextualMenu2">
  <menu label="File">
    <editMenu />

    <item label="Open..." command="XXE.open" />
    <separator/>
    <item label="Save" command="XXE.save" />
    <item label="Save As..." command="XXE.saveAs" />
  </menu>
</command>

<binding>
  <mousePressed button="3"/>
  <command name="contextualMenu" />
</binding>
```

Chapter 4. Macro commands

1. Reference

```
<command
  name = NMTOKEN
>
  Content: class | menu | macro | process
</command>

<macro
  trace = boolean : false
  repeatable = boolean : false
  undoable = boolean : false
  label = non empty token
>
  Content: choice | sequence
</macro>

<choice>
  Content: [ command|sequence|choice|pass|fail|
            match|test|get|set ]+
</choice>

<sequence>
  Content: [ command|sequence|choice|pass|fail|
            match|test|get|set ]+
</sequence>

<command
  name = NMTOKEN
  parameter = string
/>

<pass>
  Content: [ command|sequence|choice|pass|fail|
            match|test|get|set ]+
</pass>

<fail>
  Content: [ command|sequence|choice|pass|fail|
            match|test|get|set ]+
</fail>

<match
  context = XPath expr. returning a node set : "/"
  pattern = XSLT pattern
  antiPattern = boolean : false
/>

<test
  context = XPath expr. returning a node set : "/"
  expression = XPath expr. returning a boolean
/>

<get
  context = XPath expr. returning a node set : "/"
  expression = XPath expr. returning a string
/>

<set
  variable = QName
  context = XPath expr. returning a node set : "/"
  expression = XPath expression
  plainString = boolean : false
/>
```

Define, to make it simple, a sequence of native commands, menu commands, process commands or other macro-commands.

Attributes of `macro`:

`trace`

When specified with value `true`, this attribute causes the macro to print debug information on the console, which is extremely useful when developing a sophisticated macro.

`repeatable`

When specified with value `true`, this attribute marks the macro as being repeatable as a whole.

By default, macros are *not* marked as being repeatable as a whole because few macros really need this. For example, macros which are bound to a keystroke don't need to be marked repeatable.

`undoable`

When specified with value `true`, this attribute marks the macro as being undoable as a whole.

By default, macros are *not* marked as being undoable as a whole because few macros really need this. For example, macros which just select text or nodes, macros which are used to invoke process commands, macros which just perform a *single* editing action chosen by examining the editing context, don't need to be marked as undoable.

`label`

Label used by the GUI (example: the Edit popup menu) to refer to an undoable and/or repeatable macro-command.

If attribute `label` is not specified, a label is automatically generated by ``beautifying" the name under which the macro-command has been registered.

Example 1: label `"Transpose chars"` is used for macro `"transposeChars"`.

Example 2: label `"Move list item down"` is used for macro `"docb.moveListItemDown"`. In this case, simple rules are used to recognize `"docb."` as a prefix and therefore to discard it from the generated label.

Simple child elements of `macro`:

`sequence`

Can be executed if its first child can be executed (See Execution of a command). Executes all its children one after the other.

Returns the result of its last child.

`choice`

Can be executed if any of its children can be executed. Execute the first child that can be executed.

Returns the result of its executed child.

`pass`

Can be executed if all its children can be executed. Execution does nothing at all: this element is just a test.

See pass and fail cheat sheet.

Returns the result of the last executed `get`, `sequence`, `choice` or `command` (that is, does not change `%_`).

`fail`

Can be executed if any of its children cannot be executed. Execution does nothing at all: this element is just a test.

`Fail` is the negation of `pass`. See pass and fail cheat sheet.

Returns the result of the last executed `get`, `sequence`, `choice` or `command` (that is, does not change `%_`).

XPath-based child elements of `macro`:

`match`

Can be executed if the context node matches specified XSLT pattern. Execution does nothing at all: this element is just a test.

If attribute `antiPattern` is specified with value `true`, this pseudo-command can be executed if the context node *does not match* specified pattern.

Returns the result of the last executed `get`, `sequence`, `choice` or `command` (that is, does not change `%_`).

The `context` and `pattern` attributes can contain references to variables: user variables or variables mapped to the selection in XXE. See XPath variables.

`test`

Can be executed if the specified expression evaluates to true given the context node. Execution does nothing at all: this element is just a test.

Returns the result of the last executed `get`, `sequence`, `choice` or `command` (that is, does not change `%_`).

The `context` and `expression` attributes can contain references to variables: user variables or variables mapped to the selection in XXE. See XPath variables.

`get`

Can be executed if there is a context node (that is, attribute `context` evaluates to a node). Execution returns the string value of specified expression.

The `context` and `expression` attributes can contain references to variables: user variables or variables mapped to the selection in XXE. See XPath variables.

`set`

Can be executed if there is a context node. Execution assigns to specified variable the value of specified expression.

Attribute `variable` specifies the qualified name of the variable to be assigned.

Caution

Do not specify: `<set variable="$x" expression="2+2"/>`. Specify: `<set variable="x" expression="2+2"/>`.

If attribute `plainString` is specified with value `true`, attribute `expression` is considered to contain a plain string rather than an XPath expression. In this case, `expression` is not evaluated before being assigned to the variable.

Returns nothing at all (that is, clears `%_`).

The `context` and `expression` attributes can contain references to variables: user variables or variables mapped to the selection in XXE. See XPath variables.

For the above XPath-based elements, the *context node* is the result of the `context` expression (evaluated using the document as its own context node).

If the `context` expression is not specified, the context node is the document itself.

If this `context` expression evaluates to multiple nodes, the context node is the first node of the node set in document order.

If this `context` expression evaluates to anything other than a node set, the `match`, `test`, `get`, `set` pseudo-commands cannot be executed.

If this `context` expression evaluates to a node which is not attached to a document or which is attached to a document other than the one for which the macro-command is executed, the `match`, `test`, `get`, `set` pseudo-commands cannot be executed.

1.1. Pass and fail cheat sheet

<code>pass A B</code>	<code>pass</code> can be executed if <code>A</code> and <code>B</code> can be executed.
<code>pass sequence A B</code>	<code>pass</code> can be executed if <code>A</code> can be executed.
<code>pass choice A B</code>	<code>pass</code> can be executed if <code>A</code> or <code>B</code> can be executed.
<code>fail A B</code>	<code>fail</code> can be executed if <code>A</code> or <code>B</code> cannot be executed.
<code>fail sequence A B</code>	<code>fail</code> can be executed if <code>A</code> cannot be executed.
<code>fail choice A B</code>	<code>fail</code> can be executed if <code>A</code> and <code>B</code> cannot be executed.

1.2. Macro-variables

The parameter of a command `C` contained in the macro-command can contain variables. These variables are substituted with their values before executing command `C`.

Macro-variable substitution is also performed in the `context`, `pattern` and `expression` attributes of the `match`, `test`, `get`, `set` pseudo-commands.

Excerpt of example 2 below: `<command name="putAttribute" parameter="%0 %1"/>`.

Variable	Description
<code>%0, %1, %2, ..., %9, %*</code>	A macro-command can have a parameter. This string is split like in a command line. First 10 parts of the split parameter can be referenced as variables <code>%0, %1, %2, ..., %9</code> . <code>%*</code> can be used to reference the whole parameter of the macro-command.
<code>%D, %d</code>	<code>%D</code> is the file name of the document being edited. Example: <code>C:\novel\chapter1.xml</code> . This variable is replaced by an empty string if the document being edited is found on a remote HTTP or FTP server. <code>%d</code> is the URL of the document being edited. Example: <code>file:///C:/novel/chapter1.xml</code> .
<code>%P, %p</code>	<code>%P</code> is the name of the directory containing the document being edited. Example: <code>C:\novel</code> . This variable is replaced by an empty string if the document being edited is found on a remote HTTP or FTP server. <code>%p</code> is the URL of the directory containing the document being edited. Example: <code>file:///C:/novel</code> . Note that this URL does not end with a <code>'/'</code> .
<code>%N, %R, %E</code>	<code>%N</code> is the base name of the document being edited. Example: <code>chapter1.xml</code> . <code>%R</code> is the base name of the document being edited without the extension, if any (sometimes called the root name). Example: <code>chapter1</code> .

Variable	Description
	%E is the extension of the document being edited, if any. Example: xml. Note that the extension does not start with a '.'.
%S	%S is the native path component separator of the platform. Example: '\' on Windows.
%U	User's account name. Example: john.
%H	User's home directory. Example: /home/john.
%W, %w	%W is the name of the user's current working directory. Example: /home/john/docs/report22. %w is the URL of the user's current working directory. Example: file:/home/john/docs/report22. Note that this URL does not end with a '/'.
%_	A macro-command can be used to specify a ``pipe" of commands. If variable %_ is referenced in the parameter of command <i>c</i> contained in the macro-command, this variable is substituted with the result of the command executed before <i>c</i> . Each executed command overwrites %_. This variable is set to the empty string for commands which do not return a result.

The "%" character can be escaped using "%%". The above variables can be specified as %*{0}*, %*{1}*, ..., %*{R}*, %*{E}*, etc, if it helps (see note about escaped URIs).

1.3. XPath variables

Totally different from the above macro-variables. A reference to an XPath variable starts with "\$" and can only occur in *match*, *test*, *get*, *set*.

1.3.1. User variables

A user variable which name has no namespace or which name is in a namespace other than "http://www.xmlmind.com/xmleditor/namespace/scope/view" is *local* to the macro being executed.

A user variable which name is in the "http://www.xmlmind.com/xmleditor/namespace/scope/view" namespace *uses the view of the document as its scope*.

This kind of variable is persistent from an invocation of the macro to the other and/or can be shared with other macros *when this macro and/or the other macros are executed in the document view in which the variable has been created*.

A easy way to remember this is to consider that the name of this kind of variable is implicitly prefixed with the unique ID of the document view.

1.3.2. Predefined variables

There are many predefined variables, most of them mapped to the selection in the document view in which the macro is executed.

Read example: `<get expression"$selectedChars"/>` returns selected text if any, the empty string otherwise.

Write example: `<set variable="dotOffset" expression="$dotOffset + 1"/>` moves the caret by one character to the right.

Variable	Value	Read	Write
implicitElement	element	Implicitly or explicitly selected element. Empty node set if there is no node selection or if multiple nodes are selected or if the selected node is not an element.	Selects specified element. If the value of the variable is not a valid node set ^a , clears the node selection.
selectedElement	element	Explicitly selected element. Empty node set if there is no node selection or if multiple nodes are selected or if the selected node is not an element.	Selects specified element. If the value of the variable is not a valid node set ^a , clears the node selection.
implicitNode	node	Implicitly or explicitly selected node. Empty node set if there is no node selection or if multiple nodes are selected.	Selects specified node. If the value of the variable is not a valid node set ^a , clears the node selection.
selectedNode	node	Explicitly selected node. Empty node set if there is no node selection or if multiple nodes are selected.	Selects specified node. If the value of the variable is not a valid node set ^a , clears the node selection.
selectedChars	string	Characters contained in the text selection. Empty string if there is no text selection.	Selects text starting at first textual node of specified node set and ending at last textual node of specified node set. If the value of the variable is not a valid node set ^a , clears the text selection.
selectedNodes	node set containing sibling nodes	Nodes contained in the node selection. Empty node set if there is no node selection.	Selects specified nodes. Does nothing if specified nodes are not adjacent siblings. If the value of the variable is not a valid node set ^a , clears the node selection.
selected	node	First selected node in the node selection (not first in document order, first selected by user). Empty node set if there is no node selection.	Clears the node and text selections. Selects specified node. If the value of the variable is not a valid node set ^a , clears the node selection.
selected2	node	Last selected node in the node selection (not last in document order, last selected by user). Empty node set if there is no node selection or if the node selection contains a single node.	Extends node selection to specified node. Does nothing if there is no node selection or if specified node is not a sibling of selected nodes. If the value of the variable is not a valid node set ^a , clears the node selection.
dot	text, comment or PI node	Textual node containing the caret.	Moves caret at the beginning of specified textual node.

Variable	Value	Read	Write
		Empty node set if the document does not contain text, comments or PIs.	
dotOffset	integer	Offset of the caret within the textual node containing it. First offset is 0. Last valid offset is <i>after</i> last character. -1 if the document does not contain text, comments or PIs.	Moves caret to specified offset.
mark	text, comment or PI node	Textual node containing the ``mark of text selection" (text selection is between dot and mark). Empty node set if there is no text selection.	Clears the node selection. Moves the ``mark of text selection" at the beginning of specified textual node. Specified offset is adjusted if it is outside the valid offset range. If the value of the variable is not a valid node set ^a , clears the text selection.
markOffset	integer	Offset of the ``mark of text selection" within the textual node containing it. -1 if there is no text selection.	Moves the ``mark of text selection" to specified offset. Does nothing if there is no ``mark of text selection". Specified offset is adjusted if it is outside the valid offset range.
clickedElement	element	Element on which the user has clicked. If the user has clicked on a text node, then this variable contains its parent element. Can only be used when the macro-command is bound to a mouse click or an application event with an origin point such as <i>file-drop</i> .	N/A
clickedNode	node	Node on which the user has clicked. Can only be used when the macro-command is bound to a mouse click or an application event with an origin point such as <i>file-drop</i> .	N/A
clipboard	string	String contained in the system clipboard if any, the empty string otherwise.	Copies string value to the system clipboard.
systemSelection	string	String contained in the system selection ^b if any, the empty string otherwise.	Copies string value to the system selection ^b .

^aThe value of the variable must be a non empty node set.

All nodes in this node set must be attached to the document for which the macro is executed.

When a single node is needed, this node is the first node of the node set in document order.

^bThe system selection is emulated using a *private* clipboard on non Unix/X-Window platforms

2. Examples

Example 4.1. Using sequence and choice

```
<command name="addListItem">
  <macro undoable="true">
    <choice>
      <sequence>
        <command name="selectNode"
          parameter="ancestor[implicitElement] ul ol" />
        <command name="selectNode" parameter="child" />
        <command name="insertNode" parameter="sameElementAfter" />
      </sequence>

      <sequence>
        <choice>
          <sequence>
            <command name="selectNode"
              parameter="ancestorOrSelf[implicitElement] dt" />
            <!-- Assumes that a dt is followed by a dd. -->
            <command name="selectNode" parameter="nextSibling" />
          </sequence>
          <command name="selectNode"
            parameter="ancestorOrSelf[implicitElement] dd" />
        </choice>
        <command name="insert" parameter="after dt" />
        <command name="insert" parameter="after dd" />
        <command name="selectNode" parameter="previousSibling" />
      </sequence>
    </choice>
  </macro>
</command>
```

In example 1, the macro command `addListItem`, which is used to add a `li` to a `ul` or `ol` or to add a `dt/dd` pair to a `dl`, can be described as follows:

- Select ancestor `ul` or `ol` and then
 - Select previously select `child` which is always a `li` when the document is valid.

(The `selectNode` command selects all the ancestors one after the other until it reaches the searched ancestor. This is equivalent to interactively typing **Ctrl-Up** until the desired ancestor is selected.)

- AND insert element of same type (a new `li`) after selected element (a `li`).
- OR select
 - next sibling of ancestor `dt` (assumes that a `dt` is always followed by a `dd`);
 - OR ancestor `dd`.

Then

- Insert a `dt` after the selected element (a `dd`).
- AND insert a `dd` after the selected element (the newly inserted `dt`).
- AND select previous sibling (the newly inserted `dt`) of selected element (the newly inserted `dd`).

Example 4.2. Macro-variables

```

<command name="convertToLink">
  <macro undoable="true" repeatable="true" label="Convert to &lt;a&gt;">
    <sequence>
      <command name="convert" parameter="a" />
      <command name="putAttribute" parameter="%0 %1" />
    </sequence>
  </macro>
</command>

<binding>
  <keyPressed code="ESCAPE" />
  <charTyped char="t" />
  <command name="convertToLink" parameter="name XXX" />
</binding>

<binding>
  <keyPressed code="ESCAPE" />
  <charTyped char="l" />
  <command name="convertToLink" parameter="href ???" />
</binding>

```

In example 2, macro-command `convertToLink` must be passed two arguments which specify which type of XHTML a element is to be created: is it target or is it a link? These arguments are referenced in the parameter of the `putAttribute` command using variables `%0` and `%1`.

Example 4.3. The "%_" macro-variable

```

<command name="insertCommandOutput">
  <macro>
    <sequence>
      <command name="run" />
      <command name="insertString" parameter="%_" />
    </sequence>
  </macro>
</command>

```

In example 3, the output of the external program executed by the `run` command is referenced in the parameter of the `insertString` command using the `%_` variable. (The `run` command having no parameter will prompt the user to specify which external program is to be executed.)

Example 4.4. Using the fail construct

```

<command name="publish">
  <macro>
    <sequence>
      <pass>
        <match context="/*[1]" pattern="html" />
        <fail><command name="XXE.save" /></fail>
      </pass>
      <command name="doPublish" />
    </sequence>
  </macro>
</command>

```

Execute command `doPublish` if current document has an `html` root element and if this document does not need to be saved.

Example 4.5. Using the XPath-based constructs match and set

```

<command name="moveListItemUp">
  <macro undoable="true">
    <sequence>
      <command name="selectNode"
        parameter="ancestorOrSelf[implicitElement] listitem callout step" />
      <match context="$selected" pattern="*[position() > 1]" />
      <set variable="anchor" context="$selected"
        expression="./preceding-sibling::*[1]" />
      <command name="cut" />
      <set variable="selected" expression="$anchor" />
      <command name="paste" parameter="before" />
    </sequence>
  </macro>
</command>

```

Move a list item up in the list. That is, the preceding sibling of the explicitly or implicitly selected list item becomes its following sibling.

Example 4.6. A contextual file-drop

```

<binding>
  <appEvent name="file-drop" />
  <command name="xhtml.fileDrop" parameter="%{url}" />
</binding>

<command name="xhtml.fileDrop">
  <macro>
    <choice>
      <sequence>
        <match context="$clickedElement" pattern="a[@href]" />
        <set variable="selected" expression="$clickedElement" />
        <get expression="relativize-uri('%0')" />
        <command name="putAttribute" parameter="href '%_'" />
      </sequence>

      <command name="XXE.open" parameter="%0" />
    </choice>
  </macro>
</command>

```

When a file is dropped on an XHTML `` element, the URL of the dropped file is assigned to the href attribute (after trying to make it relative to the base URL of the a element). When a file is dropped on any other element, XXE default action is used instead: open the dropped file as an XML document.

Chapter 5. Process commands

1. Reference

```
<command
  name = NMTOKEN
>
  Content: class | menu | macro | process
</command>

<process>
  showProgress = boolean : true

  Content: [ copyDocument|convertImage|copyProcessResources|transform|
            processFO|upload|post|print|read|
            mkdir|rmdir|delete|copy|zip|jar|shell|invoke|subProcess ]+
</process>

<copyDocument
  to = Path
  selection = boolean : false
  preserveEntityRefs = boolean : false
  preserveXIncludes = boolean : false
  saveCharsAsEntityRefs = boolean : false
  indent = boolean : false
  encoding = (ISO-8859-1|ISO-8859-13|ISO-8859-15|ISO-8859-2|
             ISO-8859-3|ISO-8859-4|ISO-8859-5|ISO-8859-7|
             ISO-8859-9|KOI8-R|MacRoman|US-ASCII|UTF-16|UTF-8|
             Windows-1250|Windows-1251|Windows-1252|Windows-1253|
             Windows-1257) : UTF-8
>
  Content: [ extract ]* [ resources ]*
</copyDocument>

<extract
  xpath = Absolute XPath (subset)
  dataType = anyURI|hexBinary|base64Binary|XML
  toDir = Path
  baseName = File basename without an extension
  extension = File name extension
>
  <processingInstruction
    target = Name
    data = string
  /> | any element
</extract>

<resources
  match = Regexp pattern
  copyTo = Path
  referenceAs = anyURI
/>

<convertImage
  from = Glob pattern
  skip = List of file name extensions
  to = Path
  format = List of file name extensions
  lenient = boolean : false
>
  Content: [ parameter | parameterGroup ]*
</convertImage>

<parameter
  name = Non empty token
>
```

```
Content: Parameter value
</parameter>

<parameterGroup
  name = Non empty token
/>

<copyProcessResources
  resources = anyURI | @anyURI | Glob pattern
  to = Path
/>

<transform
  stylesheet = anyURI
  cacheStylesheet = boolean : false
  file = Path
  to = Path
>
Content: [ parameter | parameterGroup ]*
</transform>

<processFO
  processor = Non empty token
  file = Path
  to = Path
>
Content: [ parameter ]* [ processFO ]?
</processFO>

<upload
  base = anyURI
>
Content: [ copyFile|copyFiles ]+
</upload>

<copyFile
  file = Path
  to = anyURI
/>

<copyFiles
  files = Glob pattern
  toDir = anyURI
/>

<post
  url = anyURI
  valueCharset = Any encoding supported by Java : ISO-8859-1
  readResponse = boolean : false
>
Content: [ field ]+
</post>

<field
  name = Form field name (US-ASCII only)
>
Content: value | file
</field>

<value>
Content: xs:string
</value>

<file
  name = Path
  contentType = Content type
/>

<print
  file = Path
```

```
printer = Printer name
/>

<read
  file = Path
  encoding = Any encoding supported by Java or default
/>

<mkdir
  dir = Path
  quiet = boolean : false
/>

<rmdir
  dir = Path
  quiet = boolean : false
/>

<delete
  files = Glob pattern
  recurse = boolean : false
  quiet = boolean : false
/>

<copy
  files = Glob pattern
  to = Path
  recurse = boolean : false
  quiet = boolean : false
/>

<zip
  archive = Path
>
  Content: [ add ]+
</zip>

<add
  files = Glob pattern
  baseDir = Path : .
/>

<jar
  archive = Path
>
  Content: [ add ]+ [ manifestFile | manifest ]?
</jar>

<manifestFile>
  Content: Path
</manifestFile>

<manifest>
  Content: [ attribute ]+
</manifest>

<attribute
  name = xs:NMTOKEN (matches [0-9a-zA-Z_-]+
                        after substitution of variables)
>
  Content: xs:string
</attribute>

<shell
  command = Shell command
  platform = (Unix | Windows | Mac | GenericUnix)
/>

<invoke
  method = Qualified name of a Java static method
```

```

arguments = string : ""
/>

<subProcess
  name = NMTOKEN
  parameter = string
/>

```

Define an arbitrarily complex transformation of part or all of the document being edited.

A temporary directory is created for each execution of a process-command. This temporary directory is intended to contain all the files generated by the process.

Value type *Path* is a file path such as `images/log.gif` or `C:\temp\1.tmp`. *If this file path is relative, it is relative to the temporary process directory.* Character `'/'` can be used as a path component separator even on Windows. In fact, it is recommended to always use `'/'` as a path component separator to keep XXE configuration files portable across platforms.

Value type *Glob pattern* is a file path, possibly with wildcards such as `images/*.gif` or `..\[a-zA-Z]*`. Everything said about value type *Path* also applies to value type *Glob pattern*. It is called a *glob* pattern because it follows Unix conventions, not Windows conventions, for example `*.*` will not match a file called `the_document`.

A process-command returns the result of its last executed child element which itself returns a result (if any). The following child elements may return a result: `post`, `read`, `invoke`, `subProcess`.

1.1. Attributes

showProgress

Unless this attribute is set with value `false`, a dialog box is displayed during the execution of a process command to show the user what is happening.

Though process commands have been mainly designed to convert XML documents to other formats such as PDF, RTF or HTML, it is also possible to use them to write small, quick, yet sophisticated macro-commands. In such case, the process command/macro-command developer will probably want to:

- Set attribute `showProgress` of element `process` to value `false`.
- Set attribute `cacheStylesheet` of child element `transform` to value `true`.
- Use child element `read` associated to command `paste` or command `XXE.open` to replace part or all of the document being edited by the result of the XSLT transformation.

1.2. Element `copyDocument`

```

<copyDocument
  to = Path
  selection = boolean : false
  preserveEntityRefs = boolean : false
  preserveXIncludes = boolean : false
  saveCharsAsEntityRefs = boolean : false
  indent = boolean : false
  encoding = (ISO-8859-1|ISO-8859-13|ISO-8859-15|ISO-8859-2|
             ISO-8859-3|ISO-8859-4|ISO-8859-5|ISO-8859-7|
             ISO-8859-9|KOI8-R|MacRoman|US-ASCII|UTF-16|UTF-8|
             Windows-1250|Windows-1251|Windows-1252|Windows-1253|
             Windows-1257) : UTF-8
>
  Content: [ extract ]* [ resources ]*
</copyDocument>

<extract
  xpath = Absolute XPath (subset)
  dataType = anyURI|hexBinary|base64Binary|XML

```

```

toDir = Path
baseName = File basename without an extension
extension = file name extension
>
<processingInstruction
  target = Name
  data = string
/> | any element
</extract>

<resources
  match = Regexp pattern
  copyTo = Path
  referenceAs = anyURI
/>

```

Copy document being edited to the location specified by required attribute `to`.

1.2.1. Attributes

Attribute	Description
<code>to</code>	Specifies the file where the document (or the node selection) is to be copied.
<code>selection</code>	<p>If this attribute is specified with value <code>true</code> and if an element is explicitly selected, this element is saved to the specified location.</p> <p>If multiple nodes are explicitly selected, their parent element is saved and a special processing-instruction <code><?select-child-nodes></code>, specifying which nodes are selected, is added to the root element of the saved document.</p> <p>Example, the user has selected paragraphs with content 2, 3 and 4:</p> <pre> <div> <?select-child-nodes 3-5?> <p>1</p> <p>2</p> <p>3</p> <p>4</p> </div> </pre> <p>In the above example, 3-5 is a node range intended to be tested using <code>position()</code>, the XPath built-in function. See example 1 below to learn how to handle such multiple node selection in the XSLT style sheet.</p> <p>Otherwise, it is the whole document which is saved to the specified location.</p>
<code>preserveEntityRefs</code>	<p>If this attribute is specified with value <code>true</code>, the generated XML file contains references to (managed) external entities.</p> <p>Otherwise, references are replaced by the contents of the external entities.</p>
<code>preserveXIncludes</code>	<p>If this attribute is specified with value <code>true</code>, the generated XML file contains (managed) XIncludes.</p> <p>Otherwise, XIncludes are replaced by the contents of the elements pointed to.</p>
<code>saveCharsAsEntityRefs</code>	<p>If this attribute is specified with value <code>true</code>, the generated XML file contains references to character entities such as <code>&acute;</code>; (if needed to and if such entities are defined in the DTD of the document being edited).</p> <p>Otherwise, the generated XML file contains character references such as <code>&#233;</code>; (if needed to).</p>
<code>indent</code>	<p>If this attribute is specified with value <code>true</code>, the generated XML file is indented.</p> <p>Otherwise, the generated XML file is not indented.</p>

Attribute	Description
encoding	Specifies the encoding of the generated XML file.

1.2.2. Element `extract`

```
<extract
  xpath = Absolute XPath (subset)
  dataType = anyURI|hexBinary|base64Binary|XML
  toDir = Path
  baseName = File basename without an extension
  extension = File name extension
>
  <processingInstruction
    target = Name
    data = string
  /> | any element
</extract>
```

The `extract` element is designed to ease the writing of XSLT style sheets that need to transform XML documents where binary images (TIFF, PNG, etc) or XML images (typically SVG) are *embedded*.

In order to do this, the `extract` element copies the image data found in the element or the attribute specified by attribute `xpath` to a file created in the directory specified by attribute `toDir`.

The name of the image is automatically generated by `extract`. However, attributes `baseName` and `extension` may be used to parametrize to a certain extent the generation of the image file name.

Now the question is: how does the XSLT style sheet know about the ``extracted'' image files? The `extract` element offers two options:

- Replace the element containing image data by the one specified as a child element of `extract`.

If `xpath` selects an attribute instead of an element, the element containing the selected attribute is replaced.

DocBook example: replace embedded `svg:svg` (allowed in "-//OASIS//DTD DocBook SVG Module V1.0//EN") by much simpler `imagedata`:

```
<cfg:extract xmlns="" xpath="//imageobject/svg:svg" toDir="raw">
  <imagedata fileref="resources/{$url.rootName}.png" />
</cfg:extract>
```

- OR, more general approach, insert a processing instruction (which is specified by the `processingInstruction` child element of `extract`) at the beginning of the element from which data has been extracted.

If `xpath` selects an attribute instead of an element, the processing instruction is inserted in the element containing the selected attribute.

ImageDemo example (complete configuration found in `XXE_install_dir/doc/configure/samples/image-demo/`): insert `<?extracted extracted_file_name?>` in `imgd:image_ab` and `imgd:image_eb`:

```
<extract xpath="//imgd:image_ab/@data | //imgd:image_eb" toDir="raw">
  <processingInstruction target="extracted"
    data="resources/{$url.rootName}.png" />
</extract>
```

The replacement element (attribute values or text nodes in the element or in any of its descendant) and the inserted processing instruction (target and data) can reference the following variables which are substituted by their values during the extraction step:

Variable	Value
<code>{\$file.path}</code>	Pathname of the extracted image file. Example: <code>"/tmp/xxe1234/book_image_3.svg"</code> .

Variable	Value
{ <code>\$file.parent</code> }	Pathname of the directory containing the extracted image file. Example: <code>"/tmp/xxe1234/"</code> .
{ <code>\$file.name</code> }	Name of the extracted image file. Example: <code>"book_image_3.svg"</code> .
{ <code>\$file.rootName</code> }	Name of the extracted image file, but without an extension. Example: <code>"book_image_3"</code> .
{ <code>\$file.extension</code> }	Extension of the extracted image file name. Example: <code>"svg"</code> .
{ <code>\$file.separator</code> }	Native path component separator of the platform. Example: <code>'\'</code> on Windows.
{ <code>\$url</code> }	URL of the extracted image file. Example: <code>"file:///tmp/xxe1234/book_image_3.svg"</code> . Note Unlike <code>{<code>\$file.XXX</code>}</code> variables, the values of <code>{<code>\$url.XXX</code>}</code> variables are escaped if needed to.
{ <code>\$url.parent</code> }	URL of the directory containing the extracted image file. Example: <code>"file:///tmp/xxe1234"</code> . Note that this URL does not end with a <code>'/</code> .
{ <code>\$url.name</code> }	Name of the extracted image file. Example: <code>"book_image_3.svg"</code> .
{ <code>\$url.rootName</code> }	Name of the extracted image file, but without an extension. Example: <code>"book_image_3"</code> .
{ <code>\$url.extension</code> }	Extension of the extracted image file name. Example: <code>"svg"</code> .

In fact, any XPath expression (*full XPath 1.0*, not just the subset used in attribute `xpath`), not only variable references, can be put between curly braces (example: `{./@id}`). Such XPath expressions are evaluated as strings in the context of the element selected by attribute `xpath`. If attribute `xpath` selects an attribute, its parent element is used as an evaluation context for the XPath expression.

Attributes:

`xpath`

Selects elements and attributes containing the image data to be extracted.

This XPath expression must conform to the XPath subset needed to implement W3C XML Schemas (but not only relative paths, also absolute paths).

`dataType`

Specifies how the image data is ``stored'' in the elements or the attributes selected by the above XPath expression: `anyURI`, `hexBinary`, `base64Binary` or `XML`. This cannot be guessed for documents conforming to a DTD and for documents not constrained by a grammar.

Default: find the data type using the grammar of the document being processed.

`toDir`

Specifies the directory where extracted image files are to be created. Relative directories are relative to the temporary directory created during the execution of the process (that is, `%W`).

Default: use the temporary directory created during the execution of the process (that is, `%W`).

`baseName`

Specifies the start of the extracted image file names. An automatically generated part is always added after this user prefix.

Default: the base name of an extracted image file is automatically generated in its entirety.

extension

Specifies which extension to use for extracted image file names. Specifying "svgz" for extracted SVG images allows to create compressed SVG files.

Default: the extension is guessed by XXE for a number of common image formats.

1.2.3. Element `resources`

```
<resources
  match = Regex pattern
  copyTo = Path
  referenceAs = anyURI
/>
```

The `resources` child element specifies what to do with the resources which are logically part of the document.

The resources which are logically part of the document are specified using another configuration element: `documentResources` (see XMLmind XML Editor - Configuration and Deployment).

Note that elements replaced during an extraction step specified by the `extract` element are never scanned for resources.

The default `resources` child elements are:

```
<resources match="(https|http|ftp)://.*" />
<resources match=".*" copyTo="." />
```

Attributes of the `resources` child element:

match

For each resource of the document found using the `documentResources` element, its URI is tested to see if it matches the first `resources` child element. If it does not match the first `resources` child element, the second `resources` child element is tried and so on until a matching `resources` child element is found.

If the matching `resources` element has no `copyTo` or `referenceAs` attribute, the resource is ignored. For example, rule `<resources match="(https|http|ftp)://.*" />` is designed to ignore resources with an absolute URL.

copyTo

Specifies where to copy the matched resource. This can be a file name or a directory name.

The value of this attribute can contain \$1, \$2, ..., \$9 variables, which are substituted with the substrings matching the parenthesized groups of the `match` regular expression.

Example:

```
<resources match=".*/([/]+)\.jpg"
  copyTo="resources/$1.jpeg" />
```

Matches `images/logo.jpg`, therefore file `logo.jpg` will be copied to `resources/logo.jpeg`.

referenceAs

Specifies the reference to the resource in the document created by the `copyDocument` configuration element.

Like for `copyTo`, the value of this attribute can contain \$1, \$2, ..., \$9 variables.

Generally, this attribute is not needed because the reference implied by the value of the `copyTo` attribute is sufficient. But this attribute can be useful if images are to be converted from their original format to the format supported by a FO processor.

Example (excerpt of `XXE_install_dir/addon/config/slides/xslMenu.incl`):

```

<process>
  <mkdir dir="resources" />
  <mkdir dir="raw" />
  <copyDocument to="__doc.xml">
    <resources match="(https|http|ftp)://.*" />
    <resources match=".+\.(png|jpg|jpeg|gif)"
      copyTo="resources" />
    <resources match="(?:.+/?)(.+)\.(\w+)"
      copyTo="raw" referenceAs="resources/$1.png" />
    <resources match=".*"
      copyTo="resources" />
  </copyDocument>

  <convertImage from="raw" to="resources" format="png" />
  ...
</process>

```

1.3. Element `convertImage`

```

<convertImage
  from = Glob pattern
  skip = List of file name extensions
  to = Path
  format = List of file name extensions
  lenient = boolean : false
>
  Content: [ parameter | parameterGroup ]*
</convertImage>

<parameter
  name = Non empty token
>
  Content: Parameter value
</parameter>

<parameterGroup
  name = Non empty token
/>

```

Converts between image formats using any of the image toolkit plug-ins¹ loaded by XXE.

Attributes:

from

Specifies which image files are to be converted. If the value of this attribute is a directory, all the files contained in the directory are to be converted.

skip

The value of this attribute is a list of file name extensions. All the images specified using attribute `from` having any of these extensions must *not* be converted.

Example:

```

<convertImage from="resources" skip="gif jpeg jpg png"
  to="resources" format="png" />

```

The following case is, of course, not considered to be an error: after evaluating attributes `from` and `skip`, no image at all needs to be converted. (In fact, this is a very common case.)

to

Specifies the output image file. May be a file name or a directory name.

¹Image toolkit plug-ins are generally written in the Java™ programming language. However, the `imageToolkit` configuration element (see XMLmind XML Editor - Configuration and Deployment) may be used to turn any command line tool generating GIF, JPEG or PNG images (example: ImageMagick's `convert`) to a fully functional image toolkit plug-in for XXE.

If a directory name is used, the `format` attribute must be specified too (because without a file base name, there is no other way to know the target image format).

If after evaluating attributes `from` and `skip`, several images need to be converted, the value of the `to` attribute must be a directory name.

Examples:

```
<convertImage from="resources/logo.tiff" to="resources/pixware.jpeg" />
<convertImage from="resources/*.svg" to="resources" format="png" />
```

format

The value of this attribute is a list of file name extensions. It specifies all the possible output formats in the order of preference. Ignored unless attribute `to` specifies a directory name.

Example: the document needs to be converted to PostScript. Converting images to EPS (Encapsulated PostScript) is tried before trying to convert images to PNG.

```
<convertImage from="raw" to="resources" format="eps png" />
```

lenient

Unless this attribute is specified with value `true`, an error (a crash of the image toolkit or simply the fact that the image converter needed is not available) during the image conversion step is fatal to the whole process command.

`parameters` and `parameterGroups` may be used to fine tune the conversion. Example:

```
<convertImage from="raw/*.svg" to="resources" format="jpeg">
  <parameter name="quality">0.95</parameter>
</convertImage>
```

Which parameters are supported depend on the image toolkit used for the conversion.

Which image toolkit is used for the conversion is often obvious. In the above example, no image toolkit other than Batik can convert SVG graphics to PNG.

When several image toolkits can do the same job (example built-in Java™ toolkit and the Jimi plug-in), suffice to remember that they are tried in the order given in the dialog box displayed by menu entry Help|Plug-ins.

1.3.1. Parameters supported by the built-in Java image toolkit

Parameter	Applies to output format	Value	Description
quality	JPEG	Number between 0 and 1.	Controls the tradeoff between file size and image quality.
progressive	PNG, JPEG	true false	If true, the toolkit is to write the image in a progressive mode such that the stream will contain a series of scans of increasing quality.

1.3.2. Parameters supported by the Batik image toolkit

Parameter	Applies to output format	Value	Description
pxtomm	PNG, JPEG	A positive number.	Specify the size of a px CSS unit in millimeters. Default: 0.264583.
dpi	PNG, JPEG	A positive number.	Number of dots (that is, pixels) per inch. Similar to <code>pxtomm</code> , just easier to use. Default: 96.

Parameter	Applies to output format	Value	Description
background	PNG, JPEG	Color specified using the CSS syntax. E x a m p l e s : r g b (2 5 5 , 0 , 0) , #FF0000, red.	Specify the background color to use. Default: none (transparent background).
quality	JPEG	Number between 0 and 1.	Controls the tradeoff between file size and image quality . Default: 0.8.
indexed	PNG	1 2 4 8	If specified, the generated image will be an indexed PNG having the specified color bit depth (1-bit=2 colors, 2-bit=4colors, etc). Default: none (generate true color PNG).

1.3.3. Parameters supported by the Jimi image toolkit

Parameter	Applies to output format	Value	Description
quality	JPEG	Number between 0 and 100.	Controls the tradeoff between file size and image quality .
compression	PNG	none default fast max	Controls the tradeoff between the file size and the speed of compression.

1.4. Element `copyProcessResources`

```
<copyProcessResources
  resources = anyURI | @anyURI | Glob pattern
  to = Path
/>
```

Copy resources needed by the process to the specified location. Typically, these resources are images needed by the XSLT style sheet.

Attributes:

`resources`

Specifies which resources to copy.

If the value of the `resources` attribute is a relative URL, it is relative to the directory containing the configuration file.

Wildcards, for example `xsl/images/*.png`, are supported only if the value of the `resources` attribute is a `file:` URL (after resolving this URL against the URL of the configuration file)

It is recommended to specify multiple resources using the notation `@list-in-a-text-file`, for example `@xsl/images/list.txt`. This mechanism works even the configuration file is located on a remote server.

The URI specified in this attribute may be also resolved using XML catalogs.

`to`

Specifies the destination file. If the value of the `resources` attribute specifies multiple resources, this destination must be an existing directory.

1.5. Element `transform`

```
<transform
  stylesheet = anyURI
```

```
cacheStylesheet = boolean : false
  file = Path
  to = Path
>
  Content: [ parameter | parameterGroup ]*
</transform>

<parameter
  name = Non empty token
>
  Content: Parameter value
</parameter>

<parameterGroup
  name = Non empty token
/>
```

Converts a XML file to another format using built-in XSLT engine.

Attributes:

stylesheet

Specifies which XSLT style sheet to use. If this URL is relative, it is relative to the directory containing the XXE configuration file.

The URI specified in this attribute may be also resolved using XML catalogs.

cacheStylesheet

If this attribute is specified as `true`, a precompiled form of the XSLT style sheet is built and then cached for subsequent uses.

It is not recommended to cache an XSLT style sheet unless this style sheet is small and used in highly interactive process commands (like in example 1 below).

file

Input file.

to

Output file.

Parameter and/or named `parameterGroup` child elements are used to parametrize the XSLT style sheet. Example:

```
<parameter name="paper.type">A4</parameter>.
```

These parameters are described in the documentation of the XSLT style sheets:

- See DocBook XSL Stylesheet Documentation.

The home page of the DocBook XSL style sheets is <http://docbook.sourceforge.net/projects/xsl/>.

- See The Slides Document Type.

The home page of the Slides XSL style sheets is <http://docbook.sourceforge.net/projects/slides/>.

If a `transform` element references a `parameterGroup`, this means that a `parameterGroup` configuration element (see XMLmind XML Editor - Configuration and Deployment) with the same name is defined elsewhere in this configuration file or in another configuration file. However, it is not an error to reference a `parameterGroup` for which the configuration element is not found.

1.5.1. Using a custom XSLT style sheet

A user can force the use of a custom style sheet of his own instead of the one normally specified in attribute `stylesheet`.

In order to do this, the user needs to specify a property called `process_command_name.transform` in any XEX configuration file. The value of this property must be the URL of the custom XSLT style sheet. (This property is typically specified in the user's `customize.xxe` file. See `property` configuration element in XMLmind XML Editor - Configuration and Deployment.)

If a process command has several `transform` child elements, property `process_command_name.transform` specifies a style sheet for the first transform, `process_command_name.transform.2` specifies a style sheet for the second transform, `process_command_name.transform.3` specifies a style sheet for the third transform and so on.

Example: the process command to be customized is called `docb.toPS` (see `XEX_install_dir/addon/config/docbook/xslMenu.incl`). User has added the following property to his `customize.xxe` file.

```
<property name="docb.toPS.transform" url="true">fo_docbook.xsl</property>
```

Note that the URL is relative to the configuration file containing the definition of property `docb.toPS.transform` (here, it is relative to `customize.xxe`).

The custom XSLT style sheet `fo_docbook.xsl` contains:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format"
                version='1.0'>

<xsl:import href="xex-config:docbook/xsl/fo/docbook.xsl"/>

<xsl:template match="bookinfo/author|info/author" mode="titlepage.mode">
  <fo:block>
    <xsl:call-template name="anchor"/>
    <xsl:call-template name="person.name"/>
    <xsl:if test="affiliation/orgname">
      <fo:block>
        <xsl:apply-templates select="affiliation/orgname"
                           mode="titlepage.mode"/>
      </fo:block>
    </xsl:if>
    <xsl:if test="email|affiliation/address/email">
      <fo:block>
        <xsl:apply-templates select="(email|affiliation/address/email)[1]"/>
      </fo:block>
    </xsl:if>
  </fo:block>
</xsl:template>

</xsl:stylesheet>
```

Note how the stock `docbook.xsl` is imported by this customized version.

Tip

In our opinion, it is almost impossible to cope with the complexity of customizing Norman Walsh's DocBook XSLT style sheets without reading this excellent book: *DocBook XSL: The Complete Guide - Second Edition* by Bob Stayton. See <http://www.sagehill.net/book-description.html>.

1.6. Element `processFO`

```
<processFO
  processor = Non empty token
  file = Path
  to = Path
>
  Content: [ parameter ]* [ processFO ]?
</processFO>
```

```

<parameter
  name = Non empty token
>
  Content: Parameter value
</parameter>

<parameterGroup
  name = Non empty token
/>

```

Converts a XSL-FO file to another format, typically a page description language such as PDF.

Attributes:

`processor`

Specifies which FO processor to use.

Unlike the XSLT engine used by a `transform` element, the FO processor used to perform this conversion is not built-in into XXE. A FO processor plug-in having a name equals the value of the `processor` attribute (case-insensitive) must have been registered with XXE.

`file`

Input file.

`to`

Output file.

`parameter` child elements are passed, as is, to the FO processor in order to parametrize its behavior. These parameters are described in the documentation of the FO processors.

XFC example:

```
<parameter name="outputEncoding">Cp1252</parameter>
```

XEP example:

```
<parameter name="PS.LANGUAGE_LEVEL">2</parameter>
```

Pseudo-parameters:

XEP

The XEP FO processor understands a number of actual parameters but also a *pseudo-parameter* called `OUTPUT_FORMAT`. Its value specifies which is the target format of XEP: `pdf`, `ps`.

If this pseudo-parameter is absent, which format to use is guessed from the extension of the output file name.

FOP

The FOP FO processor understands a number of actual parameters, but also the following pseudo-parameters:

`renderer`

Its value specifies which renderer to use: `pdf`, `ps`, `pcl`, `svg`, `xml`, `mif`, `txt`.

If this pseudo-parameter is absent, which renderer to use is guessed from the extension of the output file name.

`configuration`

Specifies the *absolute* URL or filename of a FOP user configuration file. Such configuration files are useful to specify font metrics, hyphenation files, etc. More information on FOP's web site.

The `processFO` optional child element:

This optional child element specifies which FO processor to use when the FO processor specified by the parent `processFO` element is not available.

Example: try to use FOP when XEP is not available:

```
<processFO processor="XEP" file="__doc.fo" to="__doc.pdf">
  <parameter name="OUTPUT_FORMAT">pdf</parameter>

  <processFO processor="FOP" file="__doc.fo" to="__doc.pdf">
    <parameter name="renderer">pdf</parameter>
  </processFO>
</processFO>
```

1.7. Element `upload`

```
<upload
  base = anyURI
>
  Content: [ copyFile|copyFiles ]+
</upload>

<copyFile
  file = Path
  to = anyURI
/>

<copyFiles
  files = Glob pattern
  toDir = anyURI
/>
```

Copies files generated during the process to remote FTP or WebDAV servers or to the local file system. Can create directories on the fly if needed to.

Child elements:

`copyFile`

Specifies a single file to be copied to the upload destination.

The URL specified by the `to` attribute is resolved against the URL specified by the `base` attribute.

`copyFiles`

Specifies possibly multiple files to be copied to the upload destination. If a matched file is a directory, it will be recursively copied. Has no effect if specified wildcard does not match any file.

The URL specified by the `toDir` attribute is resolved against the URL specified by the `base` attribute.

1.8. Element `post`

```
<post
  url = anyURI
  valueCharset = Any encoding supported by Java : ISO-8859-1
  readResponse = boolean : false
>
  Content: [ field ]+
</post>

<field
  name = Form field name (US-ASCII only)
>
  Content: value | file
</field>

<value>
  Content: xs:string
</value>

<file
  name = Path
```

```
contentType = Content type
/>
```

Emulates an HTML form possibly containing `input type="file"` elements. The `post` element posts the content of the emulated form to the CGI or Servlet specified by attribute `url`, using the `multipart/form-data` method.

An emulated form field has a name specified by required attribute `name`. There are two type of fields:

value

Emulates `input type="text"` or `input type="hidden"` elements found in an HTML form. The content of this element, a possibly empty string, specifies the value of the field.

Attribute `valueCharset` of the `post` element specifies the charset used for *all* `value` fields. By default, this charset is `ISO-8859-1`.

file

Emulates `input type="file"` elements found in an HTML form. The `name` attribute of this element specifies the filename of the file to be uploaded.

Unless specified, the content type of the file is guessed using the extension of the filename. If the filename ends with:

`.zip`
the content type is supposed to be `application/zip`;

`.jar`
the content type is supposed to be `application/x-java-archive`;

`.xml`
the content type is supposed to be `text/xml`.

Otherwise, the content type is supposed to be `application/octet-stream`.

If attribute `readResponse` is specified with value `true`, this element returns the response of the server. Otherwise, this element returns no result at all.

Moreover, for this element to return a result, the server must respond to the post request with a success code different from "No Content" (204) and must send `text/*` data (e.g. `text/plain`, `text/html`, etc). If the content type of the sent data has no charset, the data is read as a string using charset `"ISO-8859-1"`.

Examples:

```
<post url="http://localhost:8080/measure/archive">
  <field name="op">
    <value>add</value>
  </field>
  <field name="user">
    <value>%U</value>
  </field>
  <field name="data">
    <file name="/tmp/1052_3_CO_3.1R" />
  </field>
</post>
```

```
<post url="http://localhost:8080/measure/archive"
  valueCharset="US-ASCII" readResponse="true">
  <field name="op">
    <value>add</value>
  </field>
  <field name="user">
    <value>%U</value>
  </field>
  <field name="interactive">
    <value>>false</value>
```

```

</field>
<field name="data">
  <file name="1052_3_CO_3.1R"
    contentType="text/xml; charset=ISO-8859-1" />
</field>
</post>

```

1.9. Element `print`

```

<print
  file = Path
  printer = Printer name
/>

```

Sends the file specified by attribute `file` (typically a PostScript® file) to the printer specified by attribute `printer`.

Example:

```

<command name="docb.toPSPrinter">
  <process>
    <subProcess name="docb.toPS" parameter="'%0' '%1' '%2' '%3'" />

    <print file="__doc.%0" printer="%4" />
  </process>
</command>

```

The syntax of `printer name` is:

```
printer_name [ format ]?
```

Examples:

```

lp22
'Sales Departement' ps
lp23 'text/plain; charset=UTF-8'

```

The optional `format` part specifies the mime type of the document to be printed. The following short names are also supported: `ps` (`application/postscript`), `pdf` (`application/pdf`), `pcl` (`application/vnd.hp-PCL`).

When this `format` part is not specified, the `print` element uses the extension of the filename specified in attribute `file`.

Command `selectPrinter` allows to choose a printer using a specialized dialog box and returns a ready-to-use printer name for the `print` element of a `process` command. Example:

```

<command name="docb.printPS">
  <macro>
    <sequence>
      <command name="selectPrinter" parameter="%0" />
      <command name="docb.toPSPrinter" parameter="'%0' '%1' '%2' '%3' '%_'" />
    </sequence>
  </macro>
</command>

```

1.10. Element `read`

```

<read
  file = Path
  encoding = Any encoding supported by Java or default
/>

```

Loads the content of specified text file and returns this content.

If `encoding` is specified as `default`, the encoding of the text file is the native encoding of the platform, for example `windows-1252` on an US Windows machine.

1.11. Element `mkdir`

```
<mkdir
  dir = Path
  quiet = boolean : false
/>
```

Creates specified directory. If parent directories needs to be created in order to create specified directory, these parent directories are created too.

Will report an error if specified directory cannot be created. Attribute **quiet** can be used to suppress the error message when specified directory already exists.

1.12. Element `rmdir`

```
<rmdir
  dir = Path
  quiet = boolean : false
/>
```

Removes specified directory. For this operation to succeed, the specified directory must be empty.

Will report an error if specified directory cannot be removed. Attribute **quiet** can be used to suppress the error message when specified directory does not exist.

1.13. Element `delete`

```
<delete
  files = Glob pattern
  recurse = boolean : false
  quiet = boolean : false
/>
```

Deletes specified files.

Has no effect is specified wildcard does not match any file.

Will report an error if one of the specified files cannot be deleted. Attribute `quiet` can be used to suppress the error message when one of the specified files does not exist.

If attribute `recurse` is specified with value `true`, it will also recursively delete specified directories. Otherwise, if one of the specified files is a directory, it will report an error message.

1.14. Element `copy`

```
<copy
  files = Glob pattern
  to = Path
  recurse = boolean : false
  quiet = boolean : false
/>
```

Copies files and directories specified by attribute `files` to the file or directory specified by attribute `to`.

Has no effect is specified wildcard does not match any file.

If specified wildcard matches several files or directories, the destination must be an existing directory.

Directories will not be copied unless attribute `recurse` is specified with value `true`.

Attribute `quiet` can be used to suppress the error message when one of the specified files does not exist or when one of the specified files is a directory (and attribute `recurse` is different from `true`).

1.15. Element zip

```
<zip
  archive = Path
>
  Content: [ add ]+
</zip>

<add
  files = Glob pattern
  baseDir = Path : .
/>
```

Creates a zip archive located at `archive` containing the files specified by the `add` child elements.

See also `jar`.

Example: Let's suppose current working directory contains:

```
/tmp$ ls -R
doc.xml
doc.xml~
doc.xml.SAVE

./attachments:
data1.bin
data1.zip
data2.bin
data2.zip

./resources:
logo.png
chart1.jpeg
```

```
<zip archive="all.zip">
  <add files="doc.xml" />
  <add files="resources/*" />
  <add files="misc/*" />
  <add files="*.bin" baseDir="attachments" />
</zip>
```

The above `zip` element creates in current working directory, an archive called `all.zip`, containing:

```
/tmp$ unzip -v all.zip
doc.xml
resources/
resources/logo.png
resources/chart1.jpeg
data1.bin
data2.bin
```

Note that non-existent directory `misc/` will not cause the `zip` element to stop its processing or to report a warning.

1.16. Element jar

```
<jar
  archive = Path
>
  Content: [ add ]+ [ manifestFile | manifest ]?
</jar>

<add
  files = Glob pattern
  baseDir = Path : .
/>

<manifestFile>
  Content: Path
```

```

</manifestFile>

<manifest>
  Content: [ attribute ]+
</manifest>

<attribute
  name = xs:NMTOKEN (must match [0-9a-zA-Z_-]+ after
                    substitution of process variables)
>
  Content: xs:string
</attribute>

```

Similar to zip, except that the archive always contains a manifest (even if this manifest is empty). The manifest, if any, is specified using a `manifestFile` child element or a `manifest` child element.

Examples:

```

<jar archive="doc.jar">
  <add files="doc.xml" />
  <add files="images/*.gif" />
  <manifestFile>/tmp/manifest</manifestFile>
</jar>

<jar archive="doc2.jar">
  <add files="doc.xml" />
  <add files="images/*.gif" />
  <manifest>
    <attribute name="Master-Document">doc.xml</attribute>
    <attribute name="Publication-Date">%0</attribute>
    <attribute name="Self-Contained"></attribute>
  </manifest>
</jar>

```

1.17. Element `shell`

```

<shell
  command = Shell command
  platform = (Unix | Windows | Mac | GenericUnix)
/>

```

Executes specified command using native shell: **cmd.exe** on Windows and **/bin/sh** on Unix (Mac OS X is considered to be a Unix platform).

The current working directory of the native shell is the temporary directory created for the execution of the process-command (%W, see below).

If the `platform` attribute is not specified, the shell command is executed whatever is the platform running XXE.

If the `platform` attribute is specified, the shell command is executed only if the platform running XXE matches the value of this attribute:

Windows

Any version of Windows.

Mac

Mac OS X.

GenericUnix

A Unix which is not Mac OS X (Linux, Solaris, etc).

Unix

GenericUnix or Mac.

1.18. Element `invoke`

```
<invoke
  method = Qualified name of a Java static method
  arguments = string : ""
/>
```

Invokes specified Java™ *static* method, passing it specified string as an argument.

- The method generally belongs to a class which is contained in a `jar` dynamically discovered by XXE at startup time.
- The method must have one of the following signatures:
 - `method(java.lang.String arguments, java.io.File workingDir);`
 - `method(java.lang.String arguments, java.io.File workingDir, com.xmlmind.xmledit.util.Console console);`
 - `method(java.lang.String arguments, java.io.File workingDir, com.xmlmind.xmledit.util.Console console, com.xmlmind.xmledit.doc.Document docBeingEdited);`

arguments

The value of the `arguments` attribute after substituting all variables (`%0`, `%1`, `%D`, `%p`, `%C`, etc).

All arguments, if any, are passed as a *single string*. That is, the method is responsible for properly parsing this string.

workingDir

The temporary directory created each time a process command is executed. Relative paths are generally relative to this directory.

console

A simple way to report information and non fatal errors to the user of the process command. Throw an exception to report a fatal error.

docBeingEdited

The document being edited.

- The method may return a value. If it returns a value, this value is converted to a `java.lang.String` using `toString()` and then returned by the `invoke` element (à la read, for use in a macro command for example).
- The method may throw any exception.

Examples:

```
<invoke method="TestInvoke.echo"
  arguments="args={%*} doc='%D' pwd='%W' conf='%C'" />
<invoke method="TestInvoke.echo2" />
<invoke method="TestInvoke.gzip" arguments="__doc.xml" />
```

Static methods invoked by the above examples:

```
public final class TestInvoke {
  public static void echo(String arguments, File workingDir,
    Console console) {
```

```

        console.showMessage("arguments='" + arguments + "'", Console.INFO);
        console.showMessage("workingDir='" + workingDir + "'", Console.INFO);
    }

    public static void echo2(String arguments, File workingDir,
        Console console, Document docBeingEdited) {
        echo(arguments, workingDir, console);
        console.showMessage("docBeingEdited='" + docBeingEdited.getLocation()
            + "'", Console.INFO);
    }

    public static File gzip(String arguments, File workingDir)
        throws IOException {
        File inFile = new File(workingDir, arguments.trim());
        if (!inFile.isFile())
            throw new FileNotFoundException(inFile.getPath());
        File outFile = new File(inFile.getPath() + ".gz");
        FileInputStream in = new FileInputStream(inFile);

        try {
            GZIPOutputStream out =
                new GZIPOutputStream(new FileOutputStream(outFile));

            byte[] bytes = new byte[8192];
            int count;

            while ((count = in.read(bytes)) != -1)
                out.write(bytes, 0, count);

            out.finish();
            out.close();
        } finally {
            in.close();
        }

        return outFile;
    }
}

```

1.19. Element subProcess

```

<subProcess
  name = NMTOKEN
  parameter = string
/>

```

Invokes the `process` command whose name is specified by attribute `name`. Optional attribute `parameter` may be used to parametrize the behavior of the invoked process command.

This element returns the result of its last executed child element which itself returns a result (if any).

Examples:

```

<command name="docb.toPSFile">❶
  <process>
    <subProcess name="docb.toPS" parameter="'%0' '%1' '%2' '%3'" />

    <upload base="%4">
      <copyFile file="__doc.%0" to="%4" />
    </upload>
  </process>
</command>

<command name="docb.toPSPrinter">❷
  <process>
    <subProcess name="docb.toPS" parameter="'%0' '%1' '%2' '%3'" />

    <print file="__doc.%0" printer="%4" />

```

```
</process>
</command>
```

- 1 First process command is used to convert a DocBook document to PostScript® or to PDF.
- 2 Second process command is used to print a DocBook document on a PostScript® or a PDF printer (after converting it to these formats, of course).

Both process commands invoke `docb.tops` which actually does the job of converting a DocBook document to PostScript® or to PDF.

1.20. Process variables

Almost all child elements and attribute values in a `process` element can include variables which are substituted just before the execution of the process-command. Example: `<upload base="%0/">`.

Variable	Description
<code>%0, %1, %2, ..., %9, %*</code>	A process-command can have a parameter. This string is split like in a command line. First 10 parts of the split parameter can be referenced as variables <code>%0, %1, %2, ..., %9</code> . <code>%*</code> can be used to reference the whole parameter of the process-command.
<code>%D, %d</code>	<code>%D</code> is the file name of the document being edited. Example: <code>C:\novel\chapter1.xml</code> . This variable is replaced by an empty string if the document being edited is found on a remote HTTP or FTP server. <code>%d</code> is the URL of the document being edited. Example: <code>file:///C:/novel/chapter1.xml</code> .
<code>%P, %p</code>	<code>%P</code> is the name of the directory containing the document being edited. Example: <code>C:\novel</code> . This variable is replaced by an empty string if the document being edited is found on a remote HTTP or FTP server. <code>%p</code> is the URL of the directory containing the document being edited. Example: <code>file:///C:/novel</code> . Note that this URL does not end with a <code>'/'</code> .
<code>%N, %R, %E</code>	<code>%N</code> is the base name of the document being edited. Example: <code>chapter1.xml</code> . <code>%R</code> is the base name of the document being edited without the extension, if any (sometimes called the root name). Example: <code>chapter1</code> . <code>%E</code> is the extension of the document being edited, if any. Example: <code>xml</code> . Note that the extension does not start with a <code>'.'</code> .
<code>%S</code>	<code>%S</code> is the native path component separator of the platform. Example: <code>'\'</code> on Windows.
<code>%U</code>	User's account name. Example: <code>john</code> .
<code>%H</code>	User's home directory. Example: <code>/home/john</code> .
<code>%W, %w</code>	<code>%W</code> is the name of the temporary process directory. Example: <code>C:\temp\xxe1023E45</code> . <code>%w</code> is the URL of the temporary process directory. Example: <code>file:///C:/temp/xxe1023E45</code> . Note that this URL does not end with a <code>'/'</code> .

Variable	Description
%C, %c	<p>%C is the name of the directory containing the XXE configuration file from which the process command has been loaded. Example: C:\Program Files\XMLmind_XML_Editor\addon\config\docbook.</p> <p>%c is the URL of the above directory. Example: file:///C:/Program%20Files/XMLmind_XML_Editor/addon/config/docbook.</p> <p>Note that this URL does not end with a '/'.</p>

The "%" character can be escaped using "%%". The above variables can be specified as %{0}, %{1}, ..., %{R}, %{E}, etc, if it helps (see note about escaped URIs).

Note

Most attribute and element values described in this documentation as being URIs (data type anyURI) in fact *are not URIs*. These attribute and element values can contain %-variables. They are required to be valid URIs only *after* the %-variables have been substituted with their values.

The problem is that URIs may also contain escaped characters which look very much like references to %-variables. For example, a whitespace must be escaped as "%20", which looks like a reference to variable %2 followed by literal string "0".

In practice:

1. It is recommended to specify variables as %{0}, %{1}, ..., %{d}, %{E}, etc, rather than as %0, %1, ..., %d, %E, etc, because it makes clear what is a variable reference and what is an escaped character.
2. An escaped character such as "%20" must be specified as "%%20" because variable substitution occurs before the URIs are used and because, during variable substitution, a real percent character can be protected against substitution by doubling it.

Example: relative URI "docs/my report/my.doc.%0", where variable %0 represents a file extension, must be specified as "docs/my%%20report/my%%20doc.%0".

2. Commented examples

2.1. Convert explicitly or implicitly selected `para` to a `simpara`

This example can be found in `XXE_install_dir/doc/configure/samples2/simpara.xxe`.

```
<command name="toSimpara">
  <process showProgress="false">
    <copyDocument selection="true" to="in.xml" />
    <transform stylesheet="simpara.xslt" cacheStylesheet="true"
      file="in.xml" to="out.xml" />
    <read file="out.xml" encoding="UTF-8" />
  </process>
</command>

<command name="paraToSimpara">
  <macro>
    <sequence>
      <command name="selectNode"
        parameter="ancestorOrSelf[implicitElement] para" />
      <command name="toSimpara" />
      <command name="paste" parameter="to %_" />
    </sequence>
  </macro>
</command>
```

```
<binding>
  <keyPressed code="ESCAPE" />
  <keyPressed code="S" modifiers="mod" />
  <command name="paraToSimpara" />
</binding>
```

In the above example, `simpara.xlst` is simply:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" encoding="UTF-8" />

<xsl:template match="para">
  <simpara>
    <xsl:copy-of select="./node()" />
  </simpara>
</xsl:template>

</xsl:stylesheet>
```

Note

Adding the following generic rule to *any* XSLT style sheet used in interactive process commands allows to handle the case where the user has selected multiple nodes:

```
<xsl:template match="/*[./processing-instruction('select-child-nodes')]">
  <xsl:variable name="pi"
    select="./processing-instruction('select-child-nodes')" />
  <xsl:variable name="first" select="substring-before($pi, '-')" />
  <xsl:variable name="last" select="substring-after($pi, '-')" />

  <c:clipboard
    xmlns:c="http://www.xmlmind.com/xmleditor/namespace/clipboard">

    <xsl:for-each select="child::node()[position() >= $first and
      position() <= $last]">
      <xsl:apply-templates select="." />
    </xsl:for-each>

  </c:clipboard>
</xsl:template>
```

2.2. Convert a DocBook document to RTF

This example is taken from `XXE_install_dir/addon/config/docbook/xslMenu.incl`.

```
<command name="docb.convertToRTF">
  <macro>
    <sequence>
      <command name="selectFile" parameter="saveFileURL" />
      <command name="docb.toRTF" parameter="'%0' '%1' '%_'" />1
    </sequence>
  </macro>
</command>

<command name="docb.toRTF">
  <process>
    <mkdir dir="resources" />2
    <mkdir dir="raw" />3
    <copyDocument to="__doc.xml">4

    <!-- Do *not* declare the svg namespace. We want svg:svg to be
      a Name, not a QName. -->
    <cfg:extract xmlns="" xpath="//imageobject/svg:svg" toDir="raw">5
      <imagedata fileref="resources/{url.rootName}.png" />
    </cfg:extract>
```

```

<resources match="(https|http|ftp)://.*" />6
<resources match=".+\.(png|jpg|jpeg|gif)"
  copyTo="resources" />7
<resources match="(?:.+/)?(.+)\.(\w+)"
  copyTo="raw" referenceAs="resources/$1.png" />8
<resources match="."+
  copyTo="resources" />
</copyDocument>

<convertImage from="raw" to="resources" format="png" />9

<mkdir dir="images/callouts" />10
<copyProcessResources resources="xsl/images/draft.png" to="images" />
<copyProcessResources resources="@xsl/images/callouts/png_callouts.list"
  to="images/callouts" />

<transform stylesheet="xsl/fo/docbook.xsl"
  file="__doc.xml" to="__doc.fo">11
  <parameter name="use.extensions">1</parameter>
  <!-- Cannot work and generates a lot of error messages. -->
  <parameter name="graphicsize.extension">0</parameter>

  <parameter name="paper.type">A4</parameter>

  <parameter name="generate.toc">%0</parameter>
  <parameter name="toc.section.depth">3</parameter>
  <parameter name="section.autolabel">%1</parameter>

  <parameter name="callout.graphics">1</parameter>

  <parameter name="shade.verbatim">1</parameter>

  <parameter name="ulink.show">0</parameter>

  <parameterGroup name="docb.toRTF.transformParameters" />12
</transform>

<processFO processor="XFC" file="__doc.fo" to="__doc.rtf">13
  <parameter name="outputEncoding">Cp1252</parameter>
  <parameterGroup name="docb.toRTF.XFCParameters" />
</processFO>

<upload base="%2">14
  <copyFile file="__doc.rtf" to="%2" />
</upload>
</process>
</command>

```

1 The `docb.toRTF` process command is passed 3 arguments:

`%0`

For which elements a Table Of Contents (TOC) is to be created. Example: `"/book toc /article toc"`.

`%1`

1 if a TOC is to be generated, 0 otherwise.

`%2`

The URL of the RTF file to be created.

- 2** Images referenced in the DocBook document which are in formats supported by the XFC FO processor (GIF, JPEG and PNG) will be copied to directory `resources/`.
- 3** Images referenced in the DocBook document which are in formats not supported by the XFC FO processor will be copied to directory `raw/` in order to be converted.
- 4** Copy document being edited as `__doc.xml` in the temporary process directory.

The copied document is flattened: all references to external entities and all XIncludes are expanded.

As specified by `extract` and `resources`, references to resources such as external graphics files (example: `<imagedata fileref="XXX"/>`) are modified in the copied document to point to copies which are local to the temporary process directory.

- 5 Embedded SVG graphics (`svg:svg` allowed in "-//OASIS//DTD DocBook SVG Module V1.0//EN") are extracted to directory `raw/`.

The `svg:svg` element is replaced by a properly initialized `imagedata` element which is easily handled by the XSLT style sheet and then by the FO processor (it is translated by the XSLT style sheet to a `fo:external-graphic`).

Caution

DocBook documents are conforming to a DTD and therefore they are *not namespace aware* in XXE. That's why the `svg` prefix must *not* be declared.

By doing this, `svg:svg` is understood by XXE as being `{ }svg:svg`. If you declare the `svg` prefix, `svg:svg` is understood as being `{http://www.w3.org/2000/svg}svg`, which is wrong in the case of DocBook+SVG.

- 6 References to really absolute resources are not modified in the copy of the document.
- 7 References to PNG, GIF, JPEG graphics files are modified to point to the copies which are made in directory `resources/`.
- 8 References to other graphics files are modified to point to the converted images that will be generated in directory `resources/`. The graphics files in formats other than PNG, GIF, JPEG are copied as is in directory `raw/`, waiting to be converted.
- 9 Converts all images found in directory `raw/` to PNG images created in directory `resources/`.
- 10 Copies resources internally used by the `xsl/fo/docbook.xsl` XSLT style sheet to where the FO processor can find them.
- 11 Transforms the copy of the document `__doc.xml` to XSL-FO file `__doc.fo`.
- 12 This `parameterGroup` allows XXE users to easily customize the XSLT style sheet by adding or replacing parameters.

Example of such `parameterGroup` added to `XXE_user_preferences_dir/addon/customize.xxe`:

```
<parameterGroup name="docb.toRTF.transformParameters">
  <parameter name="callout.graphics">0</parameter>
  <parameter name="variablelist.as.blocks">1</parameter>
</parameterGroup>
```

- 13 Convert XSL-FO file `__doc.fo` to local RTF file `__doc.rtf`.
- 14 Copies local RTF file `__doc.rtf` to its user-specified destination.

The element is called `upload` because, in Professional Edition, it can be used to *publish* the converted document by sending it (and all its associated resources, if needed to) to a remote FTP or WebDAV server.

2.3. Convert ImageDemo document to HTML

This example is taken from `XXE_install_dir/doc/configure/samples/imagdemo/imagdemo.xxe`.

The ImageDemo configuration has been created to teach external consultants and local gurus how to configure XXE for XML documents *embedding binary or XML images*.

```
<command name="imgd.convertToHTML">
  <macro>
    <sequence>
      <command name="selectFile" parameter="saveFileURL" />
      <command name="imgd.toHTML" parameter="'%_'" />
    </sequence>
  </macro>
</command>

<command name="imgd.toHTML">
  <process>
```

```

<mkdir dir="resources" />
<mkdir dir="raw" />
<copyDocument to="__doc.xml">
  <extract xpath="//imgd:image_ab/@data | //imgd:image_eb" toDir="raw">❶
    <processingInstruction target="extracted"
      data="resources/{\$url.rootName}.png" />
  </extract>
  <extract xpath="//imgd:*/*:svg:svg" toDir="raw">
    <processingInstruction target="extracted"
      data="resources/{\$url.rootName}.png" />
  </extract>

  <resources match="(https|http|ftp)://.*" />
  <resources match=".+\\. (png|jpg|jpeg|gif)"
    copyTo="resources" />
  <resources match="(?:.+)?(\\.+)\\. (\\w+)"
    copyTo="raw" referenceAs="resources/\\$1.png" />
  <resources match=".+"
    copyTo="resources" />
</copyDocument>

<convertImage from="raw" to="resources" format="png" />

<mkdir dir="xslt_graphics" />
<copyProcessResources resources="xslt_graphics/*" to="xslt_graphics" />

<transform stylesheet="html.xslt"
  file="__doc.xml" to="__doc.html" />

<upload base="%0">❷
  <copyFile file="__doc.html" to="%0" />
  <copyFiles files="resources/*" toDir="resources" />
  <copyFiles files="xslt_graphics/*" toDir="xslt_graphics" />
</upload>
</process>
</command>

```

If you can follow the previous example, you can follow this one too because they are very similar. The main differences are:

- ❶ Instead of extracting the SVG graphics from `svg:svg` and replacing this element by another one such as `imgd:image_au`, it is much simpler to insert an extracted processing instruction inside `imgd:image_ab`, `imgd:image_eb` and `svg:svg`.

Doing this spares the effort of copying all the image geometry attributes, `width`, `height`, `content_width`, `content_height`, etc, from the extracted element to the replacement `imgd:image_au` element.

Also note that ImageDemo documents being namespace-aware, it is mandatory to declare the `svg` prefix. This is done at the top of `xxe_install_dir/doc/configure/samples/imagdemo/imagdemo.xxe`.

- ❷ Unlike an RTF file, an HTML file is not self-contained. All the graphics files found in `resources/` and in `xslt_graphics/` need to be copied along the generated HTML file.

3. The `convertdoc` command line tool

The `convertdoc` tool allows to execute XXE process commands from the command line, exactly as if these process commands were executed from XXE.

Just like XXE, the `convertdoc` command line tool scans directories for configurations and XML catalogs, loads image toolkit plug-ins, loads FO processor plug-ins, supports XInclude, etc.

Note that `convertdoc` Standard Edition has exactly the same artificial restrictions than XXE Standard Edition: limited support of W3C XML Schemas, no support of FO processor (except XEP Trial Edition), image toolkit plug-ins cannot be used to convert on the fly between different image formats, etc.

Usage:

```
convertdoc -l
```

or:

```
convertdoc ?-v? ?-t XSLT_stylesheet_file_or_URL?
?-p XSLT_stylesheet_param_name XSLT_stylesheet_param_value? ...
?-p XSLT_stylesheet_param_name XSLT_stylesheet_param_value?
process_command_name doc_file_or_URL
?-s|-u process_command_arg? ... ?-s|-u process_command_arg?
```

Converts XML document *doc_file_or_URL* using process command called *process_command_name*, found in any of the XXE configuration files scanned during the startup of **convertdoc** (see XMLmind XML Editor - Configuration and Deployment).

Options:

-l
Print XXE configuration (XXE configuration files but also XML catalogs, plug-ins, spell-checker dictionaries, CLASSPATH) and exit.

-v
Turn verbosity on.

-t *XSLT_stylesheet_file_or_URL*
Use this alternate XSLT style sheet instead of the one specified in the first `transform` child element of the process command.

If specified process command has no `transform` child element but has `subProcess` child elements, these sub-processes are searched recursively for a `transform` child element.

-p *XSLT_stylesheet_param_name XSLT_stylesheet_param_value*
Add/replace corresponding XSLT style sheet parameter in the first `transform` child element of the process command.

If specified process command has no `transform` child element but has `subProcess` child elements, these sub-processes are searched recursively for a `transform` child element.

-s|-u *process_command_arg ... -s|-u process_command_arg*
Pass these arguments to the process command as the values of process variables %0, %1, ..., %9.

If `-s` (String) is specified, the argument is passed as is.

If `-u` (URL) is specified, the argument, a file or directory name, is first converted to an URL.

3.1. Convert a DocBook document to multi-page HTML

Convert DocBook document `help.xml` to multi-page HTML created in directory `docs/help/` (the `docb.toHTML` process command is found in `XXE_install_dir/addon/config/docbook/xslMenu.incl`).

```
$ convertdoc -p toc.section.depth 4 -p chunk.section.depth 2 \
docb.toHTML help.xml \
-u docs/help
```

Figure 5.1. Excerpts of `docb.toHTML`

```

<command name="docb.toHTML">
  <process>
    .
    .
    .
    <parameter name="chunk.first.sections">1</parameter>
    <parameter name="chunk.section.depth">1</parameter>
    <parameter name="toc.section.depth">3</parameter>
    <parameter name="section.autolabel">1</parameter>
    .
    .
    .
    <upload base="%0/">
      <copyFiles files="*.*" toDir="." />
      <copyFiles files="resources/*" toDir="resources" />
      <copyFiles files="images/*" toDir="images" />
    </upload>
  </process>
</command>

```

The `docb.toHTML` process command expects a save directory URL as its `%0` argument. This `%0` argument is passed to the process command using `"-u docs/help"`.

3.2. Example 2: Convert a DocBook document to PDF

Convert DocBook document `doc.xml` to `commands.pdf` (the `docb.toPSFile` process command is found in `XXE_install_dir/addon/config/docbook/xslMenu.incl`).

```

$ convertdoc -t fo_docbook.xsl \
  -p toc.section.depth 4 -p callout.graphics 0 -p variablelist.as.blocks 1 \
  docb.toPSFile doc.xml \
  -s pdf -s "|pdf" -s "/book toc" -s 1 -u docs/commands/commands.pdf

```

Note that an alternate XSLT style sheet, `fo_docbook.xsl`, is used instead of the stock `docbook.xsl` which is part of Norman Walsh's DocBook XSL stylesheets.

Figure 5.2. `docb.toPSFile` and excerpts of `docb.toPS`

```

<command name="docb.toPS">
  <process>
    .
    .
    .
    <resources match="(?:.+/)?(.*?)\.(png|jpg|jpeg|gif|svg|svgz%1)"
      copyTo="resources" referenceAs="%w/resources/$1.$2" />
    .
    .
    .
    <parameter name="generate.toc">%2</parameter>
    <parameter name="toc.section.depth">3</parameter>
    <parameter name="section.autolabel">%3</parameter>
    .
    .
    .
    <processFO processor="FOP" file="__doc.fo" to="__doc.%0">
      <parameter name="renderer">%0</parameter>
      <parameterGroup name="docb.toPS.FOPParameters" />
    </processFO>
    .
    .
    .
    <upload base="%4">
      <copyFile file="__doc.%0" to="%4" />
    </upload>
  </process>
</command>

<command name="docb.toPSFile">
  <process>
    <subProcess name="docb.toPS" parameter="'%0' '%1' '%2' '%3'" />

    <upload base="%4">
      <copyFile file="__doc.%0" to="%4" />
    </upload>
  </process>
</command>

```

The `docb.toPSFile` process command expects 5 arguments:

`%0`

Specifies the output format: `pdf` or `ps`. This argument is passed to the process command using `-s pdf`.

`%1`

Specifies the image formats other than GIF, JPEG and PNG natively supported by the XSL-FO processor for the chosen output format: `"|pdf"` or `"|eps|ps"`. This argument is passed to the process command using `-s "|pdf"`.

`%2`

Specifies the value of XSLT style sheet parameter `"generate.toc"` (see DocBook XSL Stylesheet Documentation). This argument is passed to the process command using `-s "/book toc"`.

`%3`

Specifies the value of XSLT style sheet parameter `"section.autolabel"` (see DocBook XSL Stylesheet Documentation). This argument is passed to the process command using `-s 1`.

`%4`

Specifies the output file URL. This argument is passed to the process command using `-u docs/commands/commands.pdf`.

3.3. Example 3: Convert a DocBook document to JavaHelp™

Convert DocBook document `help.xml` to JavaHelp™ created in directory `xxe_help/`.

```
$ XXE_CONFIG="file://`pwd`/customize.xxe;+" \  
  convertdoc docb.toJavaHelp help.xml -u xxe_help
```

Note that the DocBook configuration bundled with XXE does not include a process command called `docb.toJavaHelp`, which allows to convert DocBook to JavaHelp™. That's why this process command has been added in a separate file called `customize.xxe`.

Chapter 6. Commands written in the Java™ programming language

In the following command reference:

selected node

means

- the explicitly selected single node;
- OR the node (text, comment, processing-instruction or element) containing the caret, if there is no explicit node selection and if the `[implicitNode]` option is used in the parameter of the command;
- OR the element containing the textual node (text, comment, processing-instruction) containing the caret, if there is no explicit node selection and if the `[implicitElement]` option is used in the parameter of the command.

selected nodes

means

- the explicitly selected single node or node range;
- OR the node (text, comment, processing-instruction or element) containing the caret, if there is no explicit node selection and if the `[implicitNode]` option is used in the parameter of the command;
- OR the element containing the textual node (text, comment, processing-instruction) containing the caret, if there is no explicit node selection and if the `[implicitElement]` option is used in the parameter of the command.

argument node

means

- an empty text node, if the parameter of the command ends with `#text`;
- OR an automatically generated empty element (see configuration element `newElementContent` in *XMLmind XML Editor - Configuration and Deployment*), if the parameter of the command ends with an element name;
- OR a copy of an element template (see configuration element `elementTemplate` in *XMLmind XML Editor - Configuration and Deployment*), if the parameter of the command ends with an element template name.

If the argument node is not explicitly specified in the parameter of a command, a dialog box is displayed and the user will have to interactively specify it.

Note that namespace prefixes cannot be used inside the parameter of a command. Notation `{namespace_URI}local_name` must be used instead.

Example 1: `{http://www.w3.org/1999/xhtml}p` means `p` in the `http://www.w3.org/1999/xhtml` namespace.

Example 2: `p` means `p` with no namespace.

These non-terminals are sometimes used in the synopsis of a parameter of a command:

```
implicit_selection -> '[implicitNode]' | '[implicitElement]'  
argument_node -> '#text' |  
                element_name |  
                '#template(' element_name ',' template_name ')'
```

```
element_name -> Name | '{' namespace_URI '}' NCName
namespace_uri -> anyURI
```

In the synopsis of a parameter of a command, *s* means space.

Note that *whitespace is not allowed inside the #template() construct*. That is, "#template(figure, image)" will not work while "#template(figure,image)" will work.

1. alert

Parameter syntax:

```
message
```

Displays an information dialog box containing the message specified by command parameter.

This command is useful to debug macro-commands.

Example:

```
alert Hello, world!
```

2. add

Parameter syntax:

```
'before'|'after' [ implicit_selection ]? S [ argument_node ]?
```

Adds argument node before or after selected node. If the grammar forbids to do so, tries the same operation with the parent of selected node. If the grammar forbids to do so, tries the same operation with the grand-parent of selected node and so on.

Examples:

```
add after[implicitElement] para
add before #template(figure,image)
```

See also `addBlockInFlow`.

3. addAttribute

Parameter syntax:

```
[ '[implicitElement]' ]? [ '[empty]'|'[none]'|'[id]' ]? attribute_name [ attribute_value ]?
```

This command is only useful to write macro commands.

Adds attribute *attribute_name* in explicitly or implicitly selected element if grammar constraining the document allows to do so.

This command is similar to `putAttribute` except that it will not replace the attribute if it already exists.

Examples:

```
addAttribute [implicitElement] cols
addAttribute linkend "chapter 1"
addAttribute [dummy] cols
addAttribute [implicitElement] [dummy] cols
```

4. addBlockInFlow

Parameter syntax:

```
'[ 'inline_container_element_name' ]' S block_element_name
```

Intelligently adds specified block element after the text node containing the caret or after the explicitly selected element.

`block_element_name`

Specifies the element to be inserted.

`inline_container_element_name`

Specifies an element which can contain a mix of text and inline elements. XHTML example: `p`. DocBook example: `simpara` (but not `para` which can also contain blocks). This element is needed to teach to the command which are the inline elements of the document type.

Initially, this command has been designed to deal with XHTML elements such as `li`, `dd`, `th`, `td`, `div`, which not only can contain blocks (`p`, `ul`, `table`, etc), but can also contain a mix of text and inline elements (`b`, `i`, `em`, `a`, etc). This kind of content model is called a *flow*.

XHTML example:

```
<ul>
    +--- caret is here
    |
    v
  <li><b>First</b> item.</li>
  <li><b>Second</b> item.</li>
</ul>
```

Generic command `"add after[implicitElement] pre"` gives:

```
<ul>
  <li><b>First</b> item.</li>
  <li><b>Second</b> item.</li>
</ul>
<pre>|</pre>
```

Smarter command `"addBlockInFlow [p] pre"1` gives:

```
<ul>
  <li><b>First</b> item.
    <pre>|</pre>
  </li>
  <li><b>Second</b> item.</li>
</ul>
```

5. beep

No parameter.

Emits an audio beep.

This command is useful to write macro-commands.

6. cancelSelection

No parameter.

¹File `xhtml.jar` contains command `xhtml.addBlock."addBlockInFlow [p] pre"` is strictly equivalent to `"xhtml.addBlock pre"`.

Cancels text or node selection if any.

7. center

No parameter.

Centers node selection, text selection or caret in the document view.

8. checkValidity

No parameter.

Checks the validity of the document and, if validity errors are found, displays a modal dialog box (similar to the Validity tool) listing them. If there is no validity error, this command just displays an OK message.

This command has been added mainly to make it easier building simple XML editors using XXE components (that is, not XXE itself).

9. confirm

Parameter syntax:

```
message
```

Displays a dialog box containing the message specified by command parameter and requesting the user to confirm an action. If the user clicks on the OK button, the action is to be performed. If the user clicks on the Cancel button, the action is to be canceled.

This command is useful when writing interactive macro-commands.

Example:

```
confirm Convert selected text to computeroutput?
```

10. convert

Parameter syntax:

```
[ implicit_selection ]? [ S '#text' | element_name ]?
```

Converts text selection or selected nodes to argument node².

Unlike `replace` which creates an *empty* new element, `convert` transfers the content of the selection to the new element which is the result of the conversion.

More precisely, in the case of node selection:

- When a single element is selected, all its children (but not its attributes) are transferred to the result of the conversion.

Example:

```
"<simpara>the <emphasis>little</emphasis> girl.</simpara>"
```

converted to <para> gives

```
"<para>the <emphasis>little</emphasis> girl.</para>".
```

²Note that, unlike commands `insert`, `replace`, `add`, a `#template()` argument is not supported by command `convert`.

- When several nodes or a single non-element node are selected, all these nodes are given a new parent element which is the result of the conversion.

Example:

```
"<simpara>Once upon a time.</simpara>"
```

plus

```
"<simpara>the <emphasis>little</emphasis> girl.</simpara>"
```

can be converted to `<blockquote>` and that gives us

```
"<blockquote><simpara>Once upon a time.</simpara><simpara>the <emphasis>little</emphasis> girl.</simpara></blockquote>"
```

Examples:

```
convert emphasis
convert [implicitElement] #text
```

See also command `wrap`, a variant of command `convert` which has a different behavior in the case of single element selection.

11. convertCase

Parameter syntax:

```
lower | upper | capital
```

Converts case of text selection or all text contained in selected node. If there is no selection, converts case from caret position to end of word.

Note that in XXE, a word is ended by a XML space character or by end of textual node whichever comes first.

lower

characters are converted to lower-case characters.

upper

characters are converted to upper-case characters.

capital

first character of a word is converted to an upper-case character.

12. copy

Parameter syntax:

```
[ implicit_selection ]?
```

Copies text selection or selected nodes to system clipboard.

Example:

```
copy [implicitElement]
```

13. copyAsInclusion

Parameter syntax:

```
[ '[implicitElement]' ]?
```

Copies explicitly selected element (or implicitly selected element if the `[implicitElement]` option is used) to system clipboard, marking clipboard content as being a reference to an element rather than ordinary data.

`CopyAsInclusion` cannot be executed when the selected element is *inside* a reference to an element. That is, the reference itself can be copied, but not any of its descendant elements.

Example:

```
copyAsInclusion [implicitElement]
```

See also command `pasteInclusionEnabled`, a variant of command `paste`, which can be used to paste the content of the clipboard as a reference rather than as data, when special attributes have been added to the clipboard content by `copyAsInclusion`.

Commands `copyAsInclusion` (generally bound to keystroke **Shift-Ctrl-C**) and `pasteInclusionEnabled` (generally bound to keystroke **Ctrl-V**) are used to compose *modular documents*, see the corresponding tutorial section in the User's Guide.

14. copyChars

Parameter syntax:

```
[ implicit_selection ]? [ '[separateParagraphs]' | '[separateNodes]' ]?
```

Copies characters found in text selection or in selected nodes to system clipboard. Unlike command `copy`, this command only copies *characters*.

By default, characters coming from different textual nodes (i.e. text, comment, PI) are simply concatenated. The following options allow to change this behavior.

`[separateParagraphs]`

Automatically add a line separator after the characters of each copied ``paragraph''.

However, this automatic detection of paragraphs is easily puzzled by content models such as XHTML 1.1, `div`, `td` (``flows'').

`[separateNodes]`

Automatically add a line separator after the characters of each copied textual node.

Examples:

```
copyChars [implicitElement]
copyChars [implicitNode]
copyChars [implicitElement][separateParagraphs]
copyChars [separateNodes]
```

15. cut

Parameter syntax:

```
[ implicit_selection ]?
```

Cuts text selection or selected nodes to system clipboard.

Example:

```
cut [implicitElement]
```

16. decrSectionLevel

Parameter syntax:

```
[ section_name | '-' title_name para_name  
  [ sect1_name sect2_name ... sectN_name ]?  
 ]?
```

Decreases the level of explicitly selected sections by moving them out of their parent section.

This command, while designed for DocBook, can also be used with other document types provided that:

- The content model of sections is compatible with the following:

```
section = title, ((elements_including_para+, section*) | (section+))  
sectN = title, ((elements_including_para+, sectM*) | (sectM+))  
  where M=N+1
```

- Elements which have roles similar to DocBook's `section`, `title`, `para`, `sect1`, `sect2`, `sect3`, `sect4`, `sect5` have been specified in the parameter string.
 - Specify '-' if the document type does not support 'recursive sections' similar to DocBook `section`.
 - Do not specify `sect1_name`, `sect2_name`, etc, if the document type exclusively supports 'recursive sections'.
 - It is possible to specify has many fixed-level sections `sect1_name`, `sect2_name`, etc, as needed to.

See also `incrSectionLevel`, which is the opposite command.

Examples:

```
decrSectionLevel  
decrSectionLevel sect sectTitle p  
decrSectionLevel - title paragraph section1 section2 section3 section4 section5 section6
```

17. declareNamespace

No parameter.

Displays a modal dialog box (similar to the dialog displayed by Tools|Declare Namespace) which allows to declare new namespaces and/or change the prefixes of existing namespaces.

This command has been added mainly to make it easier building simple XML editors using XXE components (that is, not XXE itself).

18. delete

Parameter syntax:

```
[ 'force' ]? [ implicit_selection ]?
```

Deletes text selection or selected nodes.

Option `force` may be used to force the deletion even if the grammar constraining the document forbids to do so.

Example:

```
delete force[implicitElement]
```

19. deleteChar

Parameter syntax:

```
[ 'backwards' ]?
```

Deletes character following the caret in the textual node (text, comment, processing-instruction). If there is no such character, moves caret to following textual node.

If the `backwards` option is used, deletes character preceding the caret in the textual node. If there is no such character, moves caret to preceding textual node.

20. deleteSelectionOrDeleteChar

Parameter syntax:

```
[ 'backwards' ]?
```

If there is an explicit node selection, invokes command `delete` without any parameter, otherwise invokes command `deleteChar`, passing it its own parameter if any.

This command is intended to be bound to keys `Del` and `BackSpace`.

21. deleteSelectionOrJoinOrDeleteChar

Parameter syntax:

```
[ 'backwards' ]?
```

If there is an explicit node selection, invokes command `delete` without any parameter, otherwise invokes command `joinOrDeleteChar`, passing it its own parameter if any.

This command is intended to be bound to keys `Del` and `BackSpace`.

22. editAttributes

Parameter syntax:

```
[ '[implicitElement]' ]?
```

Displays a modal dialog box (similar to the `Attributes` tool) which allows to edit the attributes of selected element.

This command has been added mainly to make it easier building simple XML editors using `XXE` components (that is, not `XXE` itself).

23. editMenu

Parameter syntax:

```
[ 'select' ]?
```

Displays `Edit` popup menu.

If the `select` option is used and if there is no text or node selection, the element clicked upon is selected before the popup menu is displayed.

24. editPITarget

Parameter syntax:

```
[ '[implicitNode]' ]? [ S target ]?
```

If `target` is specified, changes the target of the explicitly or implicitly selected processing instruction to `target`.

Otherwise displays a dialog box that can be used to interactively specify a new target for the explicitly or implicitly selected processing instruction.

Examples:

```
editPITarget [implicitNode]
editPITarget php
editPITarget [implicitNode] php
```

25. execute

No parameter.

This command is mainly used to interactively test other commands.

Displays a dialog box containing a text field where the user can enter the name of a command to be executed, possibly followed by a parameter.

Returns result of executed command if any.

26. extractImage

Parameter:

```
[attribute_name|'-' ]? ['anyURI'|'hexBinary'|'base64Binary'|'XML'|'-' ]?
[file_name]?
```

This command is the opposite of `setImage`. It can be used to save to disk the image represented by explicitly selected element. See also `extractObject`.

`attribute_name`

This parameter specifies the name of the attribute containing the URL of the image file or directly containing the image data encoded in `hexBinary` or in `base64Binary`.

If this parameter is absent (or is `'-'`), it is the selected element itself which contains the image URL or which directly contains the image data in `hexBinary`, in `base64Binary` or in XML formats.

`anyURI, hexBinary, base64Binary, XML`

Specifies how the image is ``stored'' in the element or in the attribute. Data type `'XML'` is only allowed for elements (typically an `svg:svg` element).

If this parameter is absent (or is `'-'`), the data type is found using the grammar of the document. Of course, this cannot be guessed for documents conforming to a DTD (too weakly typed) and for *invalid* documents conforming to a W3C XML Schema.

`file_name`

Specifies the name of the file created by this command.

`'%T'` specifies a temporary file name automatically generated by this command.

If specified file name ends with `'.%X'`, this string is replaced by a suffix corresponding to the format of the image. For example, this command can detect that the data compressed with `gzip` before being encoded in `base64Binary` is in fact GIF image data and in such case, it will replace `'.%X'` by `'.gif'`.

If this parameter is absent, a chooser dialog box is displayed to let the user specify where the image file is to be created.

This command returns the name of the image file it has created.

Examples:

```
extractImage
extractImage fileref anyURI
extractImage -
```

```
extractImage data - /tmp/extracted.%X  
extractImage - XML %T
```

27. extractObject

Identical to `extractImage` except that the messages which are displayed by this command are more generic: instead of using the term `image`, they use the term `object`.

This command has been added because command `extractImage` can be used for objects other than images: audio, video, measurement data, digital signatures, etc.

28. include

Parameter syntax:

```
'into' | ('replace'|'before'|'after' [ implicit_selection ]?)  
[ '['useEntity'|'useXInclude'|'chooseBest' ]? [ S url ]?
```

`into`

Pastes a reference to an element into element containing caret, at caret position.

`replace`

Pastes a reference to an element replacing text selection or selected nodes..

`before or after`

Pastes a reference to an element before of after selected nodes.

`URL` specifies which document (that is, which root element) to reference. An element deep inside the document may be referenced by adding an XPointer fragment to the URL.

A relative URL is relative to the location of the document being edited.

If `URL` is not specified, a specialized dialog box is displayed allowing the user to specify it, possibly with an XPointer fragment.

The mechanism used to paste a reference is normally specified by the Options dialog box in the Edit tab. Options `useEntity`, `useXInclude`, `chooseBest` may be used to override the user preferences specified there:

`useEntity`

Always use the reference to an external entity mechanism (for example, to ease the interchange of the documents you create with XXE).

`useXInclude`

Always use the XInclude mechanism. See <http://www.w3.org/TR/xinclude/>.

`chooseBest`

Use the reference to an external entity mechanism when possible (example: the referencing document has a `<!DOCTYPE>` and no XPointer is being used) and the XInclude mechanism otherwise (example: the referencing document does not have `<!DOCTYPE>` or a XPointer is needed).

Examples:

```
include into  
include after[implicitElement]  
include after[implicitElement][useEntity]  
include replace[useXInclude] ../common/Copyright.html  
include before http://www.numbers.com/england/numbers.html#vat
```

Commands `copyAsInclusion` (generally bound to keystroke **Shift-Ctrl-C**) and `pasteInclusionEnabled` (generally bound to keystroke **Ctrl-V**) are used to compose *modular documents*, see the corresponding tutorial section in the User's Guide.

Command `include` can be used to implement an alternative user interface for composing modular documents and may be also useful in macro-commands.

29. `incrSectionLevel`

Parameter syntax:

```
[ section_name | '-' title_name para_name  
  [ sect1_name sect2_name ... sectN_name ]?  
 ]?
```

Increases the level of explicitly selected sections by giving them a new parent section.

See also `decrSectionLevel`, which is the opposite command, for a description of the parameter string.

Examples:

```
incrSectionLevel  
incrSectionLevel sect sectTitle p  
incrSectionLevel - title paragraph section1 section2 section3 section4 section5 section6
```

30. `insert`

Parameter syntax:

```
'into' | ('before' | 'after' [ implicit_selection ]?) S [ argument_node ]?
```

If the `into` option is used, inserts argument node into:

- explicitly selected *empty* element
- OR element containing caret, at caret position.

If the `before` or `after` options are used, inserts argument node before or after selected nodes.

Examples:

```
insert into  
insert into ulink  
insert before[implicitElement]  
insert after[implicitElement] #template(table,simple)
```

31. `insertCharByName`

Parameter syntax:

```
[ '[DocBook]' ]? [ S entity_name ]?
```

Inserts at caret position a character specified using its entity name.

If entity name is not specified, this command displays a dialog box (supports auto-completion) which allows to choose the entity name interactively.

If the `[DocBook]` option is specified, the set of character entities used by this command is always the set defined in the DocBook 4.2 DTD, whatever is the DTD or XML-Schema of the document being edited.

Unless the `[DocBook]` option is specified, this command cannot be executed if the document being edited has no DTD defining character entities.

Examples:

```
insertCharByName  
insertCharByName ndash
```

```
insertCharByName [DocBook]
insertCharByName [DocBook] lambda
```

32. insertControlChar

Parameter syntax:

```
control_character
```

Inserts specified control character (newline, tab, etc) at caret position. The control character can be specified using its Java™ notation, for example: "\n" or "\u000a" for the newline character.

This command will not work if the view of the element in which the control character is to be inserted *rejects* such characters.

- In the tree view, only views of elements having `xml:space=preserve` accept control characters.
- In the styled view, only views of elements having CSS property `"white-space: pre;"` accept control characters.

Note that pasting control characters using the paste command always work.

33. insertControlCharOrSplit

Parameter syntax:

```
control_character
```

If caret is inside a paragraph-like element at any nesting level, splits this paragraph-like element in two parts at caret position.

Otherwise works like `insertControlChar`.

A paragraph-like element is any type of element having no required attributes and containing text and possibly child elements but in any order and in any number.

34. insertNode

Parameter syntax:

```
'commentInto' | 'piInto' | 'textInto' |
('commentBefore' | 'piBefore' | 'textBefore' | 'sameElementBefore' |
'commentAfter' | 'piAfter' | 'textAfter' | 'sameElementAfter' [ implicit_selection ]?)
[ pi_target ]?
```

If option ends with `Into`, inserts node specified by beginning of option (`comment`, `pi`, `text`, `sameElement`) `into`:

- explicitly selected *empty* element
- OR element containing caret, at caret position.

If option ends with `Before` or `After`, inserts node specified by beginning of option (`comment`, `pi`, `text`, `sameElement`) before or after selected node.

`pi_target` may be used to specify the target of the processing instruction to be inserted (options `piInto`, `piBefore` or `piAfter`). By default, this target is placeholder string `"target"`, which can be modified using command `edit-PITarget`.

`pi_target` is ignored for node types other than processing instructions.

Examples:

```
insertNode textBefore[implicitElement]
insertNode textInto
insertNode sameElementAfter[implicitElement]
insertNode piInto
insertNode piAfter[implicitNode] php
```

35. insertSpecialChars

Parameter syntax:

```
[ hexadecimal, octal or decimal code of first character displayed by a 256-character palette ]?
```

Inserts one or more "special characters" at caret position. These characters are selected using a modal dialog box which is similar to the Characters tool of XMLmind XML Editor.

If a Unicode code has been specified in the parameter, the dialog box displays a 256-character palette starting at this character; otherwise the dialog box displays the last character palette chosen by the user.

Examples: 9984 is first Dingbats character in decimal notation, 023400 is same character in octal notation (must start with "0"), 0x2700 is same character in hexadecimal notation (must start with "0x").

```
insertSpecialChars
insertSpecialChars 0x2700
insertSpecialChars 023400
insertSpecialChars 9984
```

This command has been added mainly to allow simple XML editors built using XXE components (that is, not XXE itself) to have the same facilities than XXE.

36. insertString

Parameter syntax:

```
string
```

Inserts specified string at caret position.

37. join

Parameter syntax:

```
[ 'after' ]? [ '[implicitElement]' ]?
```

Joins explicitly or implicitly selected element to its preceding sibling, an element of same type. This gives a single element containing the child nodes of the two joined elements.

If the `after` option is used, joins explicitly or implicitly selected element to its following sibling, an element of same type.

This command is the inverse command of `split`.

Examples:

```
join after
join [implicitElement]
join after[implicitElement]
```

38. joinOrDeleteChar

Parameter syntax:

```
[ 'backwards' ]?
```

If caret is inside a paragraph-like element at any nesting level and if caret is located after the last character of this element, joins this paragraph-like element to the following paragraph-like element of the same type.

If `backwards` option is used, if caret is inside a paragraph-like element at any nesting level and if caret is located before the first character of this element, joins this paragraph-like element to the preceding paragraph-like element of the same type

Otherwise works like `deleteChar`.

A paragraph-like element is any type of element having no required attributes and containing text and possibly child elements but in any order and in any number.

39. listBindings

No parameter.

Displays a dialog box (similar to the dialog displayed by `Help|Mouse` and `Key Bindings`) containing the mouse and key bindings that can be used in current document view.

This command has been added mainly to make it easier building simple XML editors using XXE components (that is, not XXE itself).

40. makeParagraphs

Parameter syntax:

```
[ '[blocks]' ]? [ '[systemSelection]' ]? element_name
```

This command just returns a string and therefore, is useful only inside a macro-command.

Read text lines from the clipboard. For each text line, creates an element having specified name containing the text line. Returns an XML string containing the list of elements.

Example, if *element_name* is `para` and if the clipboard contains:

```
word1, word1, word1.  
word2, word2, word2.  
  
word3, word3, word3.  
word4, word4, word4.
```

then the command returns:

```
<?xml version="1.0"?>  
<ns:clipboard  
xmlns:ns="http://www.xmlmind.com/xmleditor/namespace/clipboard"  
><para  
>word1, word1, word1.</para  
><para  
>word2, word2, word2.</para  
><para  
>word3, word3, word3.</para  
><para  
>word4, word4, word4.</para  
></ns:clipboard  
>
```

`systemSelection`

Read text lines from the system selection (only on Unix/X11) rather than from the clipboard.

`blocks`

Forces the command to convert multiple text lines separated by open lines to a single element. Without this option, each non-empty line is converted to an element.

Example, when this option is used, if *element_name* is *para* and if the clipboard contains:

```
word1, word1, word1.  
word2, word2, word2.  
  
word3, word3, word3.  
word4, word4, word4.
```

then the command returns:

```
<?xml version="1.0"?>  
<ns:clipboard  
  xmlns:ns="http://www.xmlmind.com/xmleditor/namespace/clipboard"  
><para  
>word1, word1, word1. word2, word2, word2.</para  
><para  
>word3, word3, word3. word4, word4, word4.</para  
></ns:clipboard  
>
```

Syntax examples:

```
makeParagraphs p  
makeParagraphs [blocks] simpara  
makeParagraphs [blocks][systemSelection] {http://www.foo.com/schema/bar}paragraph
```

DocBook example:

```
<command name="insertAfterAsParagraphs">  
  <macro>  
    <sequence>  
      <command name="makeParagraphs" parameter="%0" />  
      <command name="paste" parameter="after[implicitElement] %_" />  
    </sequence>  
  </macro>  
</command>  
  
<binding>  
  <keyPressed code="ESCAPE" />  
  <charTyped char="w" />  
  <command name="insertAfterAsParagraphs" parameter="para" />  
</binding>
```

41. moveDotTo

Parameter syntax:

```
'previousChar' | 'nextChar' | 'previousWord' | 'nextWord' |  
'previousTextNode' | 'nextTextNode' | 'previousElement' |  
'nextElement' | 'textNodeBegin' | 'textNodeEnd' |  
'elementBegin' | 'elementEnd' | 'documentBegin' |  
'documentEnd' | 'lineBegin' | 'lineEnd' | 'previousLine' |  
'nextLine' | 'wordBegin' | 'wordEnd'
```

Moves caret to specified location.

42. paste

Parameter syntax:

```
'into' | 'toOrInto' | ('to' | 'before' | 'after' [ implicit_selection ]?)  
([ S string ]? | [ '[systemSelection]' ]?)
```

into

Pastes the content of system clipboard into element containing caret, at caret position.

toOrInto

Pastes the content of system clipboard replacing text selection or selected nodes.

OR if there is no explicit selection, pastes the content of system clipboard into element containing caret, at caret position.

to

Pastes the content of system clipboard replacing text selection or selected nodes.

before or after

Pastes the content of system clipboard before of after selected nodes.

The system clipboard may contain one or several nodes or just plain text. The content of system clipboard, is parsed as XML if it begins with "<?xml" otherwise it is considered to be plain text.

If several nodes are to be pasted, they must be wrapped in a `{http://www.xmlmind.com/xmleditor/namespace/clipboard}clipboard` element. See last example below.

If *string* is specified in the command parameter, this string is used instead of the content of system clipboard.

If the `[systemSelection]` option is used, the content of system selection is used instead of the content of system clipboard.

Examples:

```
paste toOrInto
paste toOrInto[systemSelection]
paste before[implicitElement]
paste before[implicitElement][systemSelection]
paste after <?xml version='1.0'?><p>A paragraph.</p>

# Whitespace and newlines have been added to improve readability.
# In reality, they are not allowed here.

paste into <?xml version="1.0"?>
  <ns:clipboard xmlns:ns="http://www.xmlmind.com/xmleditor/namespace/clipboard">
    A text line containing <b>bold</b> and <i>italic</i> text.
  </ns:clipboard>
```

43. pasteInclusionEnabled

This command is a variant of command `paste`, which can be used to paste the content of the clipboard as a reference rather than as data, when special attributes have been added to the clipboard content by `copyAsInclusion`.

A confirmation dialog box is displayed when the user attempts to paste a reference rather than data.

Pasting a reference to another part of the same document is not possible and in such case, an error is reported to the user.

Unless special attributes have been added by `copyAsInclusion` to the clipboard content, this command is identical to command `paste`.

Commands `copyAsInclusion` (generally bound to keystroke **Shift-Ctrl-C**) and `pasteInclusionEnabled` (generally bound to keystroke **Ctrl-V**) are used to compose *modular documents*, see the corresponding tutorial section in the User's Guide.

44. pasteSystemSelection

No parameter.

Equivalent to `"paste into[systemSelection]"` after moving the caret to the text location clicked upon.

45. pick

Parameter syntax:

```
title S 'false' [ S item ]+
OR title S 'true' [ S label S item ]+
OR title S 'true' | 'false' S '@' S URL_or_file_name S encoding | 'default'
```

This command is only useful to write interactive macro commands.

Displays a dialog box with title *title* containing a list of strings. User clicks on a string to select it. Returns selected string.

If second field in the command parameter is *false*, the list of strings displayed by the dialog box is [*item*]+.

If second field in the command parameter is *true*, the list of strings displayed by the dialog box is [*label*]+ but when the user chooses a label, it is the item which follows it in the command parameter which is returned by this command.

If the third field is character '@', the labels and/or the items are loaded from text file specified by *URL_or_file_name*. This file contains labels and/or items separated by newlines ('\n', '\r', or '\r\n'). Open lines are ignored.

The encoding of this text file is specified by *encoding*. If *encoding* is specified as *default*, the encoding of the text file is the native encoding of the platform, for example Windows-1252 on an US Windows machine.

Examples:

```
pick 'Pick a number' false 1 2 3 4 5
pick "Pick a number" true "One" 1 "Two" 2 "Three" 3 "Four" 4 "Five" 5
pick 'Pick a number' false @ "C:\temp\number_list1.txt" default
pick 'Pick a number' true @ file:///tmp/number_list2.txt ISO-8859-1
```

46. prompt

Parameter syntax:

```
title message [ suggested_value ]?
```

This command is only useful to write interactive macro commands.

Displays a dialog box with title *title* asking the user to answer question *message* by typing a string in a text field. Returns typed string.

If *suggested_value* is specified, the text field is initialized with this value.

Examples:

```
prompt Question "Number of columns:"
prompt Question "Text align:" left
```

47. putAttribute

Parameter syntax:

```
[ '[implicitElement]' ]? [ '[empty]' | '[none]' | '[id]' ]? attribute_name [ attribute_value ]?
```

This command is only useful to write macro commands.

Adds or replaces attribute *attribute_name* in explicitly or implicitly selected element if grammar constraining the document allows to do so.

- If attribute value *attribute_value* is specified then this value is used as the new value of attribute *attribute_name* (this value is checked for validity),
- otherwise
 - If `[empty]` has been specified, sets the attribute to the empty string (without checking if it is a valid value).
 - If `[dummy]` has been specified, sets the attribute to string "???" (without checking if it is a valid value).
 - If `[id]` has been specified, sets the attribute to an automatically generated id (without checking if it is a valid value).
 - otherwise, a dialog box is displayed to let user interactively specify a value (this value is checked for validity).

Examples:

```
putAttribute [implicitElement] cols
putAttribute linkend "chapter 1"
putAttribute [dummy] cols
putAttribute [implicitElement] [dummy] cols
```

48. recordMacro

Parameter syntax:

```
'start' | 'stop' | 'toggle' | 'cancel' | 'view' | 'replay'
```

This command allows to record a sequence of commands and to replay the recorded sequence at will. Used in conjunction with commands such as `search` and `xpathSearch`, this command may be seen as an advanced, versatile, yet easy to use, search/replace facility. See also command `confirm`.

start

Starts recording a sequence of commands.

stop

Stops recording the sequence of commands.

toggle

If the recording of a sequence of commands has been started, stops this recording. Otherwise, starts recording a sequence of commands.

cancel

Cancels the recording of a sequence of commands.

view

Displays a dialog box containing recorded macro in XML form. Very handy to paste it in an XXE configuration file (see XMLmind XML Editor - Configuration and Deployment).

replay

Replays recorded sequence of commands.

At most 20 commands can be recorded. Typing contiguous characters, no matter how many, counts as a single command (`insertString`).

Attempting to record the following commands will automatically cause macro recording to be canceled:

- any command which has been designed to be bound to a mouse click (e.g. `selectAt`),
- `undo`, `redo`, `repeat`,
- any command which fails (example: searching a string and this string is not found),

- any command which cannot be executed given current editing context (most obvious example: "recordMacro replay"; other example: pasting some text to a place where the schema forbids to do so).

Recording interactive command such as "insert after" works as expected: it is the command *along with the element interactively chosen by the user* which is recorded, and not the interactive invocation of "insert after" (i.e. which displays a dialog box).

Recording command execute is fully supported and works as expected: it is the command executed by `execute` which is recorded, and not `execute` itself.

Examples:

```
recordMacro start
recordMacro stop
recordMacro replay
```

49. redo

No parameter.

Redo last undone command.

50. refresh

Parameter syntax:

```
'refresh' | 'rebuild' [ '['implicitNode'|'implicitElement'|'implicitDocument'] ]?
```

Refreshes or rebuilds selected node.

Refresh means: relayout and repaint the view of the selected node.

Rebuild means: recreate the view of the selected node.

If the `implicitDocument` option is used and if there is no explicit node selection, the entire document is refreshed or rebuilt.

Examples:

```
refresh refresh
refresh rebuild[implicitDocument]
```

51. removeAttribute

Parameter syntax:

```
[ '['force'] ]? [ '['implicitElement'] ]? attribute_name
```

This command is only useful to write macro commands.

Removes attribute `attribute_name` in explicitly or implicitly selected element if the grammar constraining the document allows to do so.

Option `[force]` may be used to remove specified attribute even if the grammar constraining the document does not allow to do so.

Examples:

```
removeAttribute [implicitElement] cols
removeAttribute role
removeAttribute [force] linkend
removeAttribute [force] [implicitElement] linkend
```

52. repeat

Parameter syntax:

```
[ index_in_command_history ]?
```

Repeats last repeatable command.

Currently only commands requiring the user to interactively specify an argument are repeatable.

Returns result of repeated command if any.

53. replace

Parameter syntax:

```
[ implicit_selection ]? [ S argument_node ]?
```

Replaces selected nodes with argument node.

Examples:

```
replace [implicitElement]
replace {http://www.xmlmind.com/xmlmind/schema/configuration}newElementTemplate
```

54. resizeSVG

No parameters.

Displays a specialized dialog box which can be used to resize the topmost `svg:svg` ancestor of implicitly or explicitly selected element.

55. run

Parameter syntax:

```
[ command_line ]?
```

Executes external command line specified by its parameter. If no parameter is specified, prompts user to input a command line.

Returns output of executed command line (that is, what is printed on `stdout` in C/Unix parlance).

The command line is executed using `/bin/sh` on Unix and using `cmd.exe` on Windows (this means that `run` will not work on Windows 9x or Windows ME).

Command line may contain variables which are substituted with their values prior to command execution:

%F

File name of a temporary file containing a copy of the selection. This temporary file is created in the same directory than the directory containing the document being edited.

- If a single element is selected, this element is saved in a DTD-less XML file, having a `.xml` extension, encoded using UTF-8.
- If several nodes are selected, the parent element of these nodes are saved in a DTD-less XML file, having a `.xml` extension, encoded using UTF-8.
- If there is a text selection or a single textual node is selected, the selected text is saved in a text file, having a `.txt` extension, encoded using the native encoding of the platform.

- If there is no explicit selection, the whole document is saved in a XML file, possibly having a DTD or XML-Schema, having same extension than document being edited, encoded using UTF-8.

%f

Same as %F except that it is a file: URL.

%d

URL of the document being edited.

%D

File name of the document being edited.

If this variable needs to be substituted and if document being edited is not stored on the local file system (example: `http://dav.acme.com/docs/mydoc.xml`), command run cannot be executed.

The "%" character can be escaped using "%%". The above variables can be specified as %`{F}`, %`{f}`, %`{d}`, %`{D}` if it helps.

Examples:

```
run date
run expand %F
run emacs "%D"
run "C:\Program Files\Info ZIP\zip.exe" -r all.zip "C:\temp\misc"
```

56. search

Parameter syntax:

```
[ '[' 'i'? 'w'? 'r'? 's'? 'b'? 'x'? ']' S ]? searched_text
```

Searches specified text from caret position to end of document, or if the `b` (Backwards) option has been specified, from caret position to beginning of document.

If searched text is not specified, displays a dialog box allowing to specify this text and any of the options.

Searched string, if specified, does not need to be quoted, even if it includes white space. However, beginning and trailing whitespace is removed from searched string before the command is executed. Therefore, the only way to search text starting and/or ending with whitespace is to quote (using single quotes or double quotes) the searched string.

Options (order of option letters is *not* important):

`i` (Ignore case)

The search is case-insensitive. Example: "foo" matches both "foo" and "Foo".

`w` (Whole word)

The found string must be a word, that is, the found string must be surrounded by white spaces. Example: "foo" matches "foo" but not "foobar".

`r` (Regular expression)

The searched string must be a valid regular expression. A regular expression is specified in a syntax similar to that used by Perl. See also <http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html>.

`s` (Smart mode)

Searching string "Hello world!" in an XML document is not as obvious as it seems: for example, is "Hello world!" with word "Hello" contained in an emphasis element followed by text node " world!" supposed to be found by `XXE`?

- If this option is selected, the answer is yes. "Hello world!" is found within "`Hello world!`" but not within "`<p>Hello </p><p>world!</p>`".

This mode uses the grammar constraining current document to recognize logically contiguous text across different types of elements.

- If this option is not selected, the answer is no. Each text node is separated from other text nodes whatever the type of the element containing it.

b (Backwards)

The document is searched from caret position to the end of the document.

x (eXtend selection)

Extends text selection from caret position to end of found text. If the `b` (Backwards) option has also been specified, extends text selection from caret position to beginning of found text.

This option is very useful when command `search` is used in macro-commands.

Examples:

```
search
search [xwi]
search XMLmind XML Editor
search [r] X\w+d
search "XMLmind "
search ' XML'
```

See also `searchReplace` and `xpathSearch`.

57. searchReplace

Parameter syntax:

```
[ search|searchBackwards|replace|replaceBackwards|
  findAgain|findAgainBackwards ]?
```

Displays a modal dialog box which allows to search and replace text in the document being edited (same as the Search tool in XMLmind XML Editor).

search

Displays the dialog box configured to be used for a text search from caret position to end of document.

searchBackwards

Displays the dialog box configured to be used for a text search from caret position to beginning of document.

replace

Displays the dialog box configured to be used for a search/replace operation from caret position to end of document.

replaceBackwards

Displays the dialog box configured to be used for a search/replace operation from caret position to beginning of document.

findAgain

The dialog box is not displayed. Repeats last text search from caret position to end of document.

findAgainBackwards

The dialog box is not displayed. Repeats last text search from caret position to beginning of document.

Without a parameter, the state of the dialog box is not changed. That is, the displayed dialog box is configured as it was the last time it has been used.

This command has been added mainly to allow simple XML editors built using XXE components (that is, not XXE itself) to have the same facilities than XXE.

58. selectAt

Parameter syntax:

```
[ 'begin' | 'extend' | 'end' ]?
```

begin

Cancels text or node selection if any. Moves caret to character clicked upon, if such character exists.

extend

If caret was moved by previous "selectAt begin", extends text selection to the character clicked upon.

end

If caret was moved by previous "selectAt begin", does nothing, otherwise selects node clicked upon.

Parameter is absent

Cancels text or node selection if any. Creates a new text selection:

- The beginning of the new text selection is the beginning of old text selection if any or caret position otherwise.
- The end of the new text selection is the character clicked upon.

59. selectBlockAtY

Parameter syntax:

```
[ 'orParent' ]?
```

``Selects the block" (paragraph, row, row group or table) on which the user has clicked. All graphical objects intersecting the Y coordinate of the mouse click are used to determine which block to select. The X coordinate of the mouse click is ignored.

This command can only be used in a styled view because a tree view does not contain ``blocks".

If the `orParent` option is specified, clicking again, exactly on the same place (i.e. *without moving the mouse at all*), selects the parent of the element selected by the previous invocation of this command.

See also `selectNodeAt`.

60. selectById

Parameter syntax:

```
('id' | 'reference' | 'nextReference' | 'previousReference' | 'swapIdAndReference') S [ id ]?
```

id

An element with an IDREF or IDREFS attribute must be implicitly or explicitly selected. Scrolls to and selects the element having this ID (if found).

reference

An element with an ID attribute must be implicitly or explicitly selected. Scrolls to and selects first element referencing this ID (if found).

nextReference

An element with an IDREF or IDREFS attribute must be implicitly or explicitly selected. Scrolls to and selects following element referencing this ID (if found).

previousReference

An element with an IDREF or IDREFS attribute must be implicitly or explicitly selected. Scrolls to and selects preceding element referencing this ID (if found).

swapIdAndReference

An element with an ID, IDREF or IDREFS attribute must be implicitly or explicitly selected. Scrolls to and selects opposite link end (if found).

Notes:

- An ID argument may be specified with options `id` and `reference` (but not with options `nextReference`, `previousReference` and `swapIdAndReference`). When this ID argument is specified (non interactive command typically used in a macro), there is no need for an element with an ID, IDREF or IDREFS attribute to be implicitly or explicitly selected.
- When implicitly or explicitly selected element has an IDREFS attribute containing several IDs, user is prompted to choose on of these IDs.

Examples:

```
selectById swapIdAndReference
selectById id introduction
```

61. selectFile

Parameter syntax:

```
[ 'openFile' | 'saveFile' | 'openDirectory' | 'saveDirectory' |
  'openFileURL' | 'saveFileURL' | 'openDirectoryURL' | 'saveDirectoryURL'
  [ URL_template ]? ]?
```

This command is only useful to write interactive macro commands.

Displays a file chooser dialog box that may be used to select a file or directory, local or remote, existing or to be created, depending on the first keyword in the parameter. By default, this file selection mode is `openFile` which specifies a local, existing, file.

When parameter is `openFile`, `saveFile`, `openDirectory` or `saveDirectory`, the standard file chooser dialog box is displayed and the command returns a file or directory name.

When parameter is `openFileURL`, `saveFileURL`, `openDirectoryURL` or `saveDirectoryURL`, an "advanced" file chooser dialog box is displayed and the command returns a file or directory URL.

The optional `URL_template` parameter is used to specify the directory initially displayed by the file chooser dialog box. When `saveXXX` options are used, this parameter is used, not only to specify initial directory, but also to suggest a basename for the save file.

See also command `selectConvertedFile`, which has been designed to be used in Convert macro-commands such as `docb.convertToHTML1`, `slides.convertToPS`, etc.

Examples:

```
selectFile
selectFile saveFile
selectFile openFileURL http://www.acme.com/doc/images/logo.gif
selectFile saveFileURL file:///tmp/article.pdf
```

62. selectConvertedFile

Variant of command `selectFile` specially designed to be used in Convert macro-commands such as `docb.convertToHTML1`, `slides.convertToPS`, etc.

Unlike command `selectFile`, this command is aware of the document being converted, which allows it to suggest smarter save file names/file URLs.

Moreover, this command supports two more "modes":

`saveFileWithExtension=`*file extension*, `saveFileURLWithExtension=`*file extension*

Identical to mode `saveFile` [`saveFileURL`], except that, when no `URL_template` has been specified,

- `saveFile` suggests the same file name/file URL as the document being converted, but with an "out" extension.
- `saveFileWithExtension=`*foo* suggests the same file name/file URL as the document being converted but with a "*foo*" extension.

File extension must not be empty, must not start with a '.' and must not contain spaces.

Examples:

```
selectConvertedFile
selectConvertedFile saveFile
selectConvertedFile saveFileWithExtension=word.xml
selectConvertedFile openFileURL http://www.acme.com/doc/images/logo.gif
selectConvertedFile saveFileURLWithExtension=ps file:///tmp/article.pdf
```

Note that when an `URL_template` has been specified, this `URL_template` is always suggested *as is* by the dialog box. For example, "`selectConvertedFile saveFileURLWithExtension=ps file:///tmp/article.pdf`" will suggest "`file:///tmp/article.pdf`" as a save file URL, and not "`file:///tmp/article.ps`".

63. selectPrinter

Parameter syntax:

```
[ format [ printer_name ]? ]?
```

This command is only useful to write interactive macro commands.

Displays a printer chooser dialog box that may be used to select a printer and configure the print job. This command does not work on a Mac with a Java™ 1.4 runtime.

format

Mime type of the document to be printed. Example: `application/vnd.hp-PC`.

The following short names are also supported: `ps` (`application/postscript`), `pdf` (`application/pdf`), `pcl` (`application/vnd.hp-PCL`).

printer_name

Name of a printer supporting format *format*. If this name is not specified, the dialog box displays the last selected printer supporting this format.

Without a parameter, the dialog box displays last selected PostScript® printer (that is, default parameter is "`ps`").

Examples:

```
selectPrinter
selectPrinter ps
selectPrinter "text/plain; charset=UTF-8"
selectPrinter application/postscript lp23
```

64. selectNode

Parameter syntax:

```
'parent' | 'child' |  
'firstChild' | 'lastChild' | 'children' |  
'previousSibling' | 'nextSibling' |  
'firstSibling' | 'lastSibling' |  
'ancestor' | 'ancestorOrSelf' | 'self' |  
'descendant' | 'descendantOrSelf' |  
'preceding' | 'precedingOrSelf' |  
'following' | 'followingOrSelf' |  
'extendToPreviousSibling' | 'extendToNextSibling'  
[ 'OrNone' | 'OrNode' | 'OrElement' ]?  
[ implicit_selection ]?  
S [ element_name | '#text' | '#comment' | '#processing-instruction' ]*
```

A number of keystrokes are bound to this command. This command is also needed to write non-trivial macro-commands.

parent

Selects parent of selected node.

child

Selects previously selected child of selected node.

If no child of selected node was previously selected, selects first child node of selected node.

firstChild

Selects first child node of selected node.

lastChild

Selects last child node of selected node.

children

Selects all the child nodes of selected node.

previousSibling

Selects preceding sibling of selected node.

nextSibling

Selects following sibling of selected node.

firstSibling

Selects first preceding sibling of selected node.

lastSibling

Selects last following sibling of selected node.

self

Selects selected node if selected node is an element, or parent element of selected node if selected node is a text. This option is mainly useful to test the name of implicitly or explicitly selected element.

ancestorOrSelf

Selects ancestor of selected node, starting at selected node. Searched ancestor is specified using a list of names. See below.

More precisely, lookup starts from selected node, if selected node is an element, or from parent element of selected node if selected node is a text, comment or processing-instruction node.

ancestor

Selects ancestor of selected node, starting at parent of selected node. Searched ancestor is specified using a list of names. See below.

More precisely, lookup starts from parent of selected node, if selected node is an element, or from grand-parent element of selected node if selected node is a text, comment or processing-instruction node.

Note that `selectNode.ancestor*` selects all the ancestors one after the other until it reaches the found ancestor. This is equivalent to interactively typing **Ctrl-Up** until the desired ancestor is selected. The idea behind that is to be able to use `selectNode ancestor` followed by `selectNode child` or `selectNode descendant*` in the same macro-command.

descendant

Selects previously selected descendant of selected node. Searched descendant is specified using a list of element names or node types. See below.

If no descendants of selected node were previously selected, searches a descendant node along the first child axis.

descendantOrSelf

Selects previously selected descendant of selected node. Searched descendant is specified using a list of element names or node types. See below.

If no descendants of selected node were previously selected, searches a descendant node along the first child axis.

Selected node itself can be explicitly selected if it corresponds to searched node.

precedingOrSelf

Selects preceding sibling of selected node, starting at selected node. Searched sibling is specified using a list of names. See below.

preceding

Selects preceding sibling of selected node, starting at sibling of selected node. Searched sibling is specified using a list of names. See below.

followingOrSelf

Selects following sibling of selected node, starting at selected node. Searched sibling is specified using a list of names. See below.

following

Selects following sibling of selected node, starting at sibling of selected node. Searched sibling is specified using a list of names. See below.

extendToPreviousSibling

Extends node selection to following sibling of last selected node.

extendToNextSibling

Extends node selection to preceding sibling of last selected node.

Examples:

```
selectNode childOrNone
selectNode parentOrNode
selectNode children
selectNode nextSibling[implicitElement]
selectNode self section
selectNode ancestorOrSelf[implicitElement] section sect5 sect4 sect3 sect2 sect1
selectNode descendant {http://www.xmlmind.com/xmleditor/schema/configuration}template \
{http://www.xmlmind.com/xmleditor/schema/configuration}css
selectNode extendToPreviousSibling
selectNode extendToNextSiblingOrElement
```

64.1. List of element names or node types

A list of element names or node types may be specified in order to conditionally perform a node selection.

Without this list, the specified `selectNode` command would select a node. Let's call it the *candidate* node.

The candidate node is tested against all items in the list, one after the other. If the candidate node matches any of these items, the candidate node is actually selected.

Element name

Candidate node must be an element having the same name.

#text

Candidate node must be a text node.

#comment

Candidate node must be a comment node.

#processing-instruction

Candidate node must be a processing instruction node.

Example 1: `selectNode child[implicitElement] para` selects first child of explicitly or implicitly selected element only if this child is a `para`.

Example 2: `selectNode.ancestor.itemizedlist orderedlist variablelist` selects first ancestor of explicitly selected element which is a list.

64.2. OrNone, OrNode, OrElement modifiers

The `OrNone`, `OrNode`, `OrElement` modifiers may be used to specify fallback behaviors for `selectNode` commands which otherwise would fail and therefore would do nothing at all.

`OrNone`

If specified `selectNode` command fails to select something new, current selection is canceled.

Example: let explicitly selected node be an empty element. In such case `selectNode child` fails and therefore, does nothing at all. But `selectNode childOrNone` succeeds and cancels current selection.

`OrNode`

If there is no explicit or implicit node selection to work with, command `selectNode` explicitly selects textual node containing caret.

Example: caret is contained in a `para` and there no explicit selection. In such case, `selectNode.parent` fails and therefore, does nothing at all. But `selectNode.parentOrNode` succeeds and selects the textual node containing the caret.

`OrElement`

If there is no explicit or implicit node selection to work with, command `selectNode` explicitly selects element containing caret.

Example: caret is contained in a `para` and there no explicit selection. In such case, `selectNode.parent` fails and therefore, does nothing at all. But `selectNode.parentOrElement` succeeds and selects the `para` containing the caret.

It does not make sense to use `OrNone`, `OrNode`, `OrElement` modifiers and `[implicitNode]`, `[implicitElement]` options in the same `selectNode` command. In such case, the `OrNone`, `OrNode`, `OrElement` modifiers are simply ignored.

65. selectNodeAt

Parameter syntax:

```
[ 'orParent' | 'extend' ]?
```

Selects the node clicked upon.

If the `orParent` option is specified, clicking again, exactly on the same place (i.e. *without moving the mouse at all*), selects the parent of the element selected by the previous invocation of this command.

If the `extend` option is used and if there is a node selection, this command extends the selected node range by adding sibling nodes to it. The location of the mouse click specifies how many sibling nodes are to be added.

See also `selectBlockAtY`.

66. selectText

Parameter syntax:

```
[ 'word' | 'line' | 'all' ]?
```

`word`

Selects the characters of the word containing caret.

`line`

Selects the text line containing caret.

`all`

Selects all the characters of the document.

Parameter is absent

Selects all the characters of the textual node (text, comment, processing instruction) containing caret.

67. selectTo

Parameter syntax:

```
'previousChar' | 'nextChar' | 'previousWord' | 'nextWord' |  
'previousTextNode' | 'nextTextNode' | 'previousElement' |  
'nextElement' | 'textNodeBegin' | 'textNodeEnd' |  
'elementBegin' | 'elementEnd' | 'documentBegin' |  
'documentEnd' | 'lineBegin' | 'lineEnd' | 'previousLine' |  
'nextLine' | 'wordBegin' | 'wordEnd'
```

Extends text selection to specified location.

68. setEditable

Parameter syntax:

```
[ 'implicitDocument' | 'implicitElement' ]? 'false' | 'true' | 'toggle'
```

Changes the read-only mark of an element or of the whole document.

- Without option `implicitDocument` or `implicitElement`, this command changes the read-only mark of the explicitly selected element.
- With option `implicitElement`, this command changes the read-only mark of the explicitly or implicitly selected element.
- With option `implicitDocument`, this command changes the read-only mark of the whole document unless an element is explicitly selected, in which case, it changes its read-only mark.

Parameter value:

`false`

Removes the read-only mark from selected node and all its descendants then marks the selected node as read-only.

true

Removes the read-only mark from selected node and all its descendants.

Note that this commands fails if selected node has an ancestor marked as read-only (because changing the read-only mark would not make selected node editable).

toggle

If selected node is editable, behaves like when parameter value is false. If selected node is not editable, behaves like when parameter value is true.

Example:

```
setEditable [implicitDocument] toggle
```

69. setImage

Parameter syntax:

```
[attribute_name|'-']? ['anyURI'|'hexBinary'|'base64Binary'|'XML'|'-']?  
['gzip'|'-']? [URL_or_file]?
```

Changes the image represented by explicitly selected element. See also setObject.

attribute_name

This parameter specifies the name of the attribute containing the URL of the image file or directly containing the image data encoded in 'hexBinary' or in 'base64Binary'.

If this parameter is absent (or is '-'), it is the selected element itself which contains the image URL or which directly contains the image data in 'hexBinary', in 'base64Binary' or in XML formats.

anyURI, hexBinary, base64Binary, XML

Specifies how the image is to be "stored" in the element or in the attribute. Data type 'XML' is only allowed for elements (typically an `svg:svg` element).

If this parameter is absent (or is '-'), the data type is found using the grammar of the document. Of course, this cannot be guessed for documents conforming to a DTD (too weakly typed) and for *invalid* documents conforming to a W3C XML Schema.

gzip

If this parameter is specified, image data is compressed with gzip before being encoded in 'hexBinary' or in 'base64Binary'.

This parameter is ignored for 'anyURI' and 'XML' data types.

If this parameter is absent (or is '-'), data is not compressed before being encoded.

URL_or_file

Specifies the source of image data.

If this parameter is absent, a chooser dialog box is displayed to let the user specify which image file to use.

Examples:

```
setImage  
setImage fileref anyURI  
setImage - hexBinary gzip  
setImage location - - file://localhost/icons/apache_pb.gif  
setImage - XML - C:\graphics\logo.svgz
```


74. split

Parameter syntax:

```
[ '[implicitElement]' ]?
```

Splits explicitly or implicitly selected element in two parts, the split point being specified by caret position.

75. start

Similar to run expect that external command is executed asynchronously (like Windows `start` or Unix `&`).

76. status

Parameter syntax:

```
message
```

Displays a message in the status bar found at the bottom of XXE main window.

This command is useful to write macro-commands.

Example:

```
status Command foo completed
```

77. toggleCollapsed

Parameter syntax:

```
[ 'toggle'|'collapse'|'expand'|'collapseAll'|'expandAll'|  
  [ '+' 'toggle'|'collapse'|'expand'|'collapseAll'|'expandAll' ]? ]?
```

Changes the state of the *nearest collapsible view*.

The ``nearest collapsible view'' is searched like this:

- Search starts at explicitly selected node if any; otherwise at node containing caret.
- If this node is an element and has a collapsible view, search is completed: this view is the ``nearest collapsible view''.
- Otherwise search continues with the parent of the node.

The parameter of this command specifies up to two operations. The default operation is `toggle`. Supported operations are:

`toggle`

Collapses nearest collapsible view if it is expanded and expands nearest collapsible view if it is collapsed.

`collapse`

Collapses nearest collapsible view if it is expanded; otherwise has no effect.

`expand`

Expands nearest collapsible view if it is collapsed; otherwise has no effect.

`collapseAll`

Collapses nearest collapsible view and then, recursively collapses all its collapsible descendant views.

expandAll

Expands nearest collapsible view and then, recursively expands all its collapsible descendant views.

Recommended bindings (found in samples2/customize.xxe):

```
<binding>
  <keyPressed code="ESCAPE" />
  <charTyped char="/" />
  <command name="toggleCollapsed" />
</binding>

<binding>
  <keyPressed code="ESCAPE" />
  <charTyped char="+" />
  <command name="toggleCollapsed" parameter="expandAll" />
</binding>

<binding>
  <keyPressed code="ESCAPE" />
  <charTyped char="-" />
  <command name="toggleCollapsed" parameter="collapseAll" />
</binding>

<binding>
  <keyPressed code="ESCAPE" />
  <charTyped char="0" />
  <command name="toggleCollapsed" parameter="collapseAll+expand" />
</binding>
```

78. undo

No parameter.

Undo last command.

79. viewHTML

Parameter syntax:

```
URL or filename
```

Use web browser to view specified file. If the parameter contains a *relative* URL or filename, this relative location is resolved against the location of the document being edited.

Specified file does not necessarily contain HTML; suffice for a file to have a content supported by the web browser.

The web browser used by this command is determined as follows:

- If system property `XXE_HTML_PREVIEW_COMMAND` has been defined, the value of this property is used as the template for the command which launches the browser.

Configuration element `property` makes it easy specifying such system property. Example: add this `property` element to your `customize.xxe` in order to preview XHTML files using Firefox:

```
<property name="XXE_HTML_PREVIEW_COMMAND">firefox "%U"</property>
```

- Otherwise

- On Windows:

```
start "" "%U"
```

- On MacOS X:

```
open "%U"
```

- On Linux:

```
(mozilla -remote "openURL(%U)" 1> /dev/null 2>&1) || (mozilla "%U" &)
```

Note that, unlike Netscape 4, Mozilla does not support the `-raise` option, nor automatically brings its window to front.

Variable `%U` is substituted with the parameter of the command (possibly after converting a filename to an URL) before the command is executed.

Note that on Unix, there is no need to end the command template with a `'&'`.

Example of use:

```
<command name="docb.startHTMLViewer">
  <macro>
    <sequence>
      <get context="$implicitElement/@url" expression="resolve-uri(.)" />
      <command name="viewHTML" parameter="%_" />
    </sequence>
  </macro>
</command>
```

80. wrap

A variant of the `convert` command. The unique difference is that when a single element is selected, the selected element is given a new parent element.

Example:

```
"<simpara>the <emphasis>little</emphasis> girl.</simpara>"
```

wrapped in a `<note>` gives

```
"<note><simpara>the <emphasis>little</emphasis> girl.</simpara></note>".
```

This is different from command `convert` which can be used to ``morph" selected element to another kind of element. For example, `convert` cannot wrap the above `simpara` in a `note` but can morph it to a `para`.

Examples:

```
wrap
wrap [implicitElement] div
```

81. xpathSearch

Parameter syntax:

```
[ implicit_selection ]? [ XPath_expression ]?
```

Evaluates specified XPath expression in the context of selected node. The evaluation of the XPath expression must return a nodeset. If this nodeset exclusively contains *contiguous siblings*, all the nodes in the nodeset are selected. Otherwise, first node (in document order) of the nodeset is selected.

If the evaluation of the expression returns attributes, the corresponding elements are selected.

It is not possible to select the document node or sibling nodes of the root element.

If the XPath expression is not specified, a dialog box is displayed. This dialog box has a Simple tab which allows to specify commonly used expressions without having to know the XPath standard and an Advanced tab which allows to specify arbitrarily complex XPath expressions.

When this command is used interactively, qualified names found in the XPath expression may be specified using the namespace prefixes defined in the document. When this command is used in XXE configuration files, the `{namespace_URI}local_part` notation must be used instead.

Example:

```
xpathSearch
xpathSearch [implicitNode]
xpathSearch //@revisionflag
xpathSearch [implicitElement] following::xs:complexType[1]
xpathSearch [implicitElement] following::{http://www.w3.org/2001/XMLSchema}complexType[1]
```

82. XXE.edit

Parameter syntax:

```
[ file_name | URL ]?
```

Opens a document in XXE, unless it is already opened, in which case this command just brings all its views to front and makes this document the ``active" document.

Without a parameter, this command tries to find a reference to an external document (created using an external general entity or a `xi:include` element) starting its search from the node selection or from the element containing the caret. If such reference is found, this command acts on the referenced document.

For example, if a book contains references to external chapter documents, moving the caret anywhere inside a chapter and executing this command without a parameter allows the user to edit this chapter.

Parameters:

file_name OR *URL*

Opens or activates specified document.

Returns newly opened or newly activated `com.xmlmind.xmlmind.doc.Document` (for use by higher-level commands) or `Command.EXECUTION_FAILED` if specified document is not already opened in the editor and fails to be opened.

DocBook example: edit other DocBook document referenced in the `url` attribute of implicitly or explicitly selected `ulink` element.

```
<command name="docb.editDocument">
  <macro>
    <sequence>
      <get context="$implicitElement/@url" expression="resolve-uri(.)" />
      <command name="XXE.edit" parameter="%_" />
    </sequence>
  </macro>
</command>
```

83. XXE.open

Parameter syntax:

```
[ '[reopen]' | '[reopenIfNewer]' | file_name | URL ]?
```

Opens a document in XXE.

Without a parameter, this command displays the file chooser dialog box to let the user specify which document to open.

Parameters:

[reopen]

Reopens document currently opened in XXE. Useful to implement a "revert to saved" command.

[reopenIfNewer]

Reopens document currently opened in XXE, but only if it has been modified by an external application.

If the document currently opened in XXE has not been modified by an external application, this command does nothing at all, succeeds and returns current `com.xmlmind.xmledit.doc.Document`.

Note that this option works exactly like [reopen] if the document is stored on a HTTP or FTP server. That is, XXE will only check the dates of local files.

file_name or *URL*

Opens specified document.

Returns newly opened `com.xmlmind.xmledit.doc.Document` (for use by higher-level commands) or `Command.EXECUTION_FAILED` if specified document cannot be opened or if user has canceled the command.

See `XXE.save` for an example of use for this command.

84. XXE.openAsTemplate

Parameter syntax:

```
[ file_name | URL ]?
```

Creates a new document using an existing document as a template. After loading a copy of the existing document, all the content of the copy is discarded to just keep a *valid skeleton* of the root element.

Without a parameter, this command displays the file chooser dialog box to let the user specify an existing document.

Parameters:

file_name or *URL*

Specifies which document to use as a template.

Returns newly created `com.xmlmind.xmledit.doc.Document` (for use by higher-level commands) or `Command.EXECUTION_FAILED` if specified template cannot be opened or if user has canceled the command.

85. XXE.openCopy

Parameter syntax:

```
[ file_name | URL ]?
```

Creates a new document by copying an existing document.

Without a parameter, this command displays the file chooser dialog box to let the user specify an existing document.

Parameters:

file_name or *URL*

Specifies which document to copy.

Returns newly created `com.xmlmind.xmledit.doc.Document` (for use by higher-level commands) or `Command.EXECUTION_FAILED` if specified document cannot be opened or if user has canceled the command.

86. `XXE.close`

Parameter syntax:

```
[ file_name | URL ]?
```

If a file name or an URL has been specified in the parameter, closes the document having this location; otherwise closes the active document.

Returns `Command.EXECUTION_FAILED` if user has canceled the command. Otherwise, returns `null`.

87. `XXE.save`

Parameter syntax:

```
[ '[ifNeeded]' ]?
```

Saves document currently opened in `XXE`.

[ifNeeded]

With this option, this command does nothing at all but can be successfully executed if current document does not need to be saved.

Without this option, this command cannot be executed if current document does not need to be saved.

This option is useful in macro commands such as the one in the example below.

Returns `Command.EXECUTION_FAILED` if document cannot be saved or if user has canceled the command. Otherwise, returns `null`.

Example:

```
<command name="editXMLSource">
  <macro>
    <sequence>
      <command name="XXE.save" parameter="[ifNeeded]" />
      <command name="run" parameter='emacs "%D"' />
      <command name="XXE.open" parameter="[reopenIfNewer]" />
    </sequence>
  </macro>
</command>
```

1. Save the document being edited, if this is needed.
2. Load it in external text editor GNU Emacs. Use this text editor to modify it or simply to view it.
3. Reload the document in `XXE`, but only if it has been modified using Emacs.

88. `XXE.saveAs`

Parameter syntax:

```
[ file_name | URL ]?
```

Saves document being edited to a different location.

Without a parameter, this command displays the file chooser dialog box to let the user specify the document location.

Parameters:

file_name OR *URL*

Specifies document location.

Returns `Command.EXECUTION_FAILED` if an error occurred when saving the document or if user has canceled the command. Otherwise, returns `null`.

89. XxE.saveCopy

Similar to `XxE.saveAs` except that it saves a copy of document being edited.

Chapter 7. XPath functions

All the standard XPath 1.0 functions are supported: `boolean`, `ceiling`, `concat`, `contains`, `count`, `false`, `floor`, `id`, `lang`, `last`, `local-name`, `name`, `namespace-uri`, `normalize-space`, `not`, `number`, `position`, `round`, `starts-with`, `string`, `string-length`, `substring`, `substring-after`, `substring-before`, `sum`, `translate`, `true`.

The following XSLT 1.0 functions are also supported: `current`, `document`, `format-number` (except that the 3-argument form, `format-number(number,string,string)`, is not supported), `system-property` (except that built-in `xsl:version`, `xsl:vendor` and `xsl:vendor-url` properties are not supported).

1. Extension functions

`node-set copy(node-set)`

Returns a deep copy of specified node set.

`node-set difference(node-set1, node-set2)`

Returns a node-set containing all nodes found in `node-set1` but not in `node-set2`.

`object if(boolean test1, object value1, ..., boolean testN, object valueN, ..., object fallback)`

Evaluates each `testi` in turn as a boolean, if the result of `testi` is true, returns corresponding `valuei`. Otherwise, if all `testi` evaluates to false, returns `fallback`.

`string join(node-set node-set, string separator)`

Converts each node in `node-set` to a string and joins all these strings using `separator`. Returns the resulting string.

Example: `join(//h1, ', ')` returns "Introduction, Conclusion" if the document contains 2 `h1` elements, one containing "Introduction" and the other "Conclusion".

`node-set intersection(node-set1, node-set2)`

Returns a node-set containing all nodes found in both `node-set1` and `node-set2`.

`boolean matches(string input, string pattern, string flags?)`

Similar to XPath 2.0 function `matches`. Returns true if `input` matches the regular expression `pattern`; otherwise, it returns false.

Note that unless `^` and `$` are used, the string is considered to match the pattern if any substring matches the pattern.

Optional `flags` may be used to parametrize the behavior of the regular expression:

`m`

Operate in multiline mode.

`i`

Operate in case-insensitive mode.

Examples: `matches("foobar", "^f.+x$")` returns true. `matches("CamelCase", "ca", "i")` returns true.

`number max(node-set), number max(number, ..., number)`

The first form returns the maximum value of all nodes of specified node set, after converting each node to a number.

Nodes which cannot be converted to a number are ignored. If all nodes cannot be converted to a number, returns NaN.

The second form returns the maximum value of all specified numbers (at least 2 numbers).

Arguments which cannot be converted to a number are ignored. If all arguments cannot be converted to a number, returns NaN.

number `min(node-set)`, number `min(number, ..., number)`

Same as `max()` but returns the minimum value of specified arguments.

number `pow(number1, number2)`

Returns *number1* raised to the power of *number2*.

string `replace(string input, string pattern, string replacement, string flags?)`

Similar to XPath 2.0 function `replace`. Returns the string that is obtained by replacing all non-overlapping substrings of *input* that match the given *pattern* with an occurrence of the *replacement* string.

The *replacement* string may use \$1 to \$9 to refer to captured groups.

Optional *flags* may be used to parametrize the behavior of the regular expression:

m

Operate in multiline mode.

i

Operate in case-insensitive mode.

Example: `replace("foobargeebar", "b(.+)r", "B$1R")` returns "fooBaRgeeBaR".

string `resolve-uri(string uri, ?string base?)`

If *uri* is an absolute URL, returns *uri*.

If *base* is specified, it must be a valid absolute URL, otherwise an error is reported.

If *uri* is a relative URL,

- if *base* is specified, returns *uri* resolved using *base*;
- if *base* is not specified, returns *uri* resolved using the base URL of the context node.

If *uri* is the empty string,

- if *base* is specified, returns *base*;
- if *base* is not specified, returns the base URL of the context node.

string `relativize-uri(string uri, ?string base?)`

Converts absolute URL *uri* to an URL which is relative to specified base URL *base*. If *base* is not specified, the base URL of the context node is used instead.

Uri must be a valid absolute URL, otherwise an error is reported. If *base* is specified, it must be a valid absolute URL, otherwise an error is reported.

Example: returns `"../john/.profile"` for `uri="file:///home/john/.profile"` and `base="file:///home/bob/.cshrc"`.

If *uri* cannot be made relative to *base* (example: `uri="file:///home/john/public_html/index.html"` and `base="http://www.xmlmind.com/index.html"`), *uri* is returned as is.

string `uri-to-file-name(string)`

Converts specified argument, a "file://" URL, to a native file name. Returns the empty string if argument is not a "file://" URL.

2. Java™ methods as extension functions

Note

The following section has been adapted from the documentation of James Clark's XT, one of the fastest XSLT engines, from which the XXE implementation of XPath has been extracted.

A call to a function `ns:foo` where `ns` is bound to a namespace of the form `java:className` is treated as a call of the static method `foo` of the class with fully-qualified name `className`. Example:

```
xmlns:file="java:java.io.File"
file:createTempFile('xxe', '.tmp')
```

Hyphens in method names are removed with the character following the hyphen being upper-cased. Example:

```
file:create-temp-file('xxe', '.tmp')
```

is equivalent to:

```
file:createTempFile('xxe', '.tmp')
```

A non-static method is treated like a static method with the `this` object as an additional first argument. Example:

```
file:delete-on-exit(file:createTempFile('xxe', '.tmp'))
```

A constructor is treated like a static method named `new`. Example:

```
xmlns:url="java:java.net.URL"
url:new('http://www.xmlmind.com/xmleditor/')
```

Overloading based on number of parameters is supported; overloading based on parameter types is not. Example, it is possible to invoke:

```
url:new('http://www.xmlmind.com/xmleditor/')
```

though both `java.net.URL(java.lang.String spec)` and `java.net.URL(java.net.URL context, java.lang.String spec)` exist. It is not possible to invoke:

```
file:new('.')
```

because both `java.io.File(java.lang.String pathname)` and `java.io.File(java.net.URI uri)` exist.

Extension functions can return objects of arbitrary types which can then be passed as arguments to other extension functions.

Types are mapped between XPath and Java™ as follows:

XPath type	Java™ type
string	java.lang.String
number	double
boolean	boolean
node-set	com.xmlmind.xmledit.xpath.NodeIterator

On return from an extension function, an object of type `com.xmlmind.xmledit.doc.XNode` is also allowed and will be treated as a node-set; also any numeric type is allowed and will be converted to a number.