XMLmind XML Editor - Commands

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

XMLmind XML Editor - Commands

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

Publication date November 15, 2024

Abstract

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

Table of Contents

List of Figures

List of Examples

Part I. Guide

Table of Contents

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 keystroke, 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 [173]. 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 [84] 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 commands) need to be declared in an **XXE** configuration file (see XMLmind XML Editor - Configuration and Deployment). Example:

```
<command name="xhtml.preview">
  <class>com.xmlmind.xmleditext.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 commands must be specified in an **XXE** configuration file (see Section 4, “command” in *XMLmind XML Editor - Configuration and Deployment*). Example:

```
<command name="convertCaseMenu">
  <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" />
  </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 must be specified in an **XXE** configuration file (see Section 4, “command” in *XMLmind XML Editor - Configuration and Deployment*). Example:

```
<command name="xhtml.convertToLink">
  <macro>
    <sequence>
      <command name="convert" parameter="a" />
      <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 must be specified in an **XXE** configuration file (see Section 4, “command” in *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/` and binding the command to be tested to a keystroke.

XXE user preferences directory is:

- *`$HOME`*`/.xxe10/` on Linux.
- *`$HOME`*`/Library/Application Support/XMLmind/XMLEditor10/` on the Mac.
- *`%APPDATA%`*`\XMLmind\XMLEditor10\` on Windows. Example: `C:\Users\john\AppData\Roaming\XMLmind\XMLEditor10\`.

  If you cannot see the "`AppData`" directory using Microsoft Windows File Manager, turn on **Tools**>**Folder Options**>**View**>**File and Folders**>**Show hidden files and folders**.

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 repeatable="true" undoable="true"
         label="Convert to Bold">
    <sequence>
      <command name="convert"
               parameter="[implicitElement] emphasis" />
      <command name="putAttribute" parameter="role bold" />
    </sequence>
  </macro>
</command>

</configuration>
```

More information 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/macro_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 [136] 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 [144] return an actual result (a file name or an URL in the case of command selectFile [144]) 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 [144], 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 repeatable="true" undoable="true"
         label="Convert to Bold">
    <sequence>
      <command name="convert"
               parameter="[implicitElement] emphasis" />
      <command name="putAttribute" parameter="role bold" />
    </sequence>
  </macro>
</command>
```

Using a sequence [27] 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 [131] because the newly created `emphasis` element has been automatically selected by the convert [92] 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 [27] element to do this:

```
<command name="convertToBold2">
  <macro repeatable="true" undoable="true"
         label="Convert to Bold">
    <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 [27] element will execute the first alternative which can be executed:

- Explicitly selecting (using command selectNode [149]) the `emphasis` element if such element is implicitly (or explicitly) selected;
- OR converting anything else to an `emphasis` element, using command convert [92].

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 a `p`, `li`, etc.

Command selectNode [149] can, not only blindly select nodes, but it can also select nodes conditionally, if theses nodes match a list of elements passed as a parameter string.

Therefore the idea is to use selectNode [149], not for its ability to select nodes, but for its ability to test where the caret is.

The pass [126] and fail [109] 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 {http://www.w3.org/1999/xhtml}br" />

      <sequence>
        <pass><command name="selectNode"
                parameter="parent {http://www.w3.org/1999/xhtml}p
                           {http://www.w3.org/1999/xhtml}li
                           {http://www.w3.org/1999/xhtml}dt
                           {http://www.w3.org/1999/xhtml}dd
                           {http://www.w3.org/1999/xhtml}th
                           {http://www.w3.org/1999/xhtml}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 [88], 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 [139] 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 [123] 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 [9]: "%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 formatTextAs [109] and paste [126].

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

<!-- template(para,PAA.para) is defined in the stock
     DocBook 4 configuration. -->
```

```
<binding>
  <keyPressed code="F5" />
  <command name="insertAfterAs"
           parameter="#template(para,PAA.para)" />
</binding>
```

Command `formatTextAs` 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 `formatTextAs` 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 [129] 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 undoable="true" label="Insert Favorite">
    <sequence>
      <command name="pick"
               parameter="Favorites true
                          W3C
                          http://www.w3.org/
                          'DocBook Home Page'
                          http://www.docbook.org/
                          Java
                          http://www.java.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 [116], which does not return a result, clears "`%_`".

```
<sequence>
  <command name="pick"
           parameter="Favorites true
                      W3C
                      http://www.w3.org/
                      'DocBook Home Page'
                      http://www.docbook.org/
                      Java
                      http://www.java.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 [129] 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 [28] and are read using special construct get [28]. 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 [28] and get [28]. This means that you don't need to learn XPath to use set [28] and get [28] to do simple things. However, it is important to remember this:

- *User-defined variables cannot be referenced in constructs other than set [28], get [28], test [28] and match [27].*

  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 [28] construct must be used because it is the only way to return in "`%_`" the value of "`$url`".

- Do not forget to add `plainString=true` to element `set`. Otherwise, the value of attribute `expression` is understood as being an XPath expression.
- 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 **gimp**. It is launched using command start [159].

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

The above macro shows how to use get [28] at its best:

1. First the `context` attribute, common to all XPath-based constructs get [28], set [28], test [28] and match [27], is evaluated as a node set, using the document as a context node.
2. The `expression` attribute, common to get [28], set [28] and test [28], is evaluated as a string using the context node found in previous step.

   If get [28], set [28], test [28] or match [27] 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 [31] 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() in *XMLmind XML Editor - Support of XPath 1.0* and uri-to-file-name() in *XMLmind XML Editor - Support of XPath 1.0* 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 Example 4.5, “Using the XPath-based constructs match and set” [40].

```
<command name="moveListItemDown">
  <macro undoable="true" label="Move List Item Down">
    <sequence>
      <command name="selectNode"
        parameter="ancestorOrSelf[implicitElement] listitem callout step" />❶
      <match context="$selected" pattern="*[position() &lt; last()]" />❷
      <set variable="anchor" context="$selected"
           expression="./following-sibling::*[1]" />❸
      <command name="cut" />❹
      <set variable="selected" expression="$anchor" />❺
      <command name="paste" parameter="after" />❻
    </sequence>
  </macro>
</command>
```

❶ This step ensures that the macro can be executed only inside a list item.

❷ This step ensures that the macro cannot be executed for last list item.

It uses the XPath-based construct match [27]. 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 [126], fail [109] and test [28], is only a test. When match [27] is actually executed, it does nothing at all.

❸ 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 [31]:

- 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).

❹ Cut selected list item.

❺ 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.

❻ 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 undoable="true" label="Transpose Characters">
    <sequence>
      <test expression="not($selected) and not($mark) and
                        $dotOffset &gt; 0 and
                        $dotOffset &lt; string-length($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 [154], moveDotTo [124], cut [95] and paste [126], but also XPath-based construct test [28].

As a pseudo-command of a macro, test [28] 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 [126], fail [109] and match [27], is only a test. When test [28] 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 pre-defined variables mapped to the selection in XXE [31].

Part II. Reference

Table of Contents

Chapter 3. Menu commands

1. Reference

```
<command
  name = NMTOKEN (optionally preceded by
  a command namespace in XMLmind XML Editor - Configuration and Deployment)
>
  Content: class | menu | macro | process
</command>

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

<separator
/>

<item
  label = non empty token
  icon = anyURI
  command = NMTOKEN (optionally preceded by
  a command namespace in XMLmind XML Editor - Configuration and Deployment)
  parameter = string
/>
```

Define a popup menu of commands. This special type of command, typically invoked from contextual macro-commands, allow to redefine or extend the right-click, contextual, popup menu. See below [21].

The parameter passed to a command containing a menu may be used to parameterize the items of this menu. This parameter is split like in a command line. A part of the split parameter may be referenced as variable `%0`, `%1`, `%2`, ..., `%9`. Variable `%*` may be used to reference the whole parameter of the command.

Note that the %-variable substitution is performed on in the `label` and in the `parameter` attributes of an `item` element. There is no %-variable substitution in the `command` attribute.

Example:

```
<command name="greetings">
  <menu>
    <item label="Say '%0'"
          command="alert" parameter="%0"/>
    <item label="Say '%1'"
          command="alert" parameter="%1"/>
  </menu>
</command>
```

Invoking `<command name="greetings" parameter="'Hello world!' 'Goodbye world.'"/>` displays a popup menu equivalent to:

```
<menu>
  <item label="Say 'Hello world!'"
        command="alert" parameter="Hello world!"/>
  <item label="Say 'Goodbye world.'"
        command="alert" parameter="Goodbye world."/>
</menu>
```

The `binding` configuration element (see Section 3, “binding” in *XMLmind XML Editor - Configuration and Deployment*) also has a `menu` child element. However menu commands are more useful because:

- They can be bound to a keystroke. Popup menus in `binding` can only be bound to a mouse click.
- The parameter passed to the command may be used to parameterize the items of the menu.
- They can be used to redefine or extend the right-click, contextual, popup menu.

2. Redefining or extending the right-click, contextual, popup menu

The right-click, contextual, popup menu is displayed by a command called `contextualMenu` [92]. In order to determine the content of this popup menu, the `contextualMenu` command proceeds as follows:

1. `contextualMenu` attempts to execute in order (see below [22]) all the commands found in the "*`current_configuration_name`* `contextualMenu`" namespace. If, given the current context, one of such commands can be executed, then this command is expected to display a custom menu replacing the “standard” right-click, contextual, popup menu. Therefore, once this command is executed, `contextualMenu` ends its work at this point.

2. `contextualMenu` attempts to execute in order all the commands found in the "`contextualMenu`" namespace. If, given the current context, one of such commands can be executed, then this command is expected to display a custom menu replacing the “standard” right-click, contextual, popup menu. Therefore, once this command is executed, `contextualMenu` ends its work at this point.

3. `contextualMenu` attempts to execute in order all the commands found in the "*`current_configuration_name`* `contextualMenuItems`" namespace. If, given the current context, one of such commands can be executed, then this command is expected to display a custom menu contributing some menu items to the “standard” right-click, contextual, popup menu. Therefore, once this command is executed, `contextualMenu` captures the menu items (without displaying any popup menu at this stage) and proceeds with the next command found in in the "*`current_configuration_name`* `contextualMenuItems`" namespace.

4. `contextualMenu` attempts to execute in order all the commands found in the "`contextualMenuItems`" namespace. If, given the current context, one of such commands can be executed, then this command is expected to display a custom menu contributing some menu items to the “standard” right-click, contextual, popup menu. Therefore, once this command is executed, `contextualMenu` captures the menu items (without displaying any popup menu at this stage) and proceeds with the next command found in in the "`contextualMenuItems`" namespace.

5. `contextualMenu` displays the “standard” right-click, contextual, popup menu. This menu contains all the items collected during step #3 and step #4.

The invocation order of the commands belonging to a given namespace is the lexical order of the local names of the commands. For example, command `{My Config contextualMenuItems}barMenuItems` is invoked before command `{My Config contextualMenuItems}fooMenuItems`.

By default, **XXE** attempts to execute the equivalent of the following command during step #2:

```
<command name="{contextualMenu}AutoSpellMenu">
  <sequence>
    <command name="autoSpellChecker"
             parameter="popupMenu"/>
  </sequence>
</command>
```

More information about command `autoSpellChecker` [88].

By default, **XXE** attempts to execute the equivalent of the following command during step #4:

```
<command name="{contextualMenuItems}xxeEditMenuItems">
  <menu>
    <item label="Repeat"
          command="repeat"/>
    <separator/>
    <item label="Cut"
          command="cut" parameter="[implicitElement]"/>
    <item label="Copy"
          command="copy" parameter="[implicitElement]"/>
    <item label="Paste Before"
          command="paste" parameter="before[implicitElement]"/>
    <item label="Paste"
          command="paste" parameter="toOrInto"/>
    <item label="Paste After"
          command="paste" parameter="after[implicitElement]"/>
    <item label="Delete"
          command="delete" parameter="[implicitElement]"/>
    <separator/>
    <item label="Replace..."
          command="replace" parameter="[implicitElement]"/>
    <item label="Insert Before..."
          command="insert" parameter="before[implicitElement]"/>
    <item label="Insert Into..."
          command="insert" parameter="into"/>
    <item label="Insert After..."
          command="insert" parameter="after[implicitElement]"/>
    <item label="Convert..."
          command="convert" parameter="[implicitElement]"/>
    <item label="Wrap..."
          command="wrap" parameter="[implicitElement]"/>
  </menu>
</command>
```

Example 3.1. Whatever the configuration, add extra menu items to the right-click, contextual, popup menu

```
<command name="{contextualMenuItems}xxeExtraMenuItems">
  <menu>
    <item label="Split" command="split" />
    <item label="Join" command="join" />
  </menu>
</command>
```

More information about commands `split` [159] and `join` [123].

If you want to the give the above snippet a try, add it to your `customize.xxe` file.

Example 3.2. DITA topic configuration: when an image is selected, allow to edit it using a helper application

```
<command name="{$c❶ contextualMenuItems}editImageMenuItem">
  <macro>
    <sequence>
      <match context="$selectedElement"
             pattern="image[@href and @href != '???']" />❷
      <command name="{dita}editImageMenuItem" />
    </sequence>
  </macro>
</command>

<command name="{dita}editImageMenuItem">❸
  <menu>
    <item label="Edit image"
          command="editObject"❹
          parameter="href anyURI" />
  </menu>
</command>
```

❶ "$c" is a shorthand for the name of the current configuration ("DITA" in the case of the above command).

❷ The "**Edit image**" item is added to the popup menu only if an `image` element having an `href` attribute is explicitly selected.

❸ A macro-command such as "`{DITA contextualMenuItems}editImageMenuItem`" may not have a `menu` child element element, hence the need to define helper command `{dita}editImageMenuItem`.

❹ More information about command `editObject` [104].

If you want to the give the above snippet a try, add it to *`XXE_install_dir`*`/addon/config/dita/topic.xxe`.

Example 3.3. Any of the two XHTML 1.0 configurations: when some text is selected, display a special, simpler, popup menu

```
<command name="{$c❶ contextualMenu}textMenu">
  <macro>
    <sequence>
      <test expression="$selectedChars" />❷
      <command name="{xhtml}textMenu" />
    </sequence>
  </macro>
</command>

<command name="{xhtml}textMenu">❸
  <menu>
    <item label="Cut" command="cut" />❹
    <item label="Copy" command="copy" />
    <item label="Paste"
          command="paste" parameter="to"/>
    <separator/>
    <item label="Convert to b"
          command="convert"
          parameter="{http://www.w3.org/1999/xhtml}b"/>
    <item label="Convert to i"
          command="convert"
          parameter="{http://www.w3.org/1999/xhtml}i"/>
    <item label="Convert to tt"
          command="convert"
          parameter="{http://www.w3.org/1999/xhtml}tt"/>
  </menu>
</command>
```

❶ "`$c`" is a shorthand for the name of the current configuration ("`XHTML Transitional`" or "`XHTML Strict`").

❷ The “standard” right-click, contextual, popup menu is replaced by `textMenu` only if some text is selected.

❸ A macro-command such as "`{XHTML Transitional textMenu}`" may not have a `menu` child element element, hence the need to define helper command `{xhtml}textMenu`.

❹ More information about commands `cut` [95], `copy` [93], `paste` [126], `convert` [92].

If you want to the give the above snippet a try, add it to *`XXE_install_dir`*`/addon/config/xhtml/xhtml_common.incl`.

Chapter 4. Macro commands

1. Reference

```
<command
  name = NMTOKEN (optionally preceded by
  a command namespace in XMLmind XML Editor - Configuration and Deployment)
>
  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|script ]+
</choice>

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

<command
  name = NMTOKEN (optionally preceded by
  a command namespace in XMLmind XML Editor - Configuration and Deployment)
  parameter = string
/>

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

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

<script
  location = anyURI
  context = XPath expr. returning a node set : "/"
  argument0 = XPath expression
  argument1 = XPath expression
  argument2 = XPath expression
  argument3 = XPath expression
  argument4 = XPath expression
  argument5 = XPath expression
  argument6 = XPath expression
  argument7 = XPath expression
  argument8 = XPath expression
  argument9 = XPath expression
><![CDATA[
  XED source
]]></script>
```

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 [5]). 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 [29].

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 [29].

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 specified pattern can be parsed and matches specified context node. 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 [31].

test

Can be executed if specified expression can be parsed and evaluates to `true()` given specified 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 [31].

get

Can be executed if specified expression can be parsed and evaluated given specified context 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 [31].

set

Can be executed if specified expression can be parsed and evaluated given specified context node. Execution assigns to specified variable the value of specified expression.

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

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 [31].

script

See Section 2, “Running XED scripts in a macro” [34].

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` and `script` 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` and `script` pseudo-commands cannot be executed.

1.1. Pass and fail cheat sheet

`pass`*`AB`*	`pass` can be executed if *`A`* and *`B`* can be executed.
`passsequence`*`AB`*	`pass` can be executed if *`A`* can be executed.
`passchoice`*`AB`*	`pass` can be executed if *`A`* or *`B`* can be executed.
`fail`*`AB`*	`fail` can be executed if *`A`* or *`B`* cannot be executed.
`failsequence`*`AB`*	`fail` can be executed if *`A`* cannot be executed.
`failchoice`*`AB`*	`fail` can be executed if *`A`* and *`B`* 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 4.2, “Macro-variables” [39] below: `<command name="putAttribute" parameter="%0 %1"/>`.

Variable	Description
`%0`, `%1`, `%2`, ..., `%9`, `%*`	A macro-command can have a parameter. This string is split like in a command line. A part of the split parameter can be referenced as variable `%0`, `%1`, `%2`, ..., `%9`. `%*` can be used to reference the whole parameter of the macro-command.
`%D`, `%d`	`%D` is the file name of the document being edited. Example: `C:\novel\chapter1.xml`. This variable is replaced by an empty string if the document being edited is found on a remote HTTP or FTP server. `%d` is the URL of the document being edited. Example: `file:///C:/novel/chapter1.xml`.
`%P`, `%p`	`%P` is the name of the directory containing the document being edited. Example: `C:\novel`.

Variable	Description
	This variable is replaced by an empty string if the document being edited is found on a remote HTTP or FTP server. %p is the URL of the directory containing the document being edited. Example: file:///C:/novel. Note that this URL does not end with a '/'.
%N, %R, %E	%N is the base name of the document being edited. Example: chapter1.xml. %R is the base name of the document being edited without the extension, if any (sometimes called the root name). Example: chapter1. %E is the extension of the document being edited, if any. Example: xml. Note that the extension does not start with a '.'.
%n, %r, %e	Similar to %N, %R, %E except that these variables contain properly escaped URI components. For example if %R contains "foo bar", then %r contains "foo%20bar".
%S	%S is the native path component separator of the platform. Example: '\' on Windows.
%U	User's account name. Example: john.
%H, %h	%H is the user's home directory. Example: /home/john. %h is the URL of the user's home directory. Example: file:///home/john. Note that this URL does not end with a '/'.
%A, %a	%A is the user's preferences directory. Example: /home/john/.xxe10. %a is the URL of the user's preferences directory. Example: file:///home/john/.xxe10. Note that this URL does not end with a '/'.
%X (or %W), %x (or %w)	%X (or %W) is the name of the user's current working directory (that is, the current working directory of XXE). Example: C:\Users\john\Documents\report. %x (or %w) is the URL of the user's current working directory. Example: file:///C:/Users/john/Documents/report. Note that this URL does not end with a '/'.
%C, %c	%C is the name of the directory containing the XXE configuration file from which the macro command has been loaded. Example: C:\Program Files\XMLmind_XML_Editor\addon\config\docbook.

Variable	Description
	%c is the URL of the above directory. Example: file:///C:/Program%20Files/XMLmind_XML_Editor/addon/config/docbook. 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 *c* contained in the macro-command, this variable is substituted with the result of the command executed before *c*. 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 [77]).

In addition to the above variables, a macro-command may reference any Java™ system property or environment variable. Examples: %{user.home} (for system property user.home), %{HOME} (for environment variable HOME)

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
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 `drop`.	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 `drop`.	N/A
clipboard	string	String contained in the system clipboard if any, the empty string otherwise.	Copies string value to the system clipboard.
dot	text, comment or PI node	Textual node containing the caret. Empty node set if the document does not contain text, comments or PIs.	Moves caret at the beginning of specified textual node.
dotOffset	integer	Offset of the caret within the textual node containing it. First offset is 0. Last valid offset is *after* last character. -1 if the document does not contain text, comments or PIs.	Moves caret to specified offset.
hasExplicitSelection	boolean	True if there is an explicit text or node selection; false otherwise.	N/A
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.
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.

Variable	Value	Read	Write
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.
selected	node	First selected node in the node selection (first in document order, not 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 (last in document order, not 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.
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.
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.
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.

Variable	Value	Read	Write
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.
systemSelection	string	String contained in the system selection[b] if any, the empty string otherwise.	Copies string value to the system selection[b].

[a]The 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.

[b]The system selection is emulated using a *private* clipboard on non Unix/X-Window platforms

2. Running XED scripts in a macro

The `script` element allows to a run a XED script in the context of a macro.

```
<script
  location = anyURI
  context = XPath expr. returning a node set : "/"
  argument0 = XPath expression
  argument1 = XPath expression
  argument2 = XPath expression
  argument3 = XPath expression
  argument4 = XPath expression
  argument5 = XPath expression
  argument6 = XPath expression
  argument7 = XPath expression
  argument8 = XPath expression
  argument9 = XPath expression
><![CDATA[
  XED source
]]></script>
```

XED is a very small, very simple scripting language, leveraging the native XPath 1.0 implementation of XMLmind XML Editor, allowing to modify in place the document being edited. The reference manual of XED is found in Part II, “The XED scripting language” in *XMLmind XML Editor - Support of XPath 1.0*.

The source of the script

The source of the XED script may be found inside the `script` element. In such case, it's strongly recommended to use a `CDATA` section.

If the source of the XED script is not found inside the `script` element, then the `location` attribute is considered. This attribute must point to an external text file, typically having a `.xed` extension, containing

the source of the script. External XED scripts are cached, therefore this is almost no performance penalty for using external script files.

The main difference between using an internal XED script and using an external XED script is that an internal XED script automatically inherits all the namespace declarations, except the default namespace declaration, which are in scope with the `script` element. Example:

```
<configuration
  xmlns="http://www.xmlmind.com/xmleditor/schema/configuration"
  xmlns:cfg="http://www.xmlmind.com/xmleditor/schema/configuration">
  ...
        <script>
          ...
        </script>
```

is equivalent to:

```
<configuration
  xmlns="http://www.xmlmind.com/xmleditor/schema/configuration"
  xmlns:cfg="http://www.xmlmind.com/xmleditor/schema/configuration">
  ...
        <script>
          namespace cfg="http://www.xmlmind.com/xmleditor/schema/configuration";
          ...
        </script>
```

Storing different scripts in the same XED script file

Note that the URI contained in the `location` attribute may have a fragment. This fragment specifies the name of the XED macro-command in *XMLmind XML Editor - Support of XPath 1.0* to be executed.

This allows to store different scripts —that is, several unrelated XED macro-commands— in the same script file. Example: script file `hello.xed` contains 2 XED macros:

```
macro sayHello() {
    message("Hello world!");
}

macro sayGoodbye() {
    message("Goodbye world!");
}
```

The following **XXE** macro-command allows to invoke `sayGoodbye()` found in script file `hello.xed`:

```
<command name="sayGoodbye">
  <macro>
    <sequence>
      <script location="hello.xed#sayGoodbye" />
    </sequence>
  </macro>
</command>
```

An equivalent **XXE** macro-command would be:

```
<command name="sayGoodbye">
  <macro>
    <sequence>
      <script>
      include "hello.xed";
      sayGoodbye();
      </script>
    </sequence>
  </macro>
</command>
```

Passing arguments to the script

Attributes `argument0` to `argument9` are evaluated as XPath expressions in the context of the node specified using the `context` attribute. The results of the evaluations are passed to the script by the means of global variables `script-argument0` to `script-argument9`.

If, for any reason, the evaluation of `argument`*`I`* fails, the script is nevertheless evaluated. Simply global variables `script-argument`*`I`* are not defined. That's why these arguments are typically used as follows:

```
set-variable("myOption", defined("script-argument7", "off"));
```

If variable `script-argument7` is defined, use its value, otherwise use string "`off`". See XPath extension function defined() in *XMLmind XML Editor - Support of XPath 1.0*.

Returning a value from the script

The result of a script is specified in global variable `script-result`. The result of a script may be used in the subsequent commands contained in the macro by the means of the "`%_`" special variable. If a script does not set global variable `script-result`, then this script will not change the value of "`%_`" special variable.

Example:

```
<command name="SayHello">
  <macro>
    <sequence>
      <script>
        set-variable("script-result", "Hello world!");
      </script>

      <command name="alert" parameter="%_"/>
    </sequence>
  </macro>
</command>
```

Changing the node or text selection from within the script

After it has been run, a script may affect the node or text selection by setting the following global variables:

Variable	Value	Description
`script-selected`	node set	Selects first node of the node set.
`script-selected2`	node set	Extends node selection to the first node of the node set.
`script-dot`	node set whose first node is a textual node	Moves the caret inside the first node of the node set. This first node must be a textual node of any kind (text, comment, processing instruction).
`script-dot-offset`	positive number; defaults to 0	Specifies the offset of the caret within the textual node obtained using variable `script-dot`.
`selected-mark`	node set whose first node is a textual node	Move the text selection mark inside the first node of the node set. This first node must be a textual node of any kind (text, comment, processing instruction).
`script-mark-offset`	positive number; defaults to 0	Specifies the offset of the text selection mark within the textual node obtained using variable `script-mark`.

It's possible to clear a selection mark of any kind but the caret, by setting the corresponding variable to an expression which does not evaluate to a node set. Examples: `set-variable("script-selected2", "")`, `set-variable("script-mark", false())`.

Otherwise, if the value of any of the above global variables has an unusable type (e.g. for `script-dot`, an empty node set or a node set not starting with a textual node), then this variable is simply ignored and no error is reported.

Setting `script-dot` (respectively `script-mark`) without setting `script-dot-offset` (respectively `script-mark-offset`) cause the caret (respectively the text selection mark) to be located at offset 0.

Caveats and pitfalls

- Always insert *newly created* nodes. Always replace existing nodes by *newly created* nodes. Never reuse nodes which have been detached from the document. Use extension function `copy()` in *XMLmind XML Editor - Support of XPath 1.0* when needed to. If you forget to do so, this is likely to break the undo manager of the XML editor.

  For example, if you need to move an element from one document location to another, first copy the element to be deleted using `copy()`, then delete the element, finally insert the *copy* at the new location.

- If you want to see what's printed by XED command `message()` in *XMLmind XML Editor - Support of XPath 1.0*, you need to set attribute `trace` to `true` on the `macro` element.

3. Examples

Example 4.1. Using sequence and choice

```
<command name="addListItem">
  <macro undoable="true">
    <choice>
      <sequence>
        <command name="selectNode"
                 parameter="ancestor[implicitElement]
                            {http://www.w3.org/1999/xhtml}ul
                            {http://www.w3.org/1999/xhtml}ol" />
        <command name="selectNode" parameter="child" />
        <command name="insertNode" parameter="sameElementAfter" />
      </sequence>

      <sequence>
        <choice>
          <sequence>
            <command name="selectNode"
                     parameter="ancestorOrSelf[implicitElement]
                                {http://www.w3.org/1999/xhtml}dt" />
            <!-- Assumes that a dt is followed by a dd. -->
            <command name="selectNode" parameter="nextSibling" />
          </sequence>
          <command name="selectNode"
                   parameter="ancestorOrSelf[implicitElement]
                              {http://www.w3.org/1999/xhtml}dd" />
        </choice>
        <command name="insert"
                 parameter="after {http://www.w3.org/1999/xhtml}dt" />
        <command name="insert"
                 parameter="after {http://www.w3.org/1999/xhtml}dd" />
        <command name="selectNode" parameter="previousSibling" />
      </sequence>
    </choice>
  </macro>
</command>
```

In the above example , 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="{http://www.w3.org/1999/xhtml}a" />
      <command name="putAttribute" parameter="%0 %1" />
    </sequence>
  </macro>
</command>

<binding>
  <keyPressed code="F3" />
  <command name="convertToLink" parameter="href ???" />
</binding>

<binding>
  <keyPressed code="F3" modifiers="shift" />
  <command name="convertToLink" parameter="name XXX" />
</binding>
```

In the above example, 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 the above example, 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="preview">
  <macro>
    <sequence>
      <pass>
        <match context="/*" pattern="html:html"
               xmlns:html="http://www.w3.org/1999/xhtml" />
        <fail><command name="XXE.save" /></fail>
      </pass>
      <command name="start" parameter="helper(defaultViewer) '%D'" />
    </sequence>
  </macro>
</command>
```

Start the web browser to preview the current document if it has an `html` root element and if it does not need to be saved.

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

```
<command name="moveListItemUp">
  <macro undoable="true" label="Move List Item Up">
    <sequence>
      <command name="selectNode"
        parameter="ancestorOrSelf[implicitElement] listitem callout step" />
      <match context="$selected" pattern="*[position() &gt; 1]" />
      <set variable="anchor" context="$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 drop

```
<binding>
  <appEvent name="drop" />
  <command name="dropURL" parameter="%{value}" />
</binding>

<command name="dropURL">
  <macro>
    <choice>
      <sequence>
        <match context="$clickedElement" pattern="html:a[@href]"
               xmlns:html="http://www.w3.org/1999/xhtml" />
        <set variable="selected" expression="$clickedElement" />
```

```
        <get expression="relativize-uri('%0')" />
        <command name="putAttribute" parameter="href '%_'" />
        <command name="status" parameter="Changed href."/>
      </sequence>

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

When a string is dropped on an XHTML `<a href="...">` element, this string is assigned to the `href` attribute (after considering this string as an URL and trying to make it relative to the base URL of the a element). When a string is dropped on any other element, XXE default action is used instead: consider the string as the URL or filename of a document to be opened.

The above macro uses the following XPath extension function: relativize-uri() in *XMLmind XML Editor - Support of XPath 1.0*.

Example 4.7. Insert nodes copied from another document

```
<binding>
  <keyPressed code="F7" />
  <command name="insertFromOtherDoc"
           parameter="into" />
</binding>

<property name="templateFile"
          url="true">VATrates.html</property>

<command name="insertFromOtherDoc">
  <macro>
    <sequence>
      <command name="prompt"
        parameter="Question
                   'ID of the element to be inserted at¬
caret position (e.g. germany_vat):'" />
      <set variable="elementId" expression="%_"
           plainString="true" />
      <get expression="serialize(document(system-property(¬
'templateFile'))//*[@id=$elementId])"/>
      <command name="paste" parameter="%0 %_" />
    </sequence>
  </macro>
</command>
```

The nodes are copied from file `VATrates.html`. Notice how a `property` configuration element having attribute `url=true` is used to make sure that the URL of the source document `VATrates.html` is resolved against the URL of the configuration file containing the macro.

The above macro uses the following XPath extension function: serialize() in *XMLmind XML Editor - Support of XPath 1.0*.

Example 4.8. Convert a `DocBook 5 para` to a `formalpara`

```
<command name="paraToFormalpara">
  <macro undoable="true"❶ label="Para to Formalpara">
    <sequence>
      <command name="selectNode"
               parameter="ancestorOrSelf[implicitElement]
                          {http://docbook.org/ns/docbook}para" />❷

      <script context="$selectedElement"❸><![CDATA[❹
        namespace db = "http://docbook.org/ns/docbook";❺

        set-variable("formalpara",
                     <db:formalpara><db:title>TITLE HERE</db:title></db:formalpara>);❻
        wrap-element($formalpara);❼

        set-variable("script-selected", $formalpara);❽
        set-variable("script-dot", ($formalpara//text())[1]);❾
      ]]></script>
    </sequence>
  </macro>
</command>
```

❶ Do not forget to specify `undoable="true"` when your `macro` contains a `script`.

❷ This makes sure that a `para` is explicitly selected.

❸ The script will not run unless a single element is explicitly selected.

❹ The XED source is directly contained in the `script` element. When this is the case, do not forget to use a `CDATA` section. An alternative would have been to specify the location of an external XED script using the `location` attribute of the `script` element.

❺ Declare the DocBook 5 namespace.

Note that the above XED script could also have used a default namespace:

```
namespace "http://docbook.org/ns/docbook";

set-variable("formalpara",
             <formalpara><title>TITLE HERE</title></formalpara>);
wrap-element($formalpara);

set-variable("script-selected", $formalpara);
set-variable("script-dot", ($formalpara//text())[1]);
```

as the default namespace is automatically used by element names, element templates (e.g. `<formalpara><title>`) and by XPath expressions, but not by attribute names and variable names (e.g. `script-dot`).

❻ Variable `formalpara` contains an empty `formalpara` element, specified using an *element template*. Note that element templates are instantiated verbatim. Therefore do not indent an element template.

Also note that without the "`TITLE HERE`" placeholder for the text of the title, the `formalpara`/`title` element would have been created without the customary empty `text()` child node, which is valid, but not user-friendly.

❼ Equivalent to `wrap-element($formalpara, .)`. This replaces the context node (`.`), which is selected `para`, by a `formalpara` containing selected `para`.

❽ Setting variable `script-selected` allows to specify the first node of the node selection. Here we want to explicitly select the newly created `formalpara`.

❾ Setting variable `script-dot` allows to specify the textual node containing the caret (also called "insertion cursor"). Here we want to move the caret inside the newly created `formalpara`.

Note that the above macro may be implemented much less efficiently by replacing the `script` child element of `macro` by the invocation of a `process` command. See Section 2.1, “Convert explicitly or implicitly selected `para` to a `formalpara`” [78].

Chapter 5. Process commands

1. Reference

```
<command
  name = NMTOKEN (optionally preceded by
  a command namespace in XMLmind XML Editor - Configuration and Deployment)
>
  Content: class | menu | macro | process
</command>

<process>
  showProgress = boolean : true
  debug = boolean : false

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

<info>
  Not documented.
</info>

<copyDocument
  to = Path
  selection = boolean : false
  preserveInclusions = boolean : false
  filterDuplicateIDs = boolean : true
  saveCharsAsEntityRefs = boolean : false
  indent = boolean : false
  encoding = NMTOKEN : 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
  /> |
  <attribute
    name = QName
```

```
    value = string
  /> |
  any element
</extract>

<resources
  include = NMTOKENS
  exclude = NMTOKENS
  match = Regexp pattern
  resolve = boolean : false
  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
  url = boolean
>
  Content: Parameter value
</parameter>

<parameterGroup
  name = Non empty token
/>

<copyProcessResources
  resources = anyURI | @anyURI | Glob pattern
  to = Path
  name = NMTOKEN
>
  Content: [ info ]?
</copyProcessResources>

<transform
  stylesheet = anyURI
  version = Non empty token : "1.0"
  cacheStylesheet = boolean : false
  file = Path
  pattern = boolean : false
  to = Path
>
  Content: [ info ]?
```

```
           [ 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
  encoding = any ASCII compatible encoding : "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
/>

<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 : .
  store = boolean : false
/>

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

<manifestFile>
  Content: Path
</manifestFile>

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

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

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

<invoke
  method = Qualified name of a Java static method
>
  Content: [ argument ]+
</invoke>

<argument>
  Content: string
</argument>

<subProcess
  name = NMTOKEN (optionally preceded
  by a command namespace in XMLmind XML Editor - Configuration and Deployment)
  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. Example 1: `*.*` matches `the_document.xml`, but does not match `the_document`. Example 2: `[a-z]*.html` matches `report.html`, but does not match `Report.html` (even on Windows where filenames are case-insensitive).

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 [65], read [67], invoke [73], subProcess [75].

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 of 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 [126] or command XXE.open [173] to replace part or all of the document being edited by the result of the XSLT transformation.

debug

If specified as `true`, this attribute prevents the command from deleting its work directory (`/tmp/xxe`*`NNNN`*`/`) at the end of the processing. This is useful if, for example, you need to look at the XSL-FO file generated by the transform [60] element of the process command.

1.2. Element `copyDocument`

```
<copyDocument
  to = Path
  selection = boolean : false
  preserveInclusions = boolean : false
  filterDuplicateIDs = boolean : true
  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
```

```
  /> |
  <attribute
    name = QName
    value = 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
`to`	Specifies the file where the document (or the node selection) is to be copied.

`selection`	If this attribute is specified with value `true` and if an element is explicitly selected, this element is saved to the specified location.
	
	If multiple nodes are explicitly selected, their parent element is saved and a special processing-instruction `<?select-child-nodes>`, specifying which nodes are selected, is added to the root element of the saved document.
	
	Example, the user has selected paragraphs with content 2, 3 and 4:
	
	`<div>`
	`  <?select-child-nodes 3-5?>`
	`  <p>1</p>`
	`  <p>2</p>`
	`  <p>3</p>`
	`  <p>4</p>`
	`</div>`
	
	In the above example, `3-5` is a node range intended to be tested using `position()`, the XPath built-in function. See Section 2.1, “Convert explicitly or implicitly selected `para` to a `formalpara`” [78] below to learn how to handle such multiple node selection in the XSLT style sheet.
	
	Otherwise, it is the whole document which is saved to the specified location.

`preserveInclusions`	If this attribute is specified with value `true` , the generated XML file contains • references to external entities, • transclusion directives (e.g. XInclude). Otherwise (default value), • references are replaced by the contents of the external entities,

Attribute	Description
	• transclusion directives (e.g. XInclude) are replaced by transcluded contents.
`filterDuplicateIDs`	Ignored unless `preserveInclusions` is set to `false`, that is, ignored unless the generated XML file contains transclusions. If this attribute is specified with value `true` (default value), an attempt is made to remove duplicate ID errors resulting from the presence of transcluded contents. This is done by adding a unique automatically generated suffix to these “false” duplicate IDs.
`saveCharsAsEntityRefs`	If this attribute is specified with value `true`, the generated XML file contains references to character entities such as `&eacute;` (if needed to and if such entities are defined in the DTD of the document being edited). Otherwise, the generated XML file contains character references such as `&#233;` (if needed to).
`indent`	If this attribute is specified with value `true`, the generated XML file is indented. Otherwise, the generated XML file is not indented.
`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
  /> |
  <attribute
    name = QName
    value = 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 three 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, replace the element containing image data by the attribute which is specified using the `attribute` child element of `extract`. This attribute is added to the parent element of the element containing image data.

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

  DocBook 5 example: replace embedded `db5:imagedata/svg:svg` by `db5:imagedata/@fileref`:

```
<cfg:extract xmlns=""
             xmlns:db5="http://docbook.org/ns/docbook"
             xmlns:svg="http://www.w3.org/2000/svg"
             xpath="//db5:imagedata/svg:svg" toDir="raw" >
  <cfg:attribute name="fileref"
                 value="resources/{$url.rootName}.png" />
</cfg:extract>
```

- OR, more general approach, insert a processing instruction (which is specified using 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.

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

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
  include = NMTOKENS
  exclude = NMTOKENS
  match = Regexp pattern
  resolve = boolean : false
  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 Section 10, “documentResources” in *XMLmind XML Editor - Configuration and Deployment*). DocBook example:

```
<cfg:documentResources xmlns="">
  <cfg:resource kind="image" path="//imagedata/@fileref"/>
  <cfg:resource kind="image" path="//graphic/@fileref"/>
  <cfg:resource kind="image" path="//inlinegraphic/@fileref"/>
  <cfg:resource kind="text" path="//textdata/@fileref"/>
  <cfg:resource kind="audio" path="//audiodata/@fileref"/>
  <cfg:resource kind="video" path="//videodata/@fileref"/>
</cfg:documentResources>
```

Note that elements replaced during an extraction step [51] specified by the `extract` element are never scanned for resources.

The default `resources` child elements are:

```
<resources match="^[a-zA-Z][a-zA-Z0-9.+-]*:/.+" />
<resources match=".+" copyTo="." />
```

Attributes of the `resources` child element specifying how to match a resource:

`match`

For each resource of the document specified by 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 `resolve`, `copyTo` or `referenceAs` attribute, the matched resource is ignored. For example, rule `<resources match="^[a-zA-Z][a-zA-Z0-9.+-]*:/.+"/>` is designed to ignore resources of any kind having an absolute URL.

`include`

This attribute contains one or more kinds of resources separated by whitespace. Example related to the above DocBook example: `include="image"`.

Unless the resource being processed has been given a kind and unless this kind is referenced in attribute `include` of element `resources`, the action corresponding to element `resources` is skipped.

`exclude`

This attribute contains one or more kinds of resources separated by whitespace. Example related to the above DocBook example: `exclude="text image"`.

If the resource being processed has been given a kind and if this kind is referenced in attribute `exclude` of element `resources`, the action corresponding to element `resources` is skipped.

Attribute `exclude` has priority over attribute `include`.

Attributes of the `resources` child element specifying an action on the matched resource:

`resolve`

If `resolve="true"`, attributes `copyTo` and `referenceAs` are ignored. Instead, in the copy of the document, the relative URI of the matched resource is replaced by its equivalent absolute URI.

Example:

```
<resources include="text" match=".+"
           resolve="true"/>
```

Let's suppose document `file:///docs/doc.xml` references text resource `examples/sample1.txt`. The copy of the document will reference absolute URI `file:///docs/examples/sample1.txt`.

`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"/>
```

Let's suppose the document references resource `images/logo.jpg`. File `logo.jpg` will be copied to `resources/logo.jpeg` and the copy of the document will reference `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 may 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 a format supported by the target XSL-FO processor.

DocBook example:

```
<process>
  <mkdir dir="resources"/>
  <mkdir dir="raw"/>

  <copyDocument to="__doc.xml">
        <resources match="^[a-zA-Z][a-zA-Z0-9.+-]*:/.+"/>

    <resources include="text" match=".+"
               resolve="true"/>

    <resources include="image" match=".+\.(png|jpg|jpeg|gif)"
               copyTo="resources"/>
    <resources include="image" match="(?:.+/)?(.+)\.(\w+)"
               copyTo="raw" referenceAs="resources/$1.png"/>

    <resources exclude="text image" 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
  url = boolean
>
  Content: Parameter value
```

```
</parameter>

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

Converts between image formats using any of the image toolkit plug-ins[1] 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.

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 needs to be converted, the value of the `to` attribute must be a directory name.

Examples:

```
<convertImage from="resources/logo.tiff" to="resources/xmlmind.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.

[1]Image toolkit plug-ins are generally written in the Java™ programming language. However, the `imageToolkit` configuration element (see Section 15, “imageToolkit” in *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.

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

lenient

Unless this attribute is specified with value `true`, a 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. The parameters supported by the built-in Java image toolkit are documented below. The documentation of the parameters supported by other image toolkits is displayed in the documentation pane of the dialog box of **Options → Install Add-ons**.

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, 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.

1.4. Element `copyProcessResources`

```
<copyProcessResources
  resources = anyURI | @anyURI | Glob pattern
  to = Path
  name = NMTOKEN
>
  Content: [ info ]?
</copyProcessResources>

<info>
  Not documented.
</info>
```

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.

Format of *`list-in-a-text-file`*

Such text files should be encoded in UTF-8.

The paths contained in such files should be separated by newline characters (`'\n'`). Open lines and lines starting with `'#'` are ignored.

Relatives paths are relative to the location of the text file.

Example:

```
# This is a comment.
resources/basic.css

resources/attention.png
resources/caution.png
resources/tip.png
```

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

name
: Giving a name to a process resource allows to easily replace it by a custom one. When a `name` attribute has been specified, the value of the `resources` attribute is preferably taken from the system property called "*`process_command_name`*`.resource.`*`name`*", if such system property exists and is not empty.

  DocBook 5 example: process command `db5.toHTMLHelp` is specified as follows:

```
<command name="db5.toHTMLHelp">
  <process>
    <subProcess name="db5.convertStep1" />
```

```
    <copyProcessResources resources="xsl/css/htmlhelp.css"
                          to="htmlhelp.css" name="css" />

    <transform stylesheet="xsl/htmlhelp/htmlhelp.xsl"
    ...
  </process>
</command>
```

Therefore defining system property db5.toHTMLHelp.resource.css allows to replace the stock `htmlhelp.css` by a custom CSS style sheet. Example:

```
<property name="db5.toHTMLHelp.resource.css" url="true">fancy.css</property>
```

(Remember that a system property can defined in a configuration file by using the `property` configuration element. See Section 22, “property” in *XMLmind XML Editor - Configuration and Deployment*.)

1.5. Element `transform`

```
<transform
  stylesheet = anyURI
  version = Non empty token : "1.0"
  cacheStylesheet = boolean : false
  file = Path
  to = Path
  pattern = boolean : false
>
  Content: [ info ]?
           [ parameter | parameterGroup ]*
</transform>

<info>
  Not documented.
</info>

<parameter
  name = Non empty token
  url = boolean
>
  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.

version

Specifies the version of the XSLT style sheet and hence, which XSLT engine to use. Default value is "1.0". The only other supported value is "2.0".

If `version="1.0"`, the bundled Saxon 6.5.5 XSLT 1 engine is used.

If `version="2.0"`, the bundled Saxon 9.2+ XSLT 2 engine is used.

A version number may be followed by one or more fully qualified method names separated by semicolons (';'). Example: "`2.0;com.xmlmind.ditac.xslt.ExtensionFunctions.registerAll`".

Such methods are used to configure a `javax.xml.transform.TransformerFactory` prior to using it. They are typically used to add extension functions to Saxon 9.2+.

Such methods must be static and must have the following signature: `void` *`method_name`*`(TransformerFactory` *`factory`*`) throws Exception`.

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.

When `pattern="false"`, the value of the `file` attribute is expected to a simple file path. However, it's also possible to specify a glob pattern, but in such case, *pattern mode* [61] is *not* enabled and this glob pattern must match exactly one file.

to

Output file or directory. When to specifies a directory, the basename of the output file is taken from the input file.

pattern

If this attribute is specified as `true`, `transform` operates in “pattern mode” and a glob pattern is allowed in the `file` attribute. Pattern mode is best explained by the two following examples:

Example 1: `file="*.xml"`, `to="temp"`. Glob pattern `*.xml` matches `foo.xml` and `bar.xml`. Attribute value `temp` specifies an existing directory. The XSLT transform will be invoked twice, first time with `foo.xml` as its input and the second with `bar.xml` as its input. The first time, the transform will generate `temp/foo.xml` and the second time it will generate `temp/bar.xml`.

Example 2: `file="*.xml"`, `to="any.htm"`. Glob pattern `*.xml` matches `foo.xml` and `bar.xml`. The XSLT transform will be invoked twice, first time with `foo.xml` as its input and the second with `bar.xml` as its input. The first time, the transform will generate `foo.htm` and the second time it will generate `bar.htm`.

`Parameter` and/or named `parameterGroup` child elements are used to parametrize the XSLT style sheet. Example: `<parameter name="paper.type">A4</parameter>`. Such parameters are described in the documentation of the XSLT style sheets (e.g. DocBook XSL Stylesheet Documentation).

If a `transform` element references a `parameterGroup`, this means that a `parameterGroup` configuration element (see Section 23, “parameterGroup” in *XMLmind XML Editor - Configuration and Deployment*) with the same name is defined elsewhere in this configuration file or in another configuration file. Note that it is not an error to reference a `parameterGroup` for which the configuration element is not found. Such reference to a possibly non-existent `parameterGroup` is useful as a placeholder.

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 XXE 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 Section 22, “property” 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 *`XXE_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="docbook-config: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.

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 by Bob Stayton.

1.6. Element `processFO`

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

<parameter
  name = Non empty token
  url = boolean
>
  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 to the XSL-FO processor in order to parametrize its behavior. These parameters are described in the documentation of the XSL-FO processors.

If the `url` attribute of a `parameter` child element is specified and its value is `true`, the parameter value must be a relative or absolute URL (properly escaped like all URLs). In such case, the value of the parameter is the fully resolved URL.

XFC example:

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

XEP example:

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

In addition to actual parameters, some XSL-FO processors may also support a number of *pseudo-parameters*. The documentation of these pseudo-parameters is displayed in the documentation pane of the dialog box of **Options** → **Install Add-ons**.

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>
    <parameter name="configuration" url="true">fop.xconf</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 value of the `file` attribute is generally a simple file path. However it's also possible to specify a glob pattern, but in such case, this glob pattern must match exactly one file.

  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 any of the matched files is a directory, it will be recursively copied. Has no effect is specified glob pattern does not match any file or directory.

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

1.8. Element `post`

```
<post
  url = anyURI
  encoding = any ASCII compatible encoding : "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. More precisely, the `post` element implements HTML5 — Form submission — Multipart form data using emulated `input type="text"` and `input type="file"` form fields.

The body of the `POST` request is encoded as "`multipart/form-data`" if the `post` element contains at least one `file` descendant element. Otherwise this body is encoded as "`application/x-www-form-urlencoded`".

Optional attribute `encoding` specifies the character encoding used by the form submission algorithms.

Always specify the name of *an ASCII compatible encoding* (`ISO-8859-1`, `UTF-8`, `Windows-1252`, etc) in the `encoding` attribute.

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.

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"
      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>

<post url="http://www.acme.com/login"
      encoding="UTF-8" readResponse="true">
  <field name="username">
    <value>admin</value>
  </field>
  <field name="password">
    <value>changeit</value>
  </field>
</post>
```

1.9. 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.

1.10. 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.

1.11. 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.

1.12. 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.

1.13. 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`).

1.14. Element `zip`

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

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

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

When specified with value `true`, the `store` attribute of the `add` child element allows to add entries to a Zip archive without compressing them.

See also jar [70].

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/*" store="true" />
  <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.

1.15. Element `jar`

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

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

<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 [69], 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" store="true" />
  <manifest>
    <attribute name="Master-Document">doc.xml</attribute>
    <attribute name="Publication-Date">%0</attribute>
    <attribute name="Self-Contained"></attribute>
  </manifest>
</jar>
```

1.16. 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 [75]).

Specified command may reference helper applications [72] declared using the **Preferences** dialog box, **Helper Applications** section.

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.

Using helper applications in commands interpreted by the native shell

This applies to the shell [71] element of a process [44] command as well as to the run [139] and start [159] commands.

Instead of containing substring "`notepad foo.txt`", a command line, interpreted by **cmd.exe** on Windows and **/bin/sh** on Unix, may contain something like "`helper(text/plain) foo.txt`" or "`helper(txt) foo.txt`" or even "helper() foo.txt".

In the above example, substring "`helper(`*`spec`*`) `*`path`*" refers to a *helper application* declared using the **Preferences** dialog box, **Helper Applications** section.

This preferences sheet allows to associate helper applications to file types. In the above example, we assume that plain text files, that is files having MIME type "`text/plain`" or having a "`txt`" filename extension, have been associated to helper application "`notepad "%F"`".

Examples of command lines making use of helper applications:

```
<command name="run"
         parameter="helper(text/plain) '%D'"/>

<command name="start"
         parameter="helper(defaultViewer) '%_'"/>

<shell command="helper(.hhp) htmlhelp.hhp || exit 0"/>

<shell command="helper(application/x-java-help-index) ."/>
```

In order to use helper applications, a command line must contain substrings having this syntax: "`helper(`*`spec`*`) `*`path`*".

spec

Specifies which helper application to use. It may be:

- A MIME type. Example: "`text/plain`".
- A filename extension, with or without a leading "`.`". Example: "`.hhp`".
- Fixed string "`defaultViewer`", which is the helper application specified in **Preferences** dialog box, **Helper Applications** section, **Default viewer** field. This default viewer is generally a Web browser.
- Empty. In which case, the filename extension of *`path`* is used.

path

Must always follow the `helper()` construct. This absolute or relative filename or URL may be quoted using single or double quotes if it happens to contain whitespace characters.

When a command line contains a "`helper(`*`spec`*`) `*`path`*" substring, this substring is substituted with the corresponding helper application. How this is done is best explained using an example.

Let's suppose the command line is "`helper(defaultViewer) 'foo.html'`".

Let's suppose the default viewer is specified as: "`(mozilla -remote "openURL(%U)" 1> /dev/null 2>&1) || (mozilla "%U" &)`".

Single quotes are stripped from path `'foo.html'` and each occurrence of `%U` (or `%F`) in the helper application is replaced by this path (without any other added value).

This gives: `"(mozilla -remote "openURL(foo.html)" 1> /dev/null 2>&1) || (mozilla "foo.html" &)"` (which, in this case, cannot work because `foo.html` is not an absolute URL).

1.17. Element `invoke`

```
<invoke
  method = Qualified name of a Java static method
>
  Content: [ argument ]+
</invoke>

<argument>
  Content: string
</argument>
```

Invokes specified Java™ *static* method, passing it specified arguments.

- 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.util.Console console);
    ```
  - ```
    method(java.lang.String[] arguments,
           java.io.File workingDir,
           com.xmlmind.util.Console console,
           com.xmlmind.xml.doc.Document docBeingEdited);
    ```

arguments

An array of Strings corresponding to the contents of the `argument` child elements. Note that the macro-variables (`%0`, `%1`, `%D`, `%p`, `%C`, etc) are substituted with their values in each argument.

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 [67], for use in a macro command for example).
- The method may throw any exception.

Examples:

```
<invoke method="TestInvoke.echo">
  <argument>args={%*}</argument>
  <argument>doc='%D'</argument>
  <argument>pwd='%W'</argument>
  <argument>conf='%C'</argument>
</invoke>

<invoke method="TestInvoke.echo2"/>

<invoke method="TestInvoke.gzip">
  <argument>__doc.xml</argument>
</invoke>
```

Static methods invoked by the above examples:

```
public final class TestInvoke {
    public static void echo(String[] arguments, File workingDir,
                            Console console) {
        console.showMessage("arguments={" +
                            StringUtil.joinArguments(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 {
        if (arguments.length != 1)
            throw new IllegalArgumentException("arguments");

        File inFile = new File(workingDir, arguments[0].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;
    }
}
```

1.18. Element `subProcess`

```
<subProcess
  name = NMTOKEN (optionally preceded by
  a command namespace)
  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).

Example: the following process command is used to convert a DocBook document to PostScript® or to PDF.

```
  <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>
```

1.19. 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
%0, %1, %2, ..., %9, %*	A process-command can have a parameter. This string is split like in a command line. A part of the split parameter can be referenced as variable %0, %1, %2, ..., %9. %* can be used to reference the whole parameter of the process-command.
%D, %d	%D is the file name of the document being edited. Example: C:\novel\chapter1.xml. This variable is replaced by an empty string if the document being edited is found on a remote HTTP or FTP server. %d is the URL of the document being edited. Example: file:///C:/novel/chapter1.xml.
%P, %p	%P is the name of the directory containing the document being edited. Example: C:\novel. This variable is replaced by an empty string if the document being edited is found on a remote HTTP or FTP server. %p is the URL of the directory containing the document being edited. Example: file:///C:/novel. Note that this URL does not end with a '/'.
%N, %R, %E	%N is the base name of the document being edited. Example: chapter1.xml. %R is the base name of the document being edited without the extension, if any (sometimes called the root name). Example: chapter1. %E is the extension of the document being edited, if any. Example: xml. Note that the extension does not start with a '.'.
%n, %r, %e	Similar to %N, %R, %E except that these variables contain properly escaped URI components. For example if %R contains "foo bar", then %r contains "foo%20bar".
%S	%S is the native path component separator of the platform. Example: '\' on Windows.
%U	User's account name. Example: john.
%H, %h	%H is the user's home directory. Example: /home/john. %h is the URL of the user's home directory. Example: file:///home/john. Note that this URL does not end with a '/'.
%A, %a	%A is the user's preferences directory. Example: /home/john/.xxe10. %a is the URL of the user's preferences directory. Example: file:///home/john/.xxe10. Note that this URL does not end with a '/'.

Variable	Description
%X, %x	%X is the name of the user's current working directory (that is, the current working directory of XXE). Example: C:\Users\john\Documents\report. %x is the URL of the user's current working directory. Example: file:///C:/Users/john/Documents/report. Note that this URL does not end with a '/'.
%W, %w	%W is the name of the temporary process directory. Example: C:\temp\xxe1023E45. %w is the URL of the temporary process directory. Example: file:///C:/temp/xxe1023E45. Note that this URL does not end with a '/'.
%C, %c	%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. %c is the URL of the above directory. Example: file:///C:/Program%20Files/XMLmind_XML_Editor/addon/config/docbook. Note that this URL does not end with a '/'.

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 [77]).

In addition to the above variables, a process command may reference any Java™ system property or environment variable. Examples: %{user.home} (for system property user.home), %{HOME} (for environment variable HOME)

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 %20.

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" *should* be specified as "%%20". However in practice there is no need to do so because variable %20 is almost never defined and a reference to a variable which is not defined is left as is.

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

"`docs/my%20report/my%20doc.%0`" works fine too as long as the macro-command or the process commands is passed less than 21 arguments.

2. Commented examples

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

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

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

In the above example, `toFormalpara.xsl` is simply:

```
<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">
  <formalpara>
    <xsl:for-each select="@*">
      <xsl:copy/>
    </xsl:for-each>

    <title></title>
    <para>
      <xsl:apply-templates select="node()"/>
    </para>
  </formalpara>
</xsl:template>

<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
```

```
</xsl:template>

</xsl:stylesheet>
```

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() &gt;= $first and
                                        position() &lt;= $last]">
      <xsl:apply-templates select="." />
    </xsl:for-each>

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

The above macro may be implemented much more efficiently by replacing the invocation of process command `toFormalpara` by a `script` child element of `macro`. See Example 4.8, “Convert a `DocBook 5 para` to a `formalpara`” [42].

2.2. Convert a DocBook document to RTF

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

<command name="docb.toRTF">
  <process>
    <mkdir dir="resources" />❷
    <mkdir dir="raw" />❸
    <copyDocument to="__doc.xml">❹
      <resources match="(https|http|ftp)://.*" />❺
      <resources match=".+\.(png|jpg|jpeg|gif)"
                 copyTo="resources" />❻
      <resources match="(?:.+/)?(.+)\.(\w+)"
                 copyTo="raw" referenceAs="resources/$1.png" />❼
```

```
      <resources match=".+"
                 copyTo="resources" />
    </copyDocument>

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

    <mkdir dir="images/callouts" />❾
    <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">❿
      <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" />⓫
    </transform>

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

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

❶ 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.

❷ 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/`.

❸ 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.

❹ 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.

❺ References to really absolute resources are not modified in the copy of the document.

❻ References to PNG, GIF, JPEG graphics files are modified to point to the copies which are made in directory `resources/`.

❼ 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 that PNG, GIF, JPEG are copied as is in directory `raw/`, waiting to be converted.

❽ Converts all images found in directory `raw/` to PNG images created in directory `resources/`.

❾ Copies resources internally used by the `xsl/fo/docbook.xsl` XSLT style sheet to where the FO processor can find them.

❿ Transforms the copy of the document `__doc.xml` to XSL-FO file `__doc.fo`.

⓫ 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>
```

⓬ Convert XSL-FO file `__doc.fo` to local RTF file `__doc.rtf`.

⓭ Copies local RTF file `__doc.rtf` to its user-specified destination.

The element is called `upload` because 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

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 [79], 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.

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.

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 Section 20, “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 Section 12, “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.

Notation used in the synopsis of a command parameter:

- In the synopsis of a command parameter, `S` means space.
- These non-terminals are sometimes used in the synopsis of a command parameter:

```
implicit_selection -> '[implicitNode]' | '[implicitElement]'

argument_node -> '#text' |
                 element_name |
                 '#template(' element_name ',' template_name ')'

element_name -> Name | '{' namespace_URI '}' NCName

namespace_uri -> anyURI
```

Common pitfalls to be avoided

- **A command parameter is*not*namespace-aware**

  Namespace prefixes cannot be used inside a command parameter. Notation `{`*`namespace_URI`*`}`*`local_name`* —the so-called James Clark's notation— 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.

- **Whitespace is significant in a command parameter**

  Most notably, whitespace is not allowed inside the `#template()` construct.

  Let's use command insert [116] as an example. The following command parameters *cannot* work:

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

  While the following command parameters should be OK:

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

1. alert

Parameter syntax:

```
[ '[ERROR]' | '[WARNING]' | '[INFO]']? message
```

Displays an alert dialog box containing the message specified by command parameter. Option `[ERROR]`, `[WARNING]` or `[INFO]` specifies the type of the alert dialog box. Defaults to `[INFO]`.

This command is useful to debug macro-commands.

Examples:

```
alert Hello, world!

alert [ERROR]Syntax error.
```

2. add

Parameter syntax:

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

Adds argument node [84] before or after selected node [84]. 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 [86].

3. addAttribute

This command is similar to `putAttribute` [131] except that it will not replace the attribute if it is already set on the selected element.

4. addBlockInFlow

Parameter syntax:

```
[ '['inline_container_element_name']' S ]? block_element_name|block_element_template
```

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

`block_element_name`, *`block_element_template`*

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.

When this optional parameter is missing, the list of inline elements is parsed from the value of system property "*`configuration_name`* inlineElementNames". This property may contain a list of fully qualified element names and/or "`container(`*`container_element_name`*`)`" shorthands. See TEI example below.

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*".

Example 6.1. XHTML example

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

Generic command "`add [86] after[implicitElement] {http://www.w3.org/1999/xhtml}pre`" gives:

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

Smarter command "`addBlockInFlow [p] {http://www.w3.org/1999/xhtml}pre`" gives:

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

Examples:

```
addBlockInFlow [p] table
addBlockInFlow [p] #template({http://www.w3.org/1999/xhtml}table,head_row)
```

Example 6.2. TEI example

```
<property name="$c inlineElementNames">
  {http://www.tei-c.org/ns/1.0}abbr
  {http://www.tei-c.org/ns/1.0}add
  ...
  container({http://www.tei-c.org/ns/1.0}pc)❶
  ...
  {http://www.tei-c.org/ns/1.0}w
</property>
```

```
addBlockInFlow❷ {http://www.tei-c.org/ns/1.0}table
```

❶ `container({http://www.tei-c.org/ns/1.0}pc)` means any child element of `{http://www.tei-c.org/ns/1.0}pc`.

❷ Because property "*$c* inlineElementNames" is defined, parameter *`inline_container_element_name`* is not needed by `addBlockInFlow`.

5. autoSpellChecker

Parameter syntax:

```
toggle|on|off|isOn|popupMenu
```

Allows to activate and deactivate the automatic spell checker.

Options:

toggle
: if the automatic spell checker is active, the command deactivates it. If the automatic spell checker is not active, the command activates it.

on
: Ensures that the automatic spell checker is active.

off
: Ensures that the automatic spell checker is not active.

isOn
: Returns `Boolean.TRUE` if the automatic spell checker is active; otherwise returns Boolean.FALSE.

popupMenu
: "`autoSpellChecker popupMenu`" cannot be bound to a keystroke, only to a mouse click. When the mouse is clicked on a misspelled word (that is, underlined in red), "`autoSpellChecker popupMenu`" displays a popup menu which allows to correct its spelling.

When the `toggle`, `on`, `off` or `isOn` option has been specified, this command returns a `Boolean` indicating whether the automatic spell checker is active.

6. beep

No parameter.

Emits an audio beep.

This command is useful to write macro-commands.

7. cancelSelection

No parameter.

Cancels text or node selection if any.

8. center

No parameter.

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

9. checkExternalRefs

No parameter.

Checks all the links to external resources found in the document being edited. All kinds of external resources are checked for existence: images, audio, video, PDF documents, HTML pages, etc.

When the resource is an HTML page and the link ends with a fragment (e.g. "`#bar`" in "`../doc/foo.html#bar`") then this fragment is also checked for existence:

- Any file having a file extension containing substring "`htm`" or "`xht`" is considered to be an HTML page.
- The fragment is searched in `id` attributes and `a`/`name` attributes.

A property called "*`configuration_name`* checkExternalRefs" must be defined for this command to work. This is done by the means of the `property` in *XMLmind XML Editor - Configuration and Deployment* configuration element.

This multi-line property is used to specify which elements or attributes are links to external resources. Syntax:

```
spec -> [ namespace_prefix ]* & [ xpath ]* & [ directive ]?

namespace_prefix -> 'xmlns:' prefix '=' namespace \n

xpath -> expression_selecting_elements_or_attributes \n

directive -> '+documentResources' \n
```

Directive `+documentResources` adds all the XPath expressions found in configuration element `documentResources` in *XMLmind XML Editor - Configuration and Deployment* to the list of XPath expressions directly specified in the "`$c` checkExternalRefs" property.

Example 6.3. XHTML checkExternalRefs property

```
<property name="$c checkedExternalRefs">
  +documentResources

  xmlns:h=http://www.w3.org/1999/xhtml
  //h:a/@href
  /h:html/h:head/h:base/@href
</property>
```

Example 6.4. DITA checkExternalRefs property

```
<property name="$c checkedExternalRefs">
  //xref[@scope="external"]/@href
  +documentResources
</property>
```

Example 6.5. TEI Lite checkExternalRefs property

```
<property name="$c checkedExternalRefs">
  xmlns:tei=http://www.tei-c.org/ns/1.0
  //tei:ref/@target[not(starts-with(., '#'))]
  //tei:ptr/@target[not(starts-with(., '#'))]

  +documentResources
</property>
```

Example 6.6. DocBook 5+ checkExternalRefs property

```
<property name="$c checkedExternalRefs">
  xmlns:db=http://docbook.org/ns/docbook
  xmlns:xlink=http://www.w3.org/1999/xlink
  //db:link/@xlink:href
  +documentResources
</property>
```

10. checkValidity

```
[ 'filterDuplicateIDs' ]? [ 'commitChanges' ]?
```

Checks the validity of the document and, if validity errors are found, displays a modal dialog box (similar to the **Validity** tool) listing them.

Options:

`filterDuplicateIDs`
: If specified, false duplicate ID errors due to the multiple inclusions of the same elements are removed from the list displayed to the user.

`commitChanges`
: If specified, send a `com.xmlmind.xmledit.edit.CommitChangesEvent` event to the document before checking its validity. This event is typically used by controls (for example, a Java™ Swing text field) embedded in the view of the document. This event instructs these controls that they must now commit their changes (for example, last characters typed in the text field).

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

11. commentOut

Parameter syntax:

```
[ implicit_selection ]? |
'uncomment' [ '[implicitNode]' ]?
```

Without the `uncomment` option, replace text or node selection by a comment containing this selection.

With the `uncomment` option, it's the inverse operation: parse the content of selected comment and replaces this comment by parsed XML nodes.

Examples:

```
commentOut
commentOut [implicitElement]
commentOut uncomment
commentOut uncomment[implicitNode]
```

12. confirm

Parameter syntax:

```
[ choices ]? message

choices -> single_char_delimiter ( 'single' '|' 'multiple' )
           single_char_delimiter [ choice ]+

choice -> [ '*' ]? choice_value '=' choice_text single_char_delimiter
```

This command is useful when writing interactive macro-commands. It displays a dialog box containing specified *`message`* and requesting the user to confirm an action (generally the command which follows this `confirm` command in the macro). 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.

Some *`choices`* may be specified to let the user select one or more options before clicking the **OK** button.

Options are separated using a single character. This delimiter must be chosen in order to be absent from *`choice_value`*, *`choice_text`* and *`message`*. It's typically `'|'`, `'/'`, `'^'`, `'~'`, or `';'`.

A *`choice`* comprises an option value and some short text describing the option. The value of an option which is turned on is immediately preceded by `'*'`. Examples: `"*yes=Yes, I want this!"` and `"no=No, please don't."`.

Keyword '`single`' specifies that the user may select just one option. Keyword '`multiple`' specifies that the user may select zero or more options.

When some *`choices`* have been specified and the user clicks the **OK** button, this commands returns the values of all selected options separated by a space character.

Examples:

```
confirm Convert selected text to computeroutput?
confirm |single|*doc=Entire document|sel=Selection only| Print
```

13. contextualMenu

No parameter.

Displays the right-click, contextual, popup menu. More information in Section 2, “Redefining or extending the right-click, contextual, popup menu” [21].

14. convert

Parameter syntax:

```
[ implicit_selection ]? [ S argument_node ]?
```

Converts text selection or selected nodes [84] to argument node [84].

Unlike replace [136] 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, and also all compatible attributes, are transferred to the result of the conversion.

  Example:

  ```
  <simpara id="p1">the <emphasis>little</emphasis> girl.</simpara>
  ```

  converted to `para` gives

  ```
  <para id="p1">the <emphasis>little</emphasis> girl.</para>
  ```

- 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 id="p1">the <emphasis>little</emphasis> girl.</simpara>
  ```

  can be converted to `blockquote` and that gives us

  ```
  <blockquote>
    <simpara>Once upon a time.</simpara>
    <simpara id="p1">the <emphasis>little</emphasis> girl.</simpara>
  </blockquote>
  ```

Note that when argument node [84] is an element template, command `convert` copies the attributes of the element template and ignores its child nodes.

Examples:

```
convert emphasis
convert [implicitElement] #text
convert #template({http://www.w3.org/1999/xhtml}span,highlight)
```

See also command wrap [166], a variant of command convert [92] which has a different behavior in the case of single element selection.

15. convertCase

Parameter syntax:

```
lower | upper | capital
```

If there is a text selection, this command converts the character case of this selection. If one or more nodes are selected, this command converts the character case of all the text contained in these nodes. If there is no selection of any kind, this command converts the character case from caret position to end of word, then it moves the caret to the next word.

lower
: All characters are converted to lowercase characters.

upper
: All characters are converted to uppercase characters.

capital
: First character of a word is converted to an uppercase character. The other characters are converted to lowercase characters.

16. copy

Parameter syntax:

```
[ implicit_selection ]?
```

Copies text selection or selected nodes [84] to system clipboard.

Example:

```
copy [implicitElement]
```

17. copyAsInclusion

Parameter syntax:

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

Copies explicitly selected nodes (or implicitly selected element if the `[implicitElement]` option is used) to system clipboard. Each node copied to the clipboard is marked as being a reference rather than plain XML data.

The `[multipleInstances]` option is a hint which indicates that the reference created by the command may be found several times, at different places, in the including document. Example: when you include a chapter in a book, there is no need to specify `[multipleInstances]`, but when you include boilerplate text like a product name or a company name, then it is recommended to specify `[multipleInstances]`. This is why the **Include** tool in *XMLmind XML Editor - Online Help* always uses option `[multipleInstances]` to create the references it pastes in a document.

Command `copyAsInclusion` will not work when one of the selected nodes is a reference or is an inclusion directive (e.g. an `xi:include` element).

Command `copyAsInclusion` cannot work unless the following conditions are met:

- Selected nodes are contained in a document associated to one or more inclusion schemes (e.g. XInclude, DITA conref).
- One of these inclusion schemes is capable of copying selected nodes as an inclusion.

Examples:

```
copyAsInclusion [implicitElement]
copyAsInclusion [multipleInstances]
copyAsInclusion [implicitElement][multipleInstances]
```

Commands `copyAsInclusion` [93] (generally bound to keystroke **Shift**+**Ctrl**+C) and `paste` [126] (generally bound to keystroke **Ctrl**+V) are used to compose *modular documents*, see the corresponding tutorial section in the User's Guide.

18. copyChars

Parameter syntax:

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

Copies characters found in text selection or in selected nodes [84] to system clipboard. Unlike command copy [93], 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 `li`, `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]
```

19. copyImage

No parameter.

If the explicitly selected element is rendered on screen as an image, the `copyImage` command copies this image to the system clipboard. This means that this command cannot function when the document being edited is displayed as a tree view (no style sheet)

20. cut

Parameter syntax:

```
[ implicit_selection ]?
```

Cuts text selection or selected nodes [84] to system clipboard.

Example:

```
cut [implicitElement]
```

21. declareNamespace

Parameter syntax:

```
[ xml="namespace"|xml:prefix="namespace"
  [ NEWLINE_CHAR xml="namespace"|xml:prefix="namespace" ]* ]?
```

Without any parameter, this command displays a modal dialog box (similar to the dialog displayed by **Tools** → **Declare Namespace**) which lets the user declare new name spaces and/or change the prefixes of existing name spaces.

The optional parameter specifies a namespace/prefix mapping and may be used to completely replace the current namespace/prefix mapping of the document being edited. However, in such case, this command will fail if the document is opened in read-only mode or if the document conforms to a DTD (the namespace/prefix mapping specified by a DTD is immutable).

22. demoteListItem

No parameter, but a property called "*configuration_name* listItemSpecification" should be defined.

Convert a paragraph to a list item and a list item to a paragraph, the new element having a greater nesting level than the original one. This command automatically creates lists or merges adjacent lists when needed to.

This is the inverse command of `promoteListItem` [131].

Note that `promoteListItem` and `demoteListItem` strictly alternate between paragraphs and list items. This means that you'll often have to invoke the command *twice in a row*. For example, in order to create a nested list, select a list item and then invoke `demoteListItem` twice. First invocation converts the list item to a plain paragraph contained in the preceding list item. Second invocation converts this paragraph to the first item of a new nested list.

For `promoteListItem` and `demoteListItem` to function, any of the following conditions should be met:

- A sequence of list items must be explicitly selected.
- A list must be explicitly selected. This is equivalent to selecting all its items.
- A sequence of blocks *starting with a paragraph* must be explicitly selected.
- A paragraph must be implicitly selected. In order to implicitly select a paragraph, move the caret anywhere inside it. However if this paragraph is the first child of a list item, then it's the list item which is implicitly selected.
- A list item to be implicitly selected. In order to implicitly select a list item, move the caret anywhere inside it.

22.1. Configuring the `promoteListItem` and `demoteListItem` commands

Commands `promoteListItem` and `demoteListItem` expect a property called "*`configuration_name`* listItemSpecification" to be defined. This is done by the means of the `property` in *XMLmind XML Editor - Configuration and Deployment* configuration element.

This property is used to specify what elements are used to represent paragraphs, list items and lists. Syntax:

```
spec -> inline_spec? paragraph_spec item_spec list_spec

inline_spec -> inline= [ name_list ]? \n

paragraph_spec -> paragraph= name_list \n

item_spec -> item= name_list \n

list_spec -> list= name_list \n

name_list -> name_list_item {S name_list_item}*

name_list_item -> name | container( name )

name = non_qualified_name | {namespace_URI}local_part
```

`inline=`

This field is optional. Specifies the names of the inline elements.

If the `inline=` field is specified, this means that a list item may directly contain text and inline elements in addition to paragraphs, nested lists and other block elements.

Because the list of inline elements can be quite large, it is possible use the `container(`*`x`*`)` construct to specify any child element of element *`x`*. Example: `container(shortdesc)` which means any child element of `shortdesc`.

If the `inline=` field is *not* specified and system property "*`configuration_name`* inlineElementNames" [87] is defined, then the list of inline elements is parsed from the value of this property. This property may contain a list of fully qualified element names and/or "`container(`*`container_element_name`*`)`" shorthands. See TEI example below.

Unlike the other fields, an empty `inline=` field is allowed and may be used to avoid parsing the value of system property "*`configuration_name`* inlineElementNames". See LwDITA example below.

`paragraph=`

Specifies the names of the elements which must be considered as being paragraphs.

`item=`

Specifies the names of the elements which must be considered as being list items.

A list item is expected to contain one or more of any of the paragraphs specified using the `paragraph=` field. A list item is also expected to contain nested lists and all sorts of other block elements in addition to paragraphs: blockquote, table, figure, etc.

`list=`

Specifies the names of the elements which must be considered as being lists.

A list is expected to be a general purpose itemized or ordered list. A list is expected to contain one or more of any of the items specified using the `item=` field.

Example 6.7. XHTML listItemSpecification

```
<property name="$c❶ listItemSpecification">
  inline=container(❷{http://www.w3.org/1999/xhtml}p)
  paragraph={http://www.w3.org/1999/xhtml}p❸
  item={http://www.w3.org/1999/xhtml}li
  list={http://www.w3.org/1999/xhtml}ul {http://www.w3.org/1999/xhtml}ol
</property>
```

❶ `$c` is a shorthand for "current configuration name".

❷ The `container()` construct here means: any child element of `p`, the XHTML paragraph.

❸ Namespace prefixes are not supported inside a property value. Hence, the name of an element must be specified using James Clark's notation [85].

Example 6.8. DITA listItemSpecification

```
<property name="$c listItemSpecification">
  inline=container(shortdesc) xref indexterm indextermref
  paragraph=p
  item=li
  list=ul ol
</property>
```

Or equivalently:

```
<property name="$c inlineElementNames">
  container(shortdesc)
```

```
  xref
  indexterm
  indextermref
</property>

<property name="$c listItemSpecification">
  paragraph=p
  item=li
  list=ul ol
</property>
```

Notice in this second listing, the implicit "`inline=`" field which is indirectly specified using system property "*$c* inlineElementNames".

Example 6.9. DocBook 5 listItemSpecification

```
<property name="$c listItemSpecification">
  paragraph={http://docbook.org/ns/docbook}para {http://docbook.org/ns/docbook}simpara
  item={http://docbook.org/ns/docbook}listitem
  list={http://docbook.org/ns/docbook}itemizedlist {http://docbook.org/ns/docbook}orderedlist
</property>
```

Example 6.10. TEI listItemSpecification

```
<property name="$c inlineElementNames">
  {http://www.tei-c.org/ns/1.0}abbr
  {http://www.tei-c.org/ns/1.0}add
  ...
  container({http://www.tei-c.org/ns/1.0}pc)
  ...
  {http://www.tei-c.org/ns/1.0}w
</property>

<property name="$c listItemSpecification">
  paragraph={http://www.tei-c.org/ns/1.0}p
  item={http://www.tei-c.org/ns/1.0}item
  list={http://www.tei-c.org/ns/1.0}list
</property>
```

Notice the implicit "`inline=`" field which is indirectly specified using system property "*$c* inlineElementNames".

Example 6.11. LwDITA listItemSpecification

```
<property name="$c inlineElementNames">
  container(shortdesc)
  xref
  indexterm
  indextermref
</property>

<property name="$c listItemSpecification">
```

```
  inline=
  paragraph=p
  item=li
  list=ul ol
</property>
```

Notice the empty "`inline=`" field which avoids parsing system property "*`$c`* inlineElementNames". (Unlike in full DITA, in LwDITA, an `li` element may not directly contain text and phrase elements..)

23. delete

Parameter syntax:

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

Deletes text selection or selected nodes [84].

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

Example:

```
delete force[implicitElement]
```

24. 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.

25. deleteSelectionOrDeleteChar

Parameter syntax:

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

- If there is a text or node selection, this selection is deleted.
- If the caret is inside an empty text node, this empty text node is deleted. If the empty text node is the sole child of its parent element, it is the parent element as a whole which is deleted.
- Otherwise, delete the character found after the caret. If option `backwards` is specified, delete the character found before the caret.

This command is intended to be bound to keys **Del** and BackSpace. See also `deleteSelectionOrJoinBlockOrDeleteChar` [100].

26. deleteSelectionOrJoinBlockOrDeleteChar

Parameter syntax:

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

- If there is a text or node selection, this selection is deleted.
- If the caret is inside an empty text node, this empty text node is deleted. If the empty text node is the sole child of its parent element, it is the parent element as a whole which is deleted.
- If the caret is found at the very end of a *block*, this block is merged with the immediately following, similar, block (if any).

  To make it simple, a block is a paragraph or a list item. However what is exactly a block must be specified as explained in Section 52.1, “Specifying splittable blocks” [122].

  If option `backwards` is specified and the caret is found at the very beginning of a block, then this block is merged with the immediately preceding, similar, block (if any).
- Otherwise, delete the character found after the caret. If option `backwards` is specified, delete the character found before the caret.

This command is intended to be bound to keys **Del** and BackSpace. See also `deleteSelectionOrDeleteChar` [99].

27. deleteWord

Parameter syntax:

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

Deletes the word following the caret in a textual node (text, comment, processing-instruction). If the `backwards` option is used, this command deletes the word preceding the caret.

Note that this command also deletes the whitespace after or before the word if needed to. That is, it will attempt not to leave superfluous whitespace between words.

28. diffSupport

```
'on' | 'off' | 'toggle' | 'state' |
'revisions.on' | 'revisions.off' | 'revisions.toggle' | 'revisions.state'
```

Adds or removes information related to revision support to/from the document being edited.

`on`

Enable the comparison of revisions of the document being edited. No effect if this comparison is already enabled.

This is implemented by adding a unique serial number to all the elements of the document. Such serial numbers are saved as `<?xxe-sn>` processing instructions. More information in Section 2, “Elements are given serial numbers” [183].

`off`

Disable the comparison of revisions of the document being edited. No effect if this comparison is already disabled.

This is implemented by removing their serial numbers from all the elements of the document.

`toggle`

Toggle the comparison of revisions of the document being edited.

`state`

Returns "`on`" if the comparison of revisions is enabled. Returns "`off`" otherwise.

`revisions.on`

Start storing all the revisions of the document being edited in the XML file containing this document. No effect if this option has already been chosen.

This is implemented by computing the binary delta (VCDIFF algorithm) between a normalized XML representation of the document being edited and a normalized XML representation of the document at the beginning of the editing session (that is, before any modification). Such binary deltas are base-64 encoded in order to be saved in a single `<?xxe-revisions>` processing instruction found at the end of the document. More information in Appendix B, *Format of the revision history* [185].

Option `revisions.on` implies option `on`. If the option of storing all the revisions of the document being edited has been chosen, then it is also possible to compare the revisions of this document.

Option `revisions.off` does not implies option `off`. Stopping to store all the revisions of the document being edited does not mean that you'll no longer be able to compare the revisions of this document. For that, you'll have to use option `off` because option `off` implies option `revisions.off`.

`revisions.off`

Stop storing all the revisions of the document being edited in the XML file containing this document. No effect if this option has already been chosen.

`revisions.toggle`

Toggle the storage of revisions in the XML file containing the document being edited.

`revisions.state`

Returns "`revisions.on`" if the option of storing all the revisions of the document being edited has been chosen. Returns "`revisions.off`" otherwise.

Examples:

```
diffSupport on
diffSupport state
diffSupport revisions.toggle
```

29. drag

No parameter.

This command is intended to be bound to a mouse input. It is generally used by the macro-command which is bound to the `drag appEvent` in *XMLmind XML Editor - Configuration and Deployment*. See example [106].

If the mouse is clicked anywhere inside the node or text selection, this command returns a string representation of the selection. Otherwise this command first selects the element clicked upon and then returns a string representation of the newly selected element.

The returned string may be quickly described as follows:

- It the text selection consists only in plain text, this command returns this plain text.
- If the node selection consists only in a single element, this command returns an XML representation of this element. This XML representation always starts with `<?xml version="1.0"?>`.
- Otherwise, this command returns an XML representation of the node range in the format of the clipboard (that is, starting with `<?xml version="1.0"?>` and using a `<ns:clipboard xmlns:ns="http://www.xmlmind.com/xmleditor/namespace/clipboard">` wrapper element).

See also ensureSelectionAt [105].

30. drop

Parameter syntax:

```
string
```

This command is intended to be bound to a mouse input. It is generally used by the macro-command which is bound to the `drop appEvent` in *XMLmind XML Editor - Configuration and Deployment*.

If the string parameter is an URL or an absolute filename, this command opens the corresponding document using command XXE.open [173].

Otherwise, the string parameter is assumed to be plain text or the string representation of one or more XML nodes. In this case, the `drop` command displays a popup menu allowing to copy or move the data being dropped.

Note that only the current text or node selection can be moved within a document. Any data other than the current text or node selection can be copied, but not moved. Moreover, the current text or node selection cannot be moved between two different documents.

DITA topic example:

```
<binding>
  <appEvent name="drop" />
  <command name="dita.drop" parameter="%{value}" />
</binding>

<command name="dita.drop">
  <macro>
    <choice>
```

```
      <sequence>
        <match context="$clickedElement"
               pattern="xref[@href]|xref[@href]//*|
                        link[@href]|link[@href]//*|
                        longdescref[@href]|
                        longquoteref[@href]" />
        <set variable="selectedElement" context="$clickedElement"
             expression="(ancestor-or-self::*[@href])[last()]" />

        <set variable="dropped" context="$selectedElement"
             expression="relativize-uri(uri-or-file-name('%*'))" />
        <get expression="$dropped" />
        <command name="putAttribute" parameter="href '%_'" />

        <get expression="$dropped" />
        <command name="status" parameter="href='%_'" />
      </sequence>

      <!-- Default drop action. -->
      <command name="drop" parameter="%*" />
    </choice>
  </macro>
</command>
```

31. editAttribute

This command is similar to `putAttribute` [131] except that it returns the attribute value as a string instead of setting the attribute on the selected element. This command is only useful to write macro commands.

Macro command example: set attribute `style`, but when the value chosen by the user is the empty string, remove attribute `style` (if any):

```
<command name="setStyle">
  <macro>
    <sequence>
      <command name="editAttribute"
               parameter="[implicitElement] style" />
      <set variable="style" expression="%_"
           plainString="true" />

       <choice>
         <sequence>
          <test expression="not($style)" />
          <command name="removeAttribute"
                   parameter="[implicitElement] style" />
         </sequence>

         <sequence>
          <get expression="replace($style, &quot;'&quot;, &quot;\\'&quot;, l)" />❶
          <command name="putAttribute"
                   parameter="[implicitElement] style '%_'" />❷
```

```
        </sequence>
      </choice>
    </sequence>
  </macro>
</command>
```

❶ Escape single quote characters "'" by replacing them by "\'". This is done using XPath function replace() in *XMLmind XML Editor - Support of XPath 1.0*.

❷ The value of attribute `style` may contain whitespace. Therefore it must quoted using single or double quotes. In the above example, it is quoted using single quotes.

32. 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).

33. editObject

Same as viewObject [163], except that the helper application is assumed to be an editor instead of a viewer. If this editor is used to modify the object, then the changes are also automatically applied to the document being edited.

Example: let's suppose the element of interest contains an image encoded using base 64 (data type `base64Binary`).

1. This command examines the first bytes of the image and, using this signature, determines which helper application to use.
2. If the helper application cannot be determined (because it has not yet been registered using the **Preferences** dialog box, **Helper Applications** section), the user is prompted to specify it.
3. It reads the image data from the element, decodes it and saves it to a temporary file.
4. It starts the image editor passing it the file containing the extracted image.
5. After the user quits the image editor, the command detects whether the extracted image has been modified and, if this is the case, reloads it in the element.

34. 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
```

35. editMediaInfo

Parameter syntax:

```
[ media_url ]?
```

Displays a dialog box letting the user configure selected media element (e.g. HTML5 `audio` or `video`) when this element is represented by a media player embedded in the styled view.

This command cannot work in the tree view. This command cannot work if feature "**In the styled view, render audio and video elements by embedding a media player**" in *XMLmind XML Editor - Online Help* is not available (no JavaFX) or has been turned off.

Optional parameter is the absolute or relative URL of an audio or video file. A relative URL is relative to the base URL of selected media element.

Examples:

```
editMediaInfo
editMediaInfo media/song.mp3
editMediaInfo http://techslides.com/demos/sample-videos/small.mp4
```

36. ensureSelectionAt

Parameter syntax:

```
[ 'selectElement' ]?
```

This command is intended to be bound to a mouse input. It is generally used by the macro-command which is bound to the `drag appEvent` in *XMLmind XML Editor - Configuration and Deployment*.

If the mouse is clicked anywhere inside the node or text selection, this command does nothing at all. That is, the selection is preserved (typically in order to be dragged). Otherwise this command selects the node clicked upon.

If the `selectElement` option is specified and the node clicked upon is not an element (e.g. a text node), then, it is its parent element which is selected.

XHTML example:

```
<binding>
  <appEvent name="drag" />
  <command name="xhtml.drag" />
</binding>

<command name="xhtml.drag">
  <macro>
    <sequence>
      <!-- Either drag the selection or
           select+drag the element clicked upon. -->
      <command name="ensureSelectionAt" parameter="selectElement" />

      <choice>
        <sequence>
          <match context="$selectedElement"
                 pattern="html:a[@href]|html:a[@href]//*" />
          <set variable="selectedElement" context="$selectedElement"
               expression="(ancestor-or-self::html:a[@href])[last()]" />
          <get context="$selectedElement" expression="resolve-uri(@href)" />
        </sequence>

        <sequence>
          <match context="$selectedElement" pattern="html:img[@src]" />
          <get context="$selectedElement" expression="resolve-uri(@src)" />
        </sequence>

        <!-- Default drag action. -->
        <command name="drag" />
      </choice>
    </sequence>
  </macro>
</command>
```

See also drag [102].

37. 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.

38. executeMenuItem

Parameter syntax:

```
[ '[' menu_item_index ']' ]?
  command_name
  | '"' command_name '"' [ command_parameter ]*
  | ''' command_name ''' [ command_parameter ]*
```

Executes command called *command_name* with optional parameter *command_parameter*. This command, typically found in the "*current_configuration_name* contextualMenuItems" namespace [21], is expected to return a menu [20].

The default value of *menu_item_index* is 0, the index of the first menu item. Menu item #*menu_item_index* specifies a command which is then executed. executeMenuItem returns the result of this command, if any.

Example 6.12. Implementing a "click to follow link" facility using executeMenuItem

The executeMenuItem command may be used to implement a "click to follow link" facility which works whatever the kind of link.

DocBook 5+ example: the implementation leverages the menu items added to the contextual menu when a link of any kind is implicitly or explicitly selected.

```
<command name="{$c contextualMenuItems}linkMenuItems">
  <macro>
    <choice>
      <!-- link, xref -->
      <sequence>
        <match context="$implicitElement"
               pattern="*[@linkend]|*[@linkend]//*" />
        <command name="{db5}linkMenuItems1" parameter="linkend" />
      </sequence>
      ...
      <!-- olink -->
      <sequence>
        <match context="$implicitElement" pattern="db:olink|db:olink//*" />
        <command name="{db5}linkMenuItems3" parameter="targetdoc" />
      </sequence>
    </choice>
  </macro>
</command>

<command name="{db5}linkMenuItems1">
  <menu>
    <item label="Follow Link"
          command="{db5}followLink" parameter="%0" />
    <item label="Set Link Target..."
          command="{db5}setLinkEnd" parameter="%0 %0" />
  </menu>
</command>
...
<command name="{db5}linkMenuItems3">
  <menu>
    <item label="Follow Link"
          command="{db5}followLink" parameter="%0" />
```

```
    <item label="Set Link Target..." command="db5.setOlinkTarget" />
  </menu>
</command>
```

The “click to follow link” facility may then be implemented as follows:

```
<command name="db5.followLinkAt">
  <macro>
    <sequence>
      <command name="selectAt" parameter="begin" />
      <command name="selectAt" parameter="end" />

      <command name="executeMenuItem"
               parameter="{$c contextualMenuItems}linkMenuItems" />
    </sequence>
  </macro>
</command>

<binding>
  <mouseClicked button="1" modifiers="mod alt" />
  <command name="db5.followLinkAt" />
</binding>
```

When the user clicks an internal link, the command which actually gets the job done is then "`{db5}followLink`" with parameter "`linkend`". When the user clicks an `olink`, the command which actually gets the job done is then "`{db5}followLink`" with parameter "`targetdoc`".

39. extendSelectionAt

No parameter.

Extends node selection if any, text selection otherwise.

See also selectAt [144].

40. extractObject

Parameter:

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

This command is the opposite of setObject [156]. It can be used to save to disk the object (generally an image) represented by explicitly selected element.

attribute_name

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

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

anyURI, hexBinary, base64Binary, XML

Specifies how the object 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 or RELAX NG 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 object. 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 object file is to be created.

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

Examples:

```
extractObject
extractObject fileref anyURI
extractObject -
extractObject data - /tmp/extracted.%X
extractObject - XML %T
```

41. fail

No parameter.

This command can never be executed.

This command is useful to write macro-commands. See also pass [126].

42. formatTextAs

Parameter syntax:

```
[ filter S ]* '#template(' element_name ',' template_title ')'
```

This command formats the plain text copied to the clipboard after its element template parameter. For example, if the clipboard contains:

```
This is the first line.
This is the second line.
This is the third line.
```

and the element template parameter points to (XHTML example):

```
<elementTemplate name="PAA.p" selectable="false">
  <p xmlns="http://www.w3.org/1999/xhtml">{$line}</p>
</elementTemplate>
```

This command will generate and return as a result the following string:

```
<?xml version="1.0"?>
<ns:clipboard xmlns:ns2="http://www.w3.org/1999/xhtml"
              xmlns:ns="http://www.xmlmind.com/xmleditor/namespace/clipboard">
  <ns2:p>This is the first line.</ns2:p>
  <ns2:p>This is the second line.</ns2:p>
  <ns2:p>This is the third line.</ns2:p>
</ns:clipboard>
```

This command is useful to create macro-commands [25]. It is also used internally by command pasteAs [128], which may be seen as a ready-to-use macro-command.

Example:

```
formatTextAs #template({http://www.w3.org/1999/xhtml}p,PAA.p)
```

Note how the name of the element is specified using James Clark's notation [85].

See also command `pasteAs` [128].

42.1. Specifying an element template for use by command `formatTextAs`

For example, let's suppose the clipboard contains:

```
This is line #1.
This is line #2.

This is line #3.
This is line #4.
```

The → symbol represents *a tab character*.

1. It is strongly recommended to add a selectable="false" in *XMLmind XML Editor - Configuration and Deployment* attribute to the element template. Doing this allows not to pollute the list of elements displayed the **Edit** tool with what is just a special purpose, private to the `formatTextAs` command, element template.

2. The element template must contain one or more text nodes referencing one of the following variables:

Variable	Description	Example
`{$line}`	The clipboard contains text lines separated by one or more newline characters. Open lines are ignored. This variable represents one such line.	The variable will be substituted 4 times: a. This is→line→#1. b. This is→line→#2. c. This is→line→#3.

Variable	Description	Example
		d. This is→line→#4.
`{$lineGroup}`	The clipboard contains groups of consecutive text lines separated by one or more open lines. This variable represents one such group of lines.	The variable will be substituted 2 times: a. This is→line→#1. This is→line→#2. b. This is→line→#3. This is→line→#4.
`{$lines}`	This variable represents the textual contents of the clipboard, as is. That is, whitespace is preserved.	The variable will be substituted 1 time: a. This is→line→#1. This is→line→#2. This is→line→#3. This is→line→#4.
`{$field}`	The clipboard contains text lines separated by one or more newline characters. Open lines are ignored. Each line contains one or more fields separated by tab characters. A field may be empty. That is, two consecutive tab characters may be used to specify an empty field. This variable represents one such field.	The variable will be substituted 12 times: • This is • line • #1. • This is • ... • line • #4.

You cannot mix different variables. For example, the same template cannot reference both `{$line}` and `{$field}`.

3. The element which is the parent of the text node referencing the variable is replicated as many times the variable needs to be substituted.

   All the elements containing variables are consumed in turn. If there are too many elements containing variables, the extra elements are discarded.

   For example (XHTML example), if the element template is:

```
<elementTemplate name="PAA.ul" selectable="false">
  <ul xmlns="http://www.w3.org/1999/xhtml">
    <li>A) {$line}</li>
    <li>B) {$line}</li>
    <li>C) {$line}</li>
    <li>D) {$line}</li>
    <li>E) {$line}</li>
    <li>F) {$line}</li>
```

```
  </ul>
</elementTemplate>
```

the above clipboard contents may be used to generate:

```
<?xml version="1.0"?>
<ns:ul xmlns:ns="http://www.w3.org/1999/xhtml">
  <ns:li>A) This is line #1.</ns:li>
  <ns:li>B) This is line #2.</ns:li>
  <ns:li>C) This is line #3.</ns:li>
  <ns:li>D) This is line #4.</ns:li>
</ns:ul>
```

4. When the referenced variable is `{$field}`,
   - the element which is the parent of the text node referencing the variable is replicated as many times as there are tab-separated fields in a given text line,
   - AND the element which the *grand-parent* of the text node referencing the variable is replicated as many times as there are text lines in the clipboard.

   For example (XHTML example), if the element template is:

```
<elementTemplate name="PAA.table" selectable="false">
  <table xmlns="http://www.w3.org/1999/xhtml" border="1">
    <tr>
      <td>{$field}</td>
    </tr>
  </table>
</elementTemplate>
```

   the above clipboard contents may be used to generate:

```
<?xml version="1.0"?>
<ns:table border="1" xmlns:ns="http://www.w3.org/1999/xhtml">
  <ns:tr>
    <ns:td>This is</ns:td>
    <ns:td>line</ns:td>
    <ns:td>#1.</ns:td>
  </ns:tr>
  <ns:tr>
    <ns:td>This is</ns:td>
    <ns:td>line</ns:td>
    <ns:td>#2.</ns:td>
  </ns:tr>
  <ns:tr>
    <ns:td>This is</ns:td>
    <ns:td>line</ns:td>
    <ns:td>#3.</ns:td>
  </ns:tr>
  <ns:tr>
    <ns:td>This is</ns:td>
    <ns:td>line</ns:td>
```

```
    <ns:td>#4.</ns:td>
  </ns:tr>
</ns:table>
```

5. In some cases you need to replicate an *ancestor* of the text node referencing the variable rather than its direct parent. In such case, explicitly add attribute `cfg:replicate="true"` to all the elements that are to be replicated.

   For example (DocBook example), if the element template is:

```
<elementTemplate name="PAA.itemizedlist" selectable="false"
  xmlns:cfg="http://www.xmlmind.com/xmleditor/schema/configuration">
  <itemizedlist xmlns="">
    <listitem cfg:replicate="true"><para>{$line}</para></listitem>
  </itemizedlist>
</elementTemplate>
```

   the above clipboard contents may be used to generate:

```
<?xml version="1.0"?>
<itemizedlist>
  <listitem>
    <para>This is line #1.</para>
  </listitem>
  <listitem>
    <para>This is line #2.</para>
  </listitem>
  <listitem>
    <para>This is line #3.</para>
  </listitem>
  <listitem>
    <para>This is line #4.</para>
  </listitem>
</itemizedlist>
```

42.2. Filtering the text pasted in the document

The element template may be preceded by one or more filters separated by whitespace. These filters are used to replace some characters (or to discard some characters) in the values of variables `{$line}`, `{$lineGroup}`, `{$lines}` or `{$field}`. This facility is used for example to discard the leading bullet from a list item.

The syntax of a filter is:

```
separator regex_pattern separator replacement separator g?i?m?s?
```

Example having an empty replacement (means: discard matched characters): "`/^\d\.//`".

Example using the `g` and `i` flags: "`^XXE^XMLmind XML Editor^gi`".

The same separator character must occur three times within a filter. This character may be any character, though it's customary to use "/".

The syntax supported for the regular expression pattern is documented in http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html.

Moreover, *extension character class* `\p{listItemBullet}`, which is equivalent to:

```
\u2022|\u2023|\u00B7|o|\*|#|-|[\u2012-\u2015]
|\p{InBoxDrawing}|\p{InDingbats}|\p{InPrivateUseArea}
|(\p{Alnum}(\.|\)))
|(\(\p{Alnum}\))
|([ivxlcdmIVXLCDM]+\.)
```

may be used to match leading bullets and numeric labels in list items. Example: "`/^\p{listItemBullet}\s//`".

The replacement text may be empty or it may contain `$0`, `$1`, ..., `$9` variables.

The final separator character may be immediately followed by one or more “flags”:

Flag	Description
g	Replace all occurrences of the matched text. By default, only the first occurrence is replaced.
i	Enable case-insensitive matching. By default, matching is case-sensitive.
m	Enable multiline mode. In multiline mode, expressions "^" and "$" match just after or just before, respectively, an end of line character or the end of the input sequence. By default, these expressions only match at the beginning and the end of the entire input sequence.
s	Enable dotall mode. In dotall mode, expression "." matches any character, including an end of line character. By default, this expression does not match end of line characters.

43. include

Parameter syntax:

```
'into' | ('replace'|'before'|'after' [ implicit_selection ]?)
[ '[multipleInstances]' ]?
[ referenced_document_URL reference_id [  '[absoluteReference]' ]? ]?
```

into
: Pastes a reference into element containing caret, at caret position.

replace
: Pastes a reference replacing text selection or selected nodes [84].

before or after
: Pastes a reference before of after selected nodes [84].

Commands copyAsInclusion [93] (generally bound to keystroke **Shift**+**Ctrl**+**C**) and paste [126] (generally bound to keystroke **Ctrl**+**V**) are used to compose *modular documents*.

When parameters *`referenced_document_URL`* and *`reference_id`* are not specified, command `include` is basically an alternative user interface for composing modular documents. It displays a dialog box, similar to the **Include** tool, allowing the user to choose which reference to insert.

When parameters *`referenced_document_URL`* and *`reference_id`* are specified, command `include` may be used in macro-commands to automate tasks.

referenced_document_URL

The URL of the document containing the nodes to be referenced. May be relative or absolute. If it is a relative URL, it is relative to the URL of the including document.

Note that the fact *`referenced_document_URL`* is absolute or relative is orthogonal to the `[absoluteReference]` option.

URLs which need an XML catalog in order to be resolved are also supported here (see last example below). In such case, the `[absoluteReference]` option is ignored and the including document references the included document using as is the specified URL.

reference_id

A string identifying the nodes to be referenced:

- DITA-style ID for DITA documents (e.g. `my_topic`, `my_topic/my_paragraph`),
- standard ID or "-" for other document types.

  If you want to include the root element of a document, you must refer to it by its ID if it has one (see the section1.xml example below) or as "-" otherwise (see the section2.xml example below).

[absoluteReference]

When this option is specified, the including document references the included document using an absolute URL. By default, a URL relative to the including document is used.

Note that the effect of this option does not depend on whether *`referenced_document_URL`* is itself absolute or not.

The `[multipleInstances]` option is a hint which indicates that the reference created by the command may be found several times, at different places, in the including document. Example: when you include a chapter in a book, there is no need to specify `[multipleInstances],` but when you include boilerplate text like a product name or a company name, then it is recommended to specify `[multipleInstances]`. This is why the **Include** tool in *XMLmind XML Editor - Online Help* always uses option `[multipleInstances]` to create the references it pastes in a document.

Examples:

```
include into
include after[implicitElement]
include after[implicitElement][multipleInstances]
include replace[implicitNode]
include into file:/home/john/doc/boilerplate.xml product_name
include before[implicitElement] ../common/Copyright.xhtml copyright [absoluteReference]
include into http://www.acme.com/docs/licence.xml disclaimer [absoluteReference]
```

```
include into [multipleInstances] ../common/licence.xml disclaimer

include after section1.xml s1
include after section2.xml -

include into boilerplate:common/trademarks.xml super_foo
```

44. insert

Parameter syntax:

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

If the `into` option is specified, this command inserts its argument node [84]:

- into the explicitly selected element, after its last child node
- OR, if no element is explicitly selected, into the element containing the caret, at caret position.

If the `before` option is specified, this command inserts its argument node [84] before the selected nodes [84].

If the `after` option is specified, this command inserts its argument node [84] after the selected nodes [84].

Examples:

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

45. insertCharByName

Parameter syntax:

```
[ '[DocBook]'|'[$property_name]'|'[DocBookIfNone]' ]? [ S char_spec ]?
```

Inserts at caret position a character specified by *`char_spec`*. If *`char_spec`* is not specified, this command displays a dialog box (supporting auto-completion) which lets the user specify it interactively.

This command honors the *Overwrite Mode* (**OVR**) in *XMLmind XML Editor - Online Help* when this mode has been turned on.

This command does not insert a *reference* to a character entity, it inserts a *character*. It must be considered as an alternative to using the **Characters** tool of XMLmind XML Editor.

Parameter *`char_spec`* is normally the name of the corresponding character entity. However character representations other than entity names are also supported. For example, character `TAB` may now be specified as any of the following: `"\t"`, `"\11"`, `"\x9"`, `"\u0009"`, `"U+0009"` ,`"&#9;"`, `"&#x9;"`, `"011"`, `"0x9"`.

The character entities listed in the dialog box displayed by this command are determined as follows:

1. If a `[DocBook]` parameter has been specified, use the character entities defined in the DocBook 4.4+ DTD[1] and this, whatever the schema the document being edited is conforming to.

2. If a `[$property_name]` parameter has been specified, use the character entities defined in the Java™ properties file which is the value of Java™ property *`property_name`*.

3. Use the character entities defined in the DTD to which the document being edited is conforming to.

4. Use the character entities defined in the Java™ properties file which is the value of Java™ property *`configuration_name`*`.characterEntities`, where *`configuration_name`* is the name of the configuration associated to the document being edited.

5. If a `[DocBookIfNone]` parameter has been specified, use the character entities defined in the DocBook 4.4+ DTD.

The above steps are tried in order until a step succeeds. If all steps fail, this command cannot be executed and therefore, displays no dialog box at all.

Examples:

```
insertCharByName
insertCharByName beta
insertCharByName U+03B2
insertCharByName [DocBook]
insertCharByName [DocBook] lambda
insertCharByName [$my_favorite_chars]
insertCharByName [DocBookIfNone]
```

Example 6.13. MathML example

Let's suppose that the MathML configuration is based on `mathml2.xsd` and not on `mathml2.dtd`. Even without a DTD, you want to be able to insert math characters specified using their entity names (`CircleDot`, `sum`, `it`, etc). Here's how to do that:

a. Create a Java™ properties file defining all the character entities you need (`mathml_chars.properties`):

```
...
CircleDot=\u2299
CircleMinus=\u2296
CirclePlus=\u2295
CircleTimes=\u2297
...
```

b. Add this property configuration element to the `mathml.xxe` configuration file:

```
<property name="MathML.characterEntities"
          url="true">mathml_chars.properties</property>
```

[1] That is, the character entities defined in XML Entity Declarations for Characters.

c. When editing a MathML document, use command `insertCharByName` without any special option. Examples:

```
insertCharByName
insertCharByName InvisibleTimes
insertCharByName af
```

46. insertCharSequence

Parameter syntax:

```
first_character S second_character [ S third_character ]?
```

Makes it easy and intuitive inserting special characters by typing the same ordinary character two or three times in a row.

The first time *`first_character`* is typed, as expected, *`first_character`* is inserted at caret position. Example: the first time, you type '-' (an ordinary dash), you insert '-'.

The second time *`first_character`* is typed, previously inserted *`first_character`* is replaced by *`second_character`*. Example: the second time you type '-' , you insert a `&ndash;` special character.

The third time *`first_character`* is typed, previously inserted *`second_character`* is replaced by *`third_character`*. Example: the third time you type '-' , you insert a `&mdash;` special character. This, of course, requires *`third_character`* to have been specified, which is not mandatory.

This command is useless unless bound to the action of typing *`first_character`*.

This command honors the *Overwrite Mode* (**OVR**) in *XMLmind XML Editor - Online Help* when this mode has been turned on.

Characters may be specified

- literally (e.g. "<"),
- OR using their character entity name as defined in the DocBook DTD (whatever the schema to which the document is conforming),
- OR using some common character representations. For example, character `TAB` may now be specified as any of the following: "`\t`", "`\11`", "`\x9`", "`\u0009`", "`U+0009`" ,"`&#9;`", "`&#x9;`", "`011`", "`0x9`".

Examples (in the examples below, hexadecimal number `0x00ab` is used to represent the French opening *guillemet* "«" and octal number `0273` is used to represent the French closing *guillemet* "»"):

```
<binding>
  <charTyped char="-" />
  <command name="insertCharSequence" parameter="- ndash mdash" />
</binding>

<binding>
  <charTyped char="&lt;" />
```

```
  <command name="insertCharSequence" parameter="&lt; 0x00ab" />
</binding>

<binding>
  <charTyped char="&gt;" />
  <command name="insertCharSequence" parameter="&gt; 0273" />
</binding>
```

47. 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 honors the *Overwrite Mode* (**OVR**) in *XMLmind XML Editor - Online Help* when this mode has been turned on.

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` [126] command always work.

48. insertNewlineOrSplitBlock

No parameter.

If the caret is contained in a element which accepts newline characters (e.g. `pre`, `programlisting`), a newline character (`&0#xA;`) is inserted in this element. Otherwise, if the caret is contained in a *block*, this block is split in two parts.

To make it simple, a block is a paragraph, a heading (e.g. XHTML `h1`, `h2`, ..., `h6` or DocBook `bridgehead`) or a list item. However what is exactly a block must be specified as explained in Section 52.1, “Specifying splittable blocks” [122].

More precisely,

- If the caret is found at the very beginning of a block, a new, similar, empty block is inserted before the current one.
- If the caret is found at the very end of a block, a new, similar, empty block is inserted after the current one.
- Otherwise, the current block is split in two parts.

However, if a block is declared as being a *heading* and if another block has been declared as being the most common form of paragraphs (see Section 52.1, “Specifying splittable blocks” [122]):

- If the caret is found at the very beginning of a heading, a new, empty paragraph is inserted before this heading.
- If the caret is found at the very end of a heading, a new, empty paragraph is inserted after this heading.
- Otherwise, the current heading is split in two parts.

When acting on the current block, this command can be executed if there is no selection or if a single element is explicitly selected. When there is no selection, the current block is automatically determined among the elements which directly or indirectly contain the caret. When an element is selected, this element is considered to be the current block if and only if it directly or indirectly contains the caret and it is listed in the *`configuration_name`* `blockList` property.

This command is intended to be bound to the Enter key.

49. 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 [84].

`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 the last one interactively specified using command editPITarget [104] if any, and placeholder string "`target`" otherwise.

`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
```

50. insertOrOverwriteString

Parameter syntax:

```
string
```

Inserts or overwrites, depending on overwrite mode, specified string at caret position.

See insertString [123], overwriteString [126], overwriteMode [125].

51. insertSpecialChars

Parameter syntax:

```
[ char_spec ]?
```

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 parameter *`char_spec`* is specified, 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.

Several character representations are accepted for *`char_spec`*. For example, character `TAB` may now be specified as any of the following: "`\t`", "`\11`", "`\x9`", "`\u0009`", "`U+0009`" ,"`&#9;`", "`&#x9;`", "`011`", "`0x9`".

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**.

52. insertSameBlock

Parameter syntax:

```
[ 'before' ]?
```

Insert a new *block*, similar to the current one, after the current one. If option `before` is specified, the new block is inserted before the current one.

To make it simple, a block is a paragraph or a list item. However what exactly is a block must be specified as explained in Section 52.1, “Specifying splittable blocks” [122].

This command can be executed if there is no selection or if a single element is explicitly selected. When there is no selection, the current block is automatically determined among the elements which directly or indirectly contain the caret. When an element is selected, this element is considered to be the current block if and only if it directly or indirectly contains the caret and it is listed in the *`configuration_name`* `blockList` property.

This command is intended to be bound to keys **Ctrl**+Enter and **Ctrl**+**Shift**+Enter.

52.1. Specifying splittable blocks

A `property` in *XMLmind XML Editor - Configuration and Deployment* configuration element having a "*`configuration_name`* `blockList`" `name` attribute must be specified in order to make the following commands work properly: `deleteSelectionOrJoinBlockOrDeleteChar` [100], `insertNewlineOrSplitBlock` [119], `insertSameBlock` [121].

The property simply contains the names of the elements which are to be considered as being splittable blocks by the aforementioned commands.

- The names must be separated by whitespace (of any kind).
- The order of the names in the list is not significant.
- A qualified name must be expressed using the Clark's notation {*`namespace_URI`*}*`local_name`*, that is, namespace prefixes are not supported here.
- A name may be preceded by the name of its parent element. Example: `listitem` and `ordered-list`/`listitem` are both supported.
- Annotation "`=paragraph`" may immediately follow the name of a block. This annotation marks the block as being the most common form of paragraph.
- Annotation "`=heading`" may immediately follow the name of a block. This annotation marks the block as being a heading. See `insertNewlineOrSplitBlock` [119] to learn how this annotation is used.

DocBook 4 example:

```
<property name="DocBook❶ blockList">
  bridgehead=heading
  simpara para=paragraph
  term
  listitem❷
  varlistentry
  callout
  step
</property>
```

❶ Here the name of the configuration for which a block list is being specified is "`DocBook`".

❷ This list generally contains the names of paragraphs and list items of all kinds. However, it's also possible to add headings of all kinds.

XHTML example:

```
<property name="$c❶ blockList">
  {http://www.w3.org/1999/xhtml}p=paragraph
  {http://www.w3.org/1999/xhtml}h1=heading
  {http://www.w3.org/1999/xhtml}h2=heading
  {http://www.w3.org/1999/xhtml}h3=heading
  {http://www.w3.org/1999/xhtml}h4=heading
  {http://www.w3.org/1999/xhtml}h5=heading
  {http://www.w3.org/1999/xhtml}h6=heading
  {http://www.w3.org/1999/xhtml}li
  {http://www.w3.org/1999/xhtml}dt
</property>
```

❶ Pseudo-variable `$c` is automatically substituted with the name of the configuration being loaded by **XXE**.

53. insertString

Parameter syntax:

```
string
```

Inserts specified string at caret position.

54. insertTextOrMoveDot

Parameter syntax:

```
[ 'after' | 'before' ]?
```

When parameter `after` is specified or when no parameter is specified, this command is similar to "`insertNode textAfter[implicitElement]`".

When parameter `before` is specified, this command is similar to "`insertNode textBefore[implicitElement]`".

The only difference with command `insertNode` [120] is that when a new text cannot be inserted because there is already a text node after or before selected element, the `insertTextOrMoveDot` moves the caret to the existing text node.

55. 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 [159].

Examples:

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

56. listAnchors

No parameter.

Displays the "**List Anchors and Links**" dialog box which lets the user search and select anchors (that is, any element having an ID) and links.

57. listBindings

No parameter.

Displays a dialog box containing the mouse and key bindings that can be used in current document view.

This command is mainly useful to XML consultants and Java™ developers customizing or extending **XXE**.

58. listPlugins

No parameter.

Displays a dialog box containing information about all plug-ins currently loaded into **XXE**.

This command is mainly useful to XML consultants and Java™ developers customizing or extending **XXE**.

59. listRepeatable

Parameter syntax:

```
[ index_in_command_history ]?
```

Without a parameter, this command displays a dialog box letting the user select and execute one of the last executed repeatable commands.

The *`index_in_command_history`* specifies which command found in the command history is to be executed. Note that index 0 specifies the most recently executed command in the command history. When this index is specified, the selector dialog box is not displayed and specified command is automatically executed (when this is allowed given current editing context).

Just like `repeat` [136], this command returns the result of repeated command (if any).

60. 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.

61. moveElement

Parameter syntax:

```
'up'|'down' [ '[implicitElement]' ]?
```

Swaps selected element with its preceding sibling node (`up` option) or with its following sibling node (`down` option).

Examples:

```
moveElement down[implicitElement]
moveElement up
```

62. normalizeWhiteSpacePre

Parameter syntax:

```
[ tab_width ]?
```

Normalize whitespace in implicitly or explicitly selected element having attribute `space="preserve"`, typically a program listing.

Normalizing whitespace means:

- replacing tab characters by a number of space characters;
- removing the space characters which are common to the beginning of all text lines, that is, removing the superfluous “indentation” in the program listing;
- removing the (useless) space characters found just before newline characters.

Optional parameter *`tab_width`* represents the maximum number of space characters for an expanded tab character. If this parameter is not specified, a dialog box is displayed to let the user specify *`tab_width`*, an integer between 1 and 16, typically 4 or 8.

Examples:

```
normalizeWhiteSpacePre
normalizeWhiteSpacePre 4
```

63. overwriteMode

Parameter syntax:

```
'toggle'|'on'|'off'|isOn'
```

Allows to switch from *Insert Mode* to *Overwrite Mode* and vice versa.

Insert Mode

Typing a character inserts it at caret position.

Overwrite Mode

Typing a character *replaces* the character found at caret position by the typed character. If the caret is positioned at the very end of a text (or comment or processing-instruction) node, then typed characters are simply inserted there.

Options:

toggle

Switch from Insert Mode to Overwrite Mode and vice versa.

on

Ensures that Overwrite Mode is turned on.

off

Ensures that Overwrite Mode is turned off.

isOn

Returns `Boolean.TRUE` if Overwrite Mode is turned on; otherwise returns `Boolean.FALSE`.

Whatever the option used, this command returns a `Boolean` indicating whether Overwrite Mode is turned on.

64. overwriteString

Parameter syntax:

```
string
```

Replaces characters found at caret position by specified string.

Example: a text node contains "`Hello world!`" and the caret is before the "`w`" of "`world`". "`overwriteString 'beautiful world!'`", replaces "`world!`" by "`beauti`" and then inserts "`ful world!`" at the end of the text node.

65. pass

Parameter syntax:

```
[ string ]?
```

This command can always be executed, but does nothing at all other than returning its parameter as its result.

This command is useful to write macro-commands. See also fail [109].

66. paste

Parameter syntax:

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

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

`to`
: Pastes the content of system clipboard replacing text selection or selected nodes [84].

  Options `to`, `toOrInto`, `toOrAdd` allow to replace the explicitly selected root element of a document by another element having the same name.

`toOrInto`
: Pastes the content of system clipboard replacing text selection or selected nodes [84].

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

`before`, `after`
: Pastes the content of system clipboard before of after selected nodes [84].

`add`
: Pastes the content of system clipboard at any valid position in the document following the caret position.

`toOrAdd`
: Pastes the content of system clipboard replacing text selection or selected nodes [84].

  OR if there is no explicit selection, pastes the content of system clipboard at any valid position in the document following the caret position.

The system clipboard may contain XML or plain text.

If the `[systemSelection]` option is used, the content of system selection (always plain text) is used instead of the content of system clipboard.

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

Examples:

```
paste toOrInto
paste toOrInto[systemSelection]
paste before[implicitElement]
paste before[implicitElement][systemSelection]

paste after <?xml version='1.0'?><p>A paragraph.</p>
paste toOrAdd <?xml version='1.0'?><p>A paragraph.</p>

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>
```

In the above example, for a better readability, the XML strings to be pasted are represented as is —unescaped— and sometimes indented. *In practice, this would prevent the `paste` command from working!*

An XML string to be pasted:

- must have all its markup delimiters ("<", "&", etc) properly escaped ("`<`", "`&`", etc);
- must not contain any non-significant whitespace (i.e. indentation).

67. pasteAs

Parameter syntax:

```
'into'|'toOrInto'|'add'|'toOrAdd' | ('to'|'before'|'after' [ implicit_selection ]?)
S '#template(' element_name ',' template_title ')'
```

This command is basically equivalent to the following macro-command:

```
<command name="pasteAs">
  <macro>
    <sequence>
      <command name="formatTextAs" parameter="%1" />
      <command name="paste" parameter="%0 %_" />
    </sequence>
  </macro>
</command>
```

See commands `formatTextAs` [109] and `paste` [126].

68. pasteImageAs

Parameter syntax:

```
'into'|'toOrInto'|'add'|'toOrAdd' | ('to'|'before'|'after' [ implicit_selection ]?)
S '#template(' element_name ',' template_title ')'
```

Variant of command `paste` [126] which pastes the *image* copied to the system clipboard as an element.

The element to be pasted is specified using an element template [84]. This element template must have a descendant element or attribute containing a string starting with "`image-URI:`". After the "`image-URI:`" prefix, an absolute or relativeURI specifies *a template for the path of the save files* used to store the pasted images.

XHTML example:

```
pasteImageAs toOrInto #template({http://www.w3.org/1999/xhtml}img,PIA.img)
```

where element template `PIA.img` is:

```
<elementTemplate name="PIA.img" selectable="false">
  <img xmlns="http://www.w3.org/1999/xhtml"
       src="image-URI:images/img.jpg" alt="???"/>
</elementTemplate>
```

Using the above command creates files called `images/img14135.jpg`[2], `images/img20338.jpg`, `images/img60659.jpg`, etc, where `images/` is a subfolder of the folder containing the document being edited.

DITA example:

```
pasteImageAs toOrAdd #template(fig,PIA.fig)
```

where element template `PIA.fig` is:

```
<elementTemplate name="PIA.fig" selectable="false">
  <fig xmlns="">
    <title></title>
    <image href="image-URI:image.png" />
  </fig>
</elementTemplate>
```

Using the above command creates files called `image29116.png`, `image89803.png`, `image09833.png`, etc, into the folder containing the document being edited.

69. pasteSystemSelection

No parameter.

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

70. 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. This command returns the string selected by the user.

This dialog box supports autocompletion. This implies that the items of the pick list are automatically sorted by their labels.

[2]The user is of course prompted to choose a more meaningful filename.

If second field in the command parameter is `false`, the list of strings displayed by the dialog box is [*`item`*]+. That is, *`item`* is both a possible choice and a label for this possible choice.

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
```

71. preview

Parameter syntax:

```
'[lastConverted]' | URL_or_filename
```

This command is only useful to write interactive macro commands.

Starts a helper application allowing to preview specified URL or file. If an appropriate helper application has not yet been specified using **Options** → **Preferences**, **Helper Applications**, the user is automatically prompted to specify this application.

The file or directory to be previewed may be specified as:

`[lastConverted]`
: This specifies the last file or directory selected by the user by the means of the `selectConvertedFile` [145] command. This command has a "**Preview result in helper application**" checkbox. This checkbox must have been checked by the user.

  Commands `preview` and `selectConvertedFile` are used together in the same macro typically as follows:

```
<command name="docb.convertToPS">
  <macro>
    <sequence>
      <command name="selectConvertedFile"
               parameter="saveFileURLWithExtension=%0"/>
      <command name="docb.toPSFile" parameter='"%0" "%1" "%_"'/>
      <command name="preview" parameter="[lastConverted]"/>
    </sequence>
```

```
  </macro>
</command>
```

URL_or_filename

Specifies the file or directory to be previewed by its URL or its filename. A relative URL is relative to the URL of the document being edited. A relative filename is relative to the current working directory.

Examples:

```
preview [lastConverted]
preview C:\tmp\test.docx
preview file:///C:/tmp/test.docx
preview images/logo.svg
```

See also `selectConvertedFile` [145].

72. promoteListItem

No parameter, but a property called "*configuration_name* listItemSpecification" [96] should be defined.

Convert a paragraph to a list item and a list item to a paragraph, the new element having a lesser nesting level than the original one. This command automatically splits lists when needed to.

This is the inverse command of `demoteListItem` [95]. More Information above [95].

73. 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
```

74. putAttribute

Parameter syntax:

```
[ '[implicitElement]' ]? [ '[empty]'|'[dummy]'|'[default]'|'[id]' ]? [ '[simplePrompt]'|'[norma
attribute_name [ attribute_value ]?
```

Adds or replaces attribute *`attribute_name`* in explicitly or implicitly selected element if the 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).

  Note that an attribute value containing whitespace must be quoted using single or double quotes. See `alt` example below.

  For obvious reasons, attribute value *`attribute_value`* must not be specified when any of option `[empty]`, `[dummy]` or `[id]` is used.

- 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 `[default]` has been specified, sets the attribute to its default value if any and to string "`???`" otherwise (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).

    This dialog box simply contains a text field if option `[simplePrompt]` has been specified.

    By default or if option `[normalDialog]` has been specified, this dialog box may contain a more advanced editor, which is specific to the type of the attribute being edited.

    Note that this “normal dialog box” automatically makes use of custom attribute editors defined by the means of the `attributeEditor` configuration element in *XMLmind XML Editor - Configuration and Deployment*.

Examples:

```
putAttribute cols
putAttribute alt "XMLmind logo"
putAttribute [implicitElement] cols
putAttribute [dummy] cols
putAttribute [implicitElement] [id] xml:id
putAttribute [simplePrompt] href
putAttribute [implicitElement] [simplePrompt] href
```

See also `editAttribute` [103] and `addAttribute` [86].

75. recordMacro

Parameter syntax:

```
'start'|'stop'|'toggle'|'cancel'|'view'|'get'|'replay'
```

This command allows to record a sequence of commands and to replay the recorded sequence at will.

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 last recorded macro in XML form. Very handy to paste it in an XXE configuration file (see XMLmind XML Editor - Configuration and Deployment).

get
: Returns a string containing last recorded macro in XML form. This option is useful to write higher-level commands and actions.

replay
: Replays recorded sequence of commands.

At most 100 commands can be recorded. Typing contiguous characters, no matter how many, counts as a single command (insertString [123]).

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 [144]),
- undo [163], redo [134], repeat [136],
- 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 [116] 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 [106] 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
```

76. redo

No parameter.

Redo last undone command.

77. refresh

Parameter syntax:

```
'refresh'|'rebuild' [ '[implicitNode]'|'[implicitElement]'|'[implicitDocument]' ]?
```

Refreshes or rebuilds selected node [84].

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]
```

78. reinclude

Parameter syntax:

```
[ '[all]' | '[implicitElement]' ]?
```

When option `[all]` is specified, this command transcludes all the inclusion directives (e.g. `xi:include` elements) found in the document being edited.

Otherwise, this command replaces the selected inclusion directive by up-to-date included nodes. For this case to work, the explicitly selected element or processing-instruction or the implicitly selected element (when option `[implicitElement]` has been specified) is expected to be an inclusion directive.

This command is the inverse of uninclude [163].

79. remark

```
[ 'edit' [ S PI_content ]? |
  'delete' | 'deleteAll' |
  'previous' | 'next' | 'first' | 'last' ]?
```

Command acting on *remarks*.

A remark is simply a processing-instruction having "`xxe-remark`" as its target. This processing-instruction is typically used to comment changes, for example when reviewing a document. This processing-instruction is nicely rendered using **XXE**'s built-in CSS stylesheet and also, the **Compare** tool displays such processing-instruction in a manner which stands out from the other parts of the changes it has detected.

edit
: If a `<?xxe-remark>` is explicitly selected or if the caret is contained in a `<?xxe-remark>`, display a special dialog box —the remark editor— allowing to edit this processing-instruction. Otherwise, display the remark editor in order to create a new remark. This new `<?xxe-remark>` is inserted before the explicit selection if any, or at caret position otherwise.

  If *`PI_content`* is specified and is successfully parsed as the content of a `<?xxe-remark>` processing-instruction, the remark editor dialog box is not displayed and the `<?xxe-remark>` processing-instruction is directly modified or inserted.

delete
: Delete explicitly selected `<?xxe-remark>`, if any. Otherwise if the caret is contained in a `<?xxe-remark>`, delete this processing-instruction.

deleteAll
: Delete all `<?xxe-remark>`s.

previous
: Select preceding `<?xxe-remark>` if any.

next
: Select following `<?xxe-remark>` if any.

first
: Select first `<?xxe-remark>` if any.

last
: Select last `<?xxe-remark>` if any.

Examples:

```
remark
remark edit
remark deleteAll
remark first
remark edit john
2024-05-01T13:28:18Z
TO DO!
```

80. 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
```

81. repeat

No parameter.

Repeats last repeatable command. Returns result of repeated command (if any).

82. replace

Parameter syntax:

```
[ implicit_selection ]? [ S argument_node ]?
```

Replaces selected nodes [84] with argument node [84].

Note that XMLmind XML Editor does not allow to replace the root element of a document. However it is possible to replace all the child nodes of the root element.

Examples:

```
replace [implicitElement]
replace {http://www.xmlmind.com/xmleditor/schema/configuration}newElementTemplate
```

83. resizeImage

Parameter syntax:

```
[ attribute_name['='[attribute_value]] ]+
```

This command allows to resize an image by removing one or more attributes and/or setting one or more attributes to values expressed in pixels.

attribute_name=attribute_value

Specifies that attribute *attribute_name* must be set to value *attribute_value* (which may be the empty string).

An attribute name is specified using the {*namespace_URI*}*local_name* notation. Examples: `alt`, `{}alt`, `{http://www.w3.org/1999/xlink}:href`, `xml:id`.

attribute_name

Notice there is no `'='` sign. Specifies that attribute *attribute_name* must be removed.

This command has been designed to be bound to the following application events in *XMLmind XML Editor - Configuration and Deployment*. These application events are generated by an image-viewport() in *XMLmind XML Editor - Support of Cascading Style Sheets (W3C CSS)* when the user drags one of the handles displayed around the image:

`rescale-image`

Resize the image, but always preserve its aspect ratio.

`resize-image`

This application event is generated when the user drags a handle while pressing Ctrl (Cmd on the Mac). This allows to distort the image.

Binding one of the above application events to a command allows to have one or more of the following variables substituted in the parameter of the bound command:

`%{width}`

The new width of the image expressed in pixels.

`%{height}`

The new height of the image expressed in pixels.

`%{preserveAspect}`

`true` if the aspect ratio has been preserved while the user dragged the resize handle; `false` otherwise.

XHTML example:

```
<binding>
  <appEvent name="resize-image" />
  <command name="resizeImage"
           parameter="height=%{height} width=%{width}" />
</binding>

<binding>
  <appEvent name="rescale-image" />
  <command name="resizeImage" parameter="height width=%{width}" />
</binding>
```

DocBook example:

```
<binding>
  <appEvent name="resize-image" />
  <command name="resizeImage"
           parameter="contentdepth=%{height} contentwidth=%{width}
                      scale scalefit" />
</binding>
```

```
<binding>
  <appEvent name="rescale-image" />
  <command name="resizeImage"
           parameter="contentdepth contentwidth=%{width}
                      scale scalefit"/>
</binding>
```

84. resizeTableTemplate

Parameter syntax:

```
cell_name[ '|' cell_name]* S command S parameter
```

This commands displays a dialog box allowing to quickly specify the number of rows and the number of columns of the new table to be inserted in the document.

More precisely:

- This command executes command *`command`* with parameter *`parameter`*.
- Command *`command`* is expected to insert a new table in the document.
- Parameter *`parameter`* is expected to contain a table template specification (syntax: `#template(`*`element_name`*`,`*`template_title`*`)`).
- If the table size specified using the dialog box differs from the size of the template table, then this command replaces in parameter *`parameter`* the original template specification by a custom one prior to executing command *`command`*.
- The effect of this custom template specification in parameter *`parameter`* is that the newly inserted table has the size interactively specified by the user.

DITA `simpletable` example:

```
resizeTableTemplate stentry dita.addBlock #template(simpletable,head)
```

XHTML example (notice "`td|th`"):

```
resizeTableTemplate¬
 {http://www.w3.org/1999/xhtml}td|{http://www.w3.org/1999/xhtml}th¬
 addBlockInFlow¬
 #template({http://www.w3.org/1999/xhtml}table,simplest)
```

See also resizeCALSTableTemplate [138].

85. resizeCALSTableTemplate

Same as resizeTableTemplate [138], except that this command also sets the `cols` attributes of the `tgroup` elements of the new table.

DITA example:

```
resizeCALSTableTemplate entry dita.addBlock #template(table,head)
```

DocBook 5 example:

```
resizeCALSTableTemplate¬
 {http://docbook.org/ns/docbook}entry¬
 db5.addAfter¬
 #template({http://docbook.org/ns/docbook}informaltable,head_row)
```

86. run

Parameter syntax:

```
[ [ '[Windows]'|'[Unix]'|'[GenericUnix]'|'[Mac]' ]? 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).

Specified command line may reference helper applications [72] declared using the **Preferences** dialog box, **Helper Applications** section.

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.

If the platform option (that is, `[Windows]`, `[Unix]`, `[GenericUnix]` or `[Mac]`) is not specified, the command line is executed whatever is the platform running XXE.

If the platform option is specified, the command line is executed only if the platform running XXE matches the value of this options:

[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].

Examples:

```
run date
run expand %F
run emacs "%D"
run helper(text/plain) "%D"
run "C:\Program Files\Info ZIP\zip.exe" -r all.zip "C:\temp\misc"

<choice>
  <command name="run" parameter='[Windows] notepad "%D"' />
  <command name="run" parameter='[Unix] emacs "%D"' />
</choice>
```

87. 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, this command displays a dialog box allowing to specify such text as well as 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.

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 "`<em>Hello</em>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)

Search backwards.

x (eXtend text selection)

Extend text selection, if any, in order to include found text.

Examples:

```
search
search [xwi]
search XMLmind XML Editor
search [r] X\w+d
search "XMLmind "
search ' XML'
```

See also searchReplace [141] and xpathSearch [167].

88. searchReplace

Parameter syntax:

```
[ search|searchBackwards|replace|replaceBackwards|
  findAgain|findAgainBackwards ]?
```

Displays a modal dialog box which lets the user search and replace text in the document being edited (same as the **Search** tool in XMLmind XML Editor).

The parameter defaults to `search`.

`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.

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**.

89. searchReplaceElement

Parameter syntax:

```
[ search|searchBackwards|replace|replaceBackwards|
  findAgain|findAgainBackwards|
  replaceAgain|replaceAgainBackwards|
  find search_spec|findBackwards search_spec]?
```

Displays a (non-modal) dialog box which lets the user search and replace elements in the document being edited.

The parameter defaults to `search`.

`search`
: Displays the dialog box ready to be used to search elements from explicitly or implicitly selected node to the end of the document.

`searchBackwards`

Displays the dialog box ready to be used to search elements from explicitly or implicitly selected node to the beginning of the document (backwards).

`replace`

Displays the dialog box ready to be used to search and replace elements from explicitly or implicitly selected node to the end of the document.

`replaceBackwards`

Displays the dialog box ready to be used to search and replace elements from explicitly or implicitly selected node to the beginning of the document (backwards).

`findAgain`

The dialog box is not displayed. Repeats last specified search operation starting at explicitly or implicitly selected node and ending at the end of the document.

`findAgainBackwards`

The dialog box is not displayed. Repeats last specified search operation starting at explicitly or implicitly selected node and ending at the beginning of the document (backwards).

`replaceAgain`

The dialog box is not displayed. Repeats last specified search and replace operation starting at explicitly or implicitly selected node and ending at the end of the document.

`replaceAgainBackwards`

The dialog box is not displayed. Repeat last specified search and replace operation starting at explicitly or implicitly selected node and ending at the beginning of the document (backwards).

`find`*`search_spec`*

The dialog box is not displayed. Performs specified search operation starting at explicitly or implicitly selected node and ending at the end of the document.

`findBackwards`*`search_spec`*

The dialog box is not displayed. Performs specified search operation starting at explicitly or implicitly selected node and ending at the beginning of the document (backwards).

`search_spec`, the argument of `find` and `findBackwards`, contains up to 7 fields separated with whitespace. An empty field is specified as character minus (`-`) without a quote of any kind. A field containing whitespace must be quoted (e.g. `"1 2 3"` or `'a b c'`). Character minus (`-`) when used as an actual field value must be quoted (that is, `"-"` or `'-'`).

These 7 fields are:

- Searched element name. This field may contain one or more qualified XML names (e.g. `para`, `{http://www.w3.org/1999/xhtml}li`) or wildcards (e.g. `*`, `{http://www.w3.org/2000/svg}*`).

  Note that namespace prefixes are not supported here. A qualified name must be specified using notation {*`namespace_URI`*}*`local_name`*, the so-called James Clark's notation.

- Searched attribute name. This field may contain one or more qualified XML names (e.g. `href`, `xml:lang`) or wildcards (e.g. `*`, `{http://www.w3.org/1999/xlink}*`).

Note that namespace prefixes are not supported here. A qualified name must be specified using notation {*namespace_URI*}*local_name*, the so-called James Clark's notation.

- Operation used to compare found attribute value with searched attribute value: = (Equals), *= (Contains), ~= (Contains item), =~ (Matches Regular Expression).
- Searched attribute value.
- #text is searched element must contain text, - otherwise.
- Operation used to compare found text with searched text: = (Equals), *= (Contains), =~ (Matches Regular Expression).
- Searched text.

Examples:

```
{http://www.w3.org/1999/xhtml}caption - - - - - -

- xml:id - - - - -

"{http://www.w3.org/1999/xhtml}a {http://www.w3.org/1999/xhtml}area" {}href =~ ^http(s)?:// - -
```

90. 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. Extends or creates a 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.

91. selectFile

Parameter syntax:

```
[
  [ '[' dialog_box_title ']' ]?

  '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 `save`*`xxx`* options are used, this parameter is used, not only to specify initial directory, but also to suggest a basename for the save file.

The optional *`dialog_box_title`* parameter may be used to specify a title for the dialog box. When this parameter is absent or empty, the dialog box will have a default title which depends on the specified action.

See also command selectConvertedFile [145], which has been designed to be used in **Convert** macro-commands such as `docb.convertToHTML1`, `xhtml.convertToPS`, etc.

Examples:

```
selectFile
selectFile [Save Configuration]saveFile
selectFile [Choose An Icon] openFileURL http://www.acme.com/doc/images/logo.gif
selectFile saveFileURL file:///tmp/article.pdf
```

92. selectConvertedFile

Parameter syntax:

```
[
  [ '[' dialog_box_title ']' ]?

  [ '[processCommand=' process_command_name ']' ]?

  'openFile'|'saveFile'|'openDirectory'|'saveDirectory'|
  'openFileURL'|'saveFileURL'|'openDirectoryURL'|'saveDirectoryURL'|
  saveFileWithExtension=extension|saveFileURLWithExtension=extension
```

```
  [ URL_template ]?

]?
```

Variant of command selectFile [144] specially designed to be used in **Convert** macro-commands such as `docb.convertToHTML1`, `xhtml.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.

This command supports two more “modes”:

saveFileWithExtension=*`file extension`*, saveFileURLWithExtension=*`file extension`*

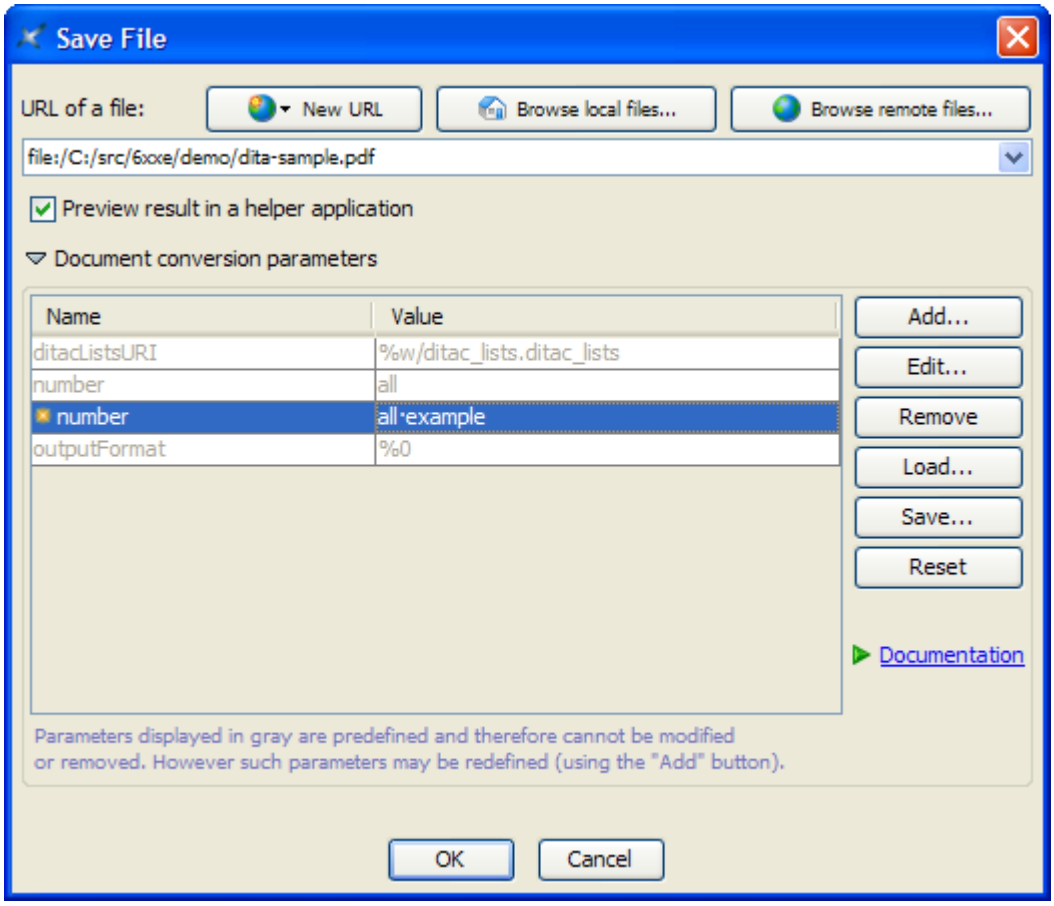
Identical to mode `saveFile` [`saveFileURL`], except that, when no *`URL_template`* has been specified,

- `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 or '.'.

This command also has a option `[processCommand=`*`process_command_name`*`]`. This option adds a "**Document conversion parameters**" panel to the dialog box. This panel allows to specify parameters for the first XSLT stylesheet (configuration element `transform`) having a named parameter group (configuration element `parameterGroup`) found in process command (configuration element `command`/`process`) having specified name.

*Figure 6.1. The "**Document conversion parameters**" panel once expanded*



Examples:

```
selectConvertedFile

selectConvertedFile saveFile

selectConvertedFile [Convert to WordML]saveFileWithExtension=wml

selectConvertedFile [Choose An Icon] openFileURL http://www.acme.com/doc/images/logo.gif

selectConvertedFile saveFileURLWithExtension=ps file:///tmp/article.pdf❶

selectConvertedFile []❷[processCommand=dita.toPS]❸ saveFileURLWithExtension=pdf
```

❶ 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`".

❷ An empty *`dialog_box_title`* may be used to specify default title.

❸ Process command `dita.toPS` looks like this:

```
<command name="dita.toPS">
  <process>
  ...
    <transform stylesheet="xsl/fo/fo.xsl" ...>
      ...
      <parameterGroup name="dita.toPS.transformParameters" />
    </transform>
  </process>
</command>
```

93. selectLink

Parameter syntax:

```
[ '[link_type_name]' ]?
'target' | 'source' | 'nextSource' | 'previousSource' | 'oppositeEnd'
```

This command allows to navigate between elements acting as link sources (simply called *links*) and elements acting as link targets (also called *anchors*).

This commands works when a `linkType` configuration element in *XMLmind XML Editor - Configuration and Deployment* has been defined in the configuration associated to the document being edited or, more simply, when the DTD or Schema of the document being edited makes use of `ID`/`IDREF`/`IDREFS`.

This command supports elements which act at the same type as a link source and as a link target (XHTML example: `<a id="here" href="#there">`). It also supports elements which are links to multiple targets (DocBook example: `<callout arearefs="co1 co2">`). When there is an ambiguity, this command displays a dialog box allowing the user to choose the destination.

In some cases (XHTML example: `<a href="foo.html#bar">`), this command allows to select an destination found in a document other than the one being edited. In such case, the user is prompted to confirm that she/he really wants to open this other document. This confirmation dialog box also allows the user to choose to open the other document in read-only mode.

target
: An element acting as a link must be implicitly or explicitly selected. Scrolls to and selects the element which is the target of this link (if found).

source
: An element acting as an anchor (that is, a link target) must be implicitly or explicitly selected. Scrolls to and selects the first element of the document which is a link to this anchor (if found).

nextSource
: An element acting as a link must be implicitly or explicitly selected. Scrolls to and selects following link element targeting the same anchor (if found).

previousSource
: An element acting as a link must be implicitly or explicitly selected. Scrolls to and selects preceding link element targeting the same anchor (if found).

oppositeEnd

An element acting as an anchor or acting as a link must be implicitly or explicitly selected. Scrolls to and selects opposite link end (if found).

An XXE configuration file may contain several `linkType` elements, each one having a different name. See Section 18, “linkType” in *XMLmind XML Editor - Configuration and Deployment*. Unless option [*`link_type_name`*] has been specified, this command acts on all types of links and anchors. If option [*`link_type_name`*] is specified, this command will act only on links and anchors belonging to the `linkType` whose name is *`link_type_name`*.

Examples:

```
selectLink target
selectLink [formControls] target
selectLink oppositeEnd
```

94. selectNode

Parameter syntax:

```
'parent' | 'child' | 'anyChild' |
'firstChild' | 'lastChild' | 'children' |
'previousSibling' | 'nextSibling' |
'firstSibling' | 'lastSibling' |
'ancestor' | 'ancestorOrSelf' | 'self' |
'descendant' | 'descendantOrSelf' |
'anyDescendant' | 'anyDescendantOrSelf' |
'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 [84].

child

Selects previously selected child of selected node [84].

If no child of selected node was previously selected, selects first child node of selected node [84].

Option `child` has not been designed to be used in macro-commands [25]. This option has been designed mainly to allow binding **Ctrl**+DOWN to command **selectNode `child`**. If you need to select the first child of selected element having a given element name or node type, please use option `anyChild` instead.

anyChild

Selects first found child node of selected node [84] having a given element name or node type. How to specify the element name or node type searched for is explained in Section 94.1, “List of element names or node types” [152].

firstChild

Selects first child node of selected node [84].

lastChild

Selects last child node of selected node [84].

children

Selects all the child nodes of selected node [84].

previousSibling

Selects preceding sibling of selected node [84].

nextSibling

Selects following sibling of selected node [84].

firstSibling

Selects first preceding sibling of selected node [84].

lastSibling

Selects last following sibling of selected node [84].

self

Selects selected node [84] 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 [84], starting at selected node [84]. Searched ancestor is specified using a list of names. See below [152].

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 [84], starting at parent of selected node [84]. Searched ancestor is specified using a list of names. See below [152].

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` *`one_or_more_element_names`* 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` *`one_or_more_element_names`* followed by `selectNode child` or `selectNode descendant` *`one_or_more_element_names`* in the same macro-command.

descendant

Selects previously selected descendant of selected node [84]. Searched descendant is specified using a list of element names or node types. See below [152].

If no descendants of selected node were previously selected, searches a descendant node *but only along the first child axis*. See also option `anyDescendant`, which is more general.

descendantOrSelf

Selects previously selected descendant of selected node [84]. Searched descendant is specified using a list of element names or node types. See below [152].

If no descendants of selected node were previously selected, searches a descendant node *but only along the first child axis*. See also option `anyDescendantOrSelf`, which is more general.

Selected node itself can be explicitly selected if it corresponds to searched node.

anyDescendant

Selects first found descendant node of selected node [84] having a given element name or node type. How to specify the element name or node type searched for is explained in Section 94.1, “List of element names or node types” [152].

anyDescendantOrSelf

Selects first found descendant node of selected node [84] having a given element name or node type. How to specify the element name or node type searched for is explained in Section 94.1, “List of element names or node types” [152].

Selected node itself can be explicitly selected if it corresponds to searched node.

precedingOrSelf

Selects preceding sibling of selected node [84], starting at selected node [84]. Searched sibling is specified using a list of names. See below [152].

preceding

Selects preceding sibling of selected node [84], starting at sibling of selected node [84]. Searched sibling is specified using a list of names. See below [152].

followingOrSelf

Selects following sibling of selected node [84], starting at selected node [84]. Searched sibling is specified using a list of names. See below [152].

following

Selects following sibling of selected node [84], starting at sibling of selected node [84]. Searched sibling is specified using a list of names. See below [152].

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
```

94.1. 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 simpara` selects first child of explicitly or implicitly selected element if and only if this first child is a `para` or a `simpara`

Example 2: `selectNode anyChild[implicitElement] para simpara` selects first found child of explicitly or implicitly selected element which is a `para` or a `simpara`.

Example 3: `selectNode ancestor itemizedlist orderedlist variablelist` selects first found ancestor of explicitly selected element which is a list.

94.2. 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.

95. selectNodeAt

Parameter syntax:

```
[ 'orParent' ]?
```

Selects the node clicked upon.

If the `orParent` option is specified, clicking again, exactly at 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 extendSelectionAt [108].

96. 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.

97. 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.

98. setProperty

Parameter syntax:

```
[ '[document]'|'[implicitElement]'|'[implicitNode]' ]?
[ '[remove]' ]?
[ '[rebuildView]' ]?
property_name [ property_value ]?
```

This command may be used to get, set or remove the property of a node. It is useful for writing macro-commands.

The default subject of this command is the explicitly selected node. Option `[document]` allows to select the document being edited. Option `[implicitElement]` allows to select the element containing the caret. Option `[implicitNode]` allows to select the text, comment or processing-instruction node containing the caret.

`property_name` specifies the qualified name of the property. Namespace prefixes (except "`xml`" which is always predefined) are not supported here. The syntax of a property name is:

```
property_name = non_qualified_name | {namespace_URI}local_part
```

`property_value` specifies the new value of the property. When this value is not specified and option `[remove]` has not been specified, this command returns the current value of the property as a string. When this value is not specified and option `[remove]` has been specified, this command removes the property from the node.

Option `[rebuildView]` allows to rebuild the view of the node for which a property has been added, updated or removed. This is a convenient alternative to invoking command refresh [134].

Examples:

```
setProperty myProp Hello world!
setProperty myProp
setProperty [remove] myProp
```

```
setProperty [implicitElement][rebuildView] {http://www.acme.com/ns/xxe}p1 1024
setProperty [implicitElement] {http://www.acme.com/ns/xxe}p1
```

Do not use this command to change the value of property `{http://www.xmlmind.com/xmleditor/namespace/property}readOnly`, instead use command setReadOnly [155].

Note that this command works even its subject node is not editable.

99. setReadOnly

Parameter syntax:

```
[ '[view]'|'[document]'|'[implicitElement]'|'[implicitNode]' ]?
[ 'false'|'true'|'remove'|'toggle' ]?
```

The default subject of this command is the explicitly selected node. The default operation is `toggle`.

If option `[view]` is specified, this command changes the flag that determines whether or not the current view of the current document is editable.

Parameter value:

false, remove
: Make the current view of the current document non-editable.

true
: Make the current view of the current document editable.

toggle
: Make the current view of the current document non-editable if it is editable and make it editable if it is non-editable.

Otherwise, this command changes the value of the `{http://www.xmlmind.com/xmleditor/namespace/property}readOnly` property of specified node.

Parameter value:

false
: Set the value of the `readOnly` property to `Boolean.FALSE`.

true
: Set the value of the `readOnly` property to `Boolean.TRUE`.

remove
: Removes property `readOnly` from specified node.

toggle
: Set the value of the `readOnly` property to the inverse of its current value. If specified node has no `readOnly` property, it is the value of the nearest ancestor having a `readOnly` property which is inversed.

Examples:

```
setReadOnly [view]
setReadOnly true
setReadOnly [document]toggle
setReadOnly [implicitElement] remove
```

See also `XXE.setReadOnly` [176].

100. setObject

Parameter syntax:

```
[attribute_name|'-']? S [data_type|'-']? S ['gzip'|'-']? S [URL_or_file]?

data_type --> 'anyURI'|'hexBinary'|'base64Binary'|'XML'
              [ '[' file_extension [',' file_extension]* ']' ]?
```

Interactive command displaying a dialog box letting the user change the object (generally an image) represented by explicitly selected element.

attribute_name

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

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

data_type (`anyURI`, `hexBinary`, `base64Binary`, `XML`)

Specifies how the object 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 or RELAX NG schema.

A data type may optionally be followed by a list of one or more file extensions. See examples below. This list is used when a file chooser dialog is displayed to let the user specify which file to use.

`gzip`

If this parameter is specified, object data is compressed using 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 the object.

If this parameter is absent, the dialog box may be used to specify this source file.

Examples:

```
setObject
setObject src anyURI
setObject fileref anyURI[png,jpg,jpeg,gif,svg,svgz]
setObject - hexBinary gzip
setObject location - - file://localhost/icons/apache_pb.gif
setObject - XML - C:\graphics\logo.svgz
setObject - XML[svg]
```

101. showContentModel

No parameter.

Displays a window (similar to the window displayed by **Help** → **Show Content Model**) showing the content model of implicitly or explicitly selected element. If there is no implicitly or explicitly selected element (for example, if several nodes have been selected), the window shows the content model of the root element.

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.

102. showElementReference

No parameter.

Opens in the Web browser the reference documentation of the explicitly or implicitly selected element.

For this command to work, a property in *XMLmind XML Editor - Configuration and Deployment* called "*`Configuration_Name`* `elementReference`" must have been defined in the **XXE** configuration file. This property specifies one or more reference manual locations separated by whitespace. A location is an absolute URL which must reference the `%{local-name}` variable. This variable is substituted with the local name of the selected element prior to invoking the Web browser in order to open the reference manual.

DITA examples:

```
<property name="$c elementReference">http://docs.oasis-open.org/dita/dita/¬
v1.3/os/part2-tech-content/langRef/ditaval/ditaval-%{local-name}.html</property>

<property name="$c elementReference">
http://docs.oasis-open.org/dita/dita/v1.3/os/part2-tech-content/¬
langRef/technicalContent/%{local-name}.html

http://docs.oasis-open.org/dita/dita/v1.3/os/part2-tech-content/¬
langRef/base/%{local-name}.html
</property>
```

103. showMatchingChar

Parameter syntax:

```
')' | '}' | ']'
```

Inserts specified character at caret position then, if the matching character (`'('`, `'{'`, `'['`) is found, highlights this matching character for half a second. If the matching character is not found, this command emits an audio beep.

This command honors the *Overwrite Mode* (**OVR**) in *XMLmind XML Editor - Online Help* when this mode has been turned on.

This command must be bound to the following keystrokes:

```
<binding>
  <charTyped char=")" />
  <command name="showMatchingChar" parameter=")" />
</binding>

<binding>
  <charTyped char="}" />
  <command name="showMatchingChar" parameter="}" />
</binding>

<binding>
  <charTyped char="]" />
  <command name="showMatchingChar" parameter="]" />
</binding>
```

Note that a binding such as "`<charTyped char="}"/>`" may not work on some platforms. For example, it does not work on Windows when using a French keyboard where `'}'` is typed by pressing **AltGr**+}.

104. showColumnRowLabels

Parameter syntax:

```
[ 'on' | 'off' | 'toggle' | 'state' ]?
```

This command may be used to add/remove A1-style labels to tables. These A1-style labels make a table look a little like a spreadsheet.

This command has obviously no effects on a tree view, only on a styled view. See also Section 10, “Making a table look like a spreadsheet” in *XMLmind XML Editor - Support of Cascading Style Sheets (W3C CSS)*.

Options:

on
: Make sure that A1-style labels are visible.

off
: Make sure that A1-style labels are hidden.

toggle
: Change the visibility of A1-style labels.

state
: Default option. Does nothing. Just useful to learn whether A1-style labels are currently visible.

This command returns `true` is A1-style labels are currently visible, `false` otherwise.

This command is mainly useful to XML consultants and Java™ developers customizing or extending **XXE**.

105. spellCheck

No parameter.

Displays a modal dialog box which allows to check the spelling of the document being edited (same as the **Spell** tool in XMLmind XML Editor).

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.

106. split

Parameter syntax:

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

Splits explicitly or implicitly selected element in two parts, the split point being specified by caret position.

107. start

Similar to run [139] except that external command is executed asynchronously (like Windows `start` or Unix `&`).

108. 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
```

109. toggleCollapsed

Parameter syntax:

```
[
  ('showLevel' '1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9') |
  'state'|'toggle'|'collapse'|'expand'|'collapseAll'|'expandAll'
]?
```

The "`showLevel N`" option allows to expand all the collapsible views of the document up to nesting level `N` and to recursively collapse all the collapsible views having a nesting level greater than `N`. For example, invoking "`toggleCollapsed showLevel 1`" or "`toggleCollapsed showLevel 2`" just after opening a large DocBook document may be used to display the outline of this document (that is, the list of its collapsed parts or chapters).

All the other options: `state`, `toggle`, `collapse`, etc, change 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.

If this collapsible view is not found, then this command cannot be executed.

If this collapsible view is found, then this command returns a string, "`expanded`" or "`collapsed`", which reflects the state of the collapsible view after applying the operations specified in its parameter.

The default operation is `toggle`. Supported operations are:

`state`
: Does nothing at all. Allows to obtain the current state, "`expanded`" or "`collapsed`", of the collapsible view.

`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.

Sample bindings (as found in the add-on called "*A sample customize.xxe*" — download and install it using **Options** → **Install Add-ons**):

```
<binding>
  <keyPressed code="ESCAPE" />
  <charTyped char="1" />
  <command name="toggleCollapsed" parameter="showLevel 1" />
</binding>

<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>
```

110. toggleTextStyle

Parameter syntax:

```
name [ '[' attr_specs ']' ]?

attr_specs -> attr_spec [ S 'and' S attr_spec ]*

attr_spec -> 'not(' attr_simple_spec ')'

attr_simple_spec -> '@' attr_name [ '=' quoted_attr_value ]?
```

This command emulates the behavior of the **Bold**, **Italic**, **Underline**, etc, toggles found in the tool bars of almost all word-processors. Such toggles add a given text style to text spans not having this text style and remove a given text style to text spans already having this text style. Hence the name *“text style” toggle*.

Note that such toggles do not require some text to be explicitly selected. If the caret is in the middle of a word, then the toggle acts on this word. If the caret is before or after a word, the toggle acts on this text location by inserting an empty text span having or on the contrary, not having, the corresponding text style.

This command supports the node selection in addition to the text selection. For example, if you want to apply a “text style” to an element in its entirety, suffice to explicitly select it. This is more handy that selecting all the text this element contains.

name

Name of an element which behaves like a *text style*. Such elements must have a mixed content and may contain text as well as other text styles in arbitrary order and number of occurrences.

DocBook 4 example, literal text style:

```
literal
```

Equivalent DocBook 5 example:

```
{http://docbook.org/ns/docbook}literal
```

Notice in the above example that XML names belonging to a namespace are specified using the Clark's notation [85].

attr_name

An element which behaves like a text style may have at most one required attribute. When this is the case, this attribute *must* be specified in the parameter of the `toggleTextStyle` command.

DocBook 4 example, external link text style:

```
ulink[@url]
```

Equivalent DocBook 5 example:

```
{http://docbook.org/ns/docbook}link[@{http://www.w3.org/1999/xlink}href]
```

quoted_attr_value

In some cases, an element which behaves like a text style has an attribute which specifies a given style. When this is the case, this attribute and its specific value *must* be specified in the parameter of the `toggleTextStyle` command.

DocBook 4 example, bold text style:

```
emphasis[@role='bold']
```

Equivalent DocBook 5 example:

```
{http://docbook.org/ns/docbook}emphasis[@role='bold']
```

This attribute value may contain a *variable*. Currently the only supported variable is `%{url}`.

DocBook 4 example, external link text style:

```
ulink[@url='%{url}']
```

or equivalently (`|`*`fallback_value`*, where by default *`fallback_value`* is "`???`", is implicit):

```
ulink[@url='%{url|???}']
```

The above specification means: convert the selection to an `ulink` element having an `url` attribute. If the text contained in the selection can be parsed as an absolute URL (e.g. "`https://www.xml.com/`") then use this text as the value of the `url` attribute, otherwise use "`???`" as the value of the `url` attribute.

attr_spec and *attr_spec* ... and *attr_spec*

Operator `and` means: this text style corresponds to specified element if this element has attributes matching *all* attribute specifications.

DITA example 1, external link style:

```
xref[@href and @scope='external' and @format='html']
```

DITA example 2, internal link text style:

```
xref[@href and not(@scope='external')]
```

not(*attr_simple_spec*)

Function `not()` means: this text style does *not* correspond to specified element if this element has specified attribute (whatever its value; see DocBook example below) or if this element has specified attribute value (see DITA example below).

DocBook example, emphasis text style:

```
emphasis[not(@role)]
```

DITA example, internal link text style:

```
xref[@href and not(@scope='external')]
```

111. undo

No parameter.

Undo last command.

112. uninclude

Parameter syntax:

```
[ '[all]' | '[lookupInclusion]' ]?
```

When option `[all]` is specified, this command untranscludes all the included nodes found in the document being edited. Untranscluding means: replacing some included nodes by the corresponding inclusion directive (e.g. `xi:include` element).

Otherwise, this command replaces the selected included nodes by the corresponding inclusion directive. Unless option `[lookupInclusion]` has been specified, one of the topmost included nodes must be explicitly selected. If option `[lookupInclusion]` has been specified, the selection, or even just the caret, may be anywhere inside the included nodes.

See also reinclude [134].

113. updateInclusions

No parameters

Updates all the references contained in current document. That is, replaces all included nodes by up-to-date nodes read from the last saved copy of referenced documents.

114. viewObject

Parameter syntax:

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

Opens in associated helper application, the ``object'' contained or represented by implicitly or explicitly selected element.

This command may be used, for example, to open an image in an external image viewer, to open a PDF file in Adobe Acrobat Reader, etc.

The parameter may be used to specify where to find the object of interest and also the data type of this object:

attribute_name

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

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

anyURI, hexBinary, base64Binary, XML

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

If this parameter is absent, 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 also for *invalid* documents conforming to a W3C XML or RELAX NG schema.

When the parameter is absent, this command displays a dialog box allowing the user to choose the attribute or element of interest from a list.

Helper applications are declared using the **Preferences** dialog box, **Helper Applications** section. This registry is searched to find an application capable of opening the contents of the selected attribute or element. When no suitable helper application is found in this registry, the user is prompted to specify one.

Note that this command considers that the default viewer (specified in the **Preferences** dialog box, **Helper Applications** section, **Default viewer** field; typically a Web browser) should be able to open HTML, text, GIF, JPEG and PNG files[3]. Therefore, in last resort, it may end up invoking the default viewer.

Examples:

```
viewObject
viewObject [implicitElement]
viewObject fileref anyURI
viewObject - XML
viewObject [implicitElement] {http://www.w3.org/1999/xlink}href
```

See also editObject [104].

[3]Command `viewObject` also considers that the default viewer should be able to open URLs starting with "`http://`" and "`https://`". DocBook example: this is handy for displaying `<ulink url="http://www.xmlmind.com/xmleditor/"/>`.

115. webSearch

Parameter syntax:

```
'[' search engine name ']' S [ searched words ]?
```

Invoke the web browser in order to perform a web search.

Which search engine to use is specified by *`search engine name`*. This search engine must have been declared as explained in Section 115.1, “Declaring search engines” [165].

The searched text is optionally specified by *`searched words`*. When *`searched words`* are not specified, the searched text is the text selected in the active document view.

Examples:

```
webSearch [Google]
webSearch [Google] XMLmind XML Editor
```

115.1. Declaring search engines

Search engines are normally declared using **Options** → **Preferences**, **Web Search** section in *XMLmind XML Editor - Online Help*.

Alternatively, search engines may be declared in system property WebSearch.services. System property WebSearch.services must contain *`search_engine_name`*/*`search_engine_URL`* pairs separated by newline characters. Open lines are ignored.

When search engines have been declared in both the user preferences (that is, using the **Preferences** dialog box) and system property WebSearch.services, those declared in the user preferences supersede those declared in the system property.

When this system property is missing, this is equivalent to having the following `property` configuration element in your `customize.xxe` file:

```
<property name="WebSearch.services"><![CDATA[
Google
https://www.google.com/search?q=${searched}&hl=${locale2}&lr=lang_${lang2}

Google (phrase)
https://www.google.com/search?as_epq=${searched}&hl=${locale2}&lr=lang_${lang2}

Wikipedia
https://${lang2}.wikipedia.org/w/index.php?title=Special%3ASearch&search=${searched}&fulltext=S

TheFreeDictionary
https://${lang2}.thefreedictionary.com/${searched}

Wiktionary
https://${lang2}.wiktionary.org/wiki/${searched}

Google Translate
```

```
https://translate.google.com/?q=${searched}&sl=${lang2}&tl=fr❶
]]></property>
```

❶ The above URI translates some text to French (`&tl=fr`). If you want to specify another target language, simply replace the language code for French, `fr`, by another language code. For example, Italian: `https://translate.google.com/?q=${searched}&sl=${lang2}&tl=it`.

The following variables are substituted in *`search_engine_URL`*:

Variable	Description
`${searched}`	Searched text.
`${locale}`	The language of the user interface (GUI) of **XXE**.
`${locale2}`	Same as `${locale}`, but just the first two ISO 639-1 letter codes. Example: if `${locale}` is `fr-CA`, then `${locale2}` is `fr`.
`${lang}`	The language of searched text. This language is automatically determined as follows: • This language is the value of attribute `languageAttribute` of the `spellCheckOptions` configuration element in *XMLmind XML Editor - Configuration and Deployment*. When `languageAttribute` is not specified, the attribute lookup uses `xml:lang`. • When the searched text is the selected text, the lookup starts at the element containing the caret. When the searched text is specified in the parameter of the command, the lookup starts (and ends) at the root element of the document. • If the attribute lookup didn't give any result, the language used here is the value of attribute `defaultLanguage` of the `spellCheckOptions` configuration element in *XMLmind XML Editor - Configuration and Deployment*. When attribute `defaultLanguage` is not specified, the language used here is taken from field "**Default language**" of the "**Check Spelling**" dialog box in *XMLmind XML Editor - Online Help*.
`${lang2}`	Same as `${lang}`, but just the first two ISO 639-1 letter codes.

The value of a variable is automatically escaped if needed too. For example, "`vin rosé`" becomes "`vin%20ros%C3%A9`".

116. wrap

A variant of the convert [92] command. The unique difference is that when a single element is selected, the selected element is given a new parent element.

Example:

```
<simpara id="p1">the <emphasis>little</emphasis> girl.</simpara>
```

wrapped in a `note` gives

```
<note>
  <simpara id="p1">the <emphasis>little</emphasis> girl.</simpara>
</note>
```

This is different from command convert [92] which can be used to “morph” selected element to another kind of element. For example, convert [92] cannot wrap the above `simpara` in a `note` but can morph it to a `para`.

Examples:

```
wrap
wrap [implicitElement] div
```

117. xIncludeText

Parameter syntax:

```
[ file_or_URL ]?
```

Includes at caret position the contents of a text file (of any kind: XML, HTML, `.bat`, C/C++, etc). This command allows to easily create documents containing `xi:include parse="text"` elements. Thus, this command is disabled when the document being edited does not support the XInclude inclusion scheme.

Optional parameter *`file_or_URL`* specifies the location of the text file to be included. When this parameter is not specified, a file chooser dialog box is displayed allowing to choose the text file to be included.

Examples:

```
xIncludeText
xIncludeText /home/john/src/hello.c
```

118. xpathSearch

Parameter syntax:

```
[ implicit_selection ]? [ XPath_expression ]?
```

Evaluates specified XPath 1.0 expression in the context of selected node [84].

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.

Note that for this command, *implicit_selection* defaults to `[implicitNode]`.

If *XPath_expression* is not specified, a dialog box is displayed. This dialog box may be used to specify arbitrarily complex XPath 1.0 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. Note that for a greater ease of use, the default namespace if any is also considered when parsing element names. DocBook 5 example: `following::para` is equivalent to `following::db:para`.

When this command is used in an **XXE** configuration file, namespace prefixes may be different in the configuration file and in the document being edited so it's safer to use the {*namespace_URI*}*local_part* notation.

Examples:

```
xpathSearch
xpathSearch [implicitNode]
xpathSearch //@revisionflag
xpathSearch [implicitElement] preceding::li
xpathSearch [implicitElement] following::xs:complexType[1]
xpathSearch [implicitElement] following::{http://www.w3.org/2001/XMLSchema}complexType[1]
```

119. 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 `CommandResult.CANCELED` if user has canceled the command. Otherwise, returns `CommandResult.DONE`.

120. XXE.compare

Parameter syntax:

```
[ '[verticalSplit]' ]? original_file_or_URL [ revised_file_or_URL ]?
```

Automate the comparison of two revisions of the same initial document for which the comparison of revisions has been enabled (Background information about the compare revisions feature: the **Tools** → **Revisions** submenu in *XMLmind XML Editor - Online Help*, the **Compare** tool in *XMLmind XML Editor - Online Help*.)

The original document is specified by *original_file_or_URL*. This path may be a filename or an URL. A relative filename is relative to the current working directory. A relative URL is relative to the active document of **XXE**.

The revised copy is specified by *revised_file_or_URL*. When this parameter is absent, the revised copy is the active document of **XXE**.

If option `[verticalSplit]` is specified, this command ensures that the document area is displayed is split vertically in two parts (as if the user checked **Window** → **Split Windows Vertically**).

Examples:

```
XXE.compare [verticalSplit] art23.xml~
XXE.compare ../old_docs/art23.xml
XXE.compare file:/home/john/docs/art23.xml file:/home/john/old_docs/art23.xml
```

Useful macro-command making use of this command: compare a document with its backup file (having a '~' suffix) automatically created by **XXE**:

```
<command name="compareWithBackup">
  <macro>
    <sequence>
      <choice>
        <command name="XXE.open [173]" parameter="[checkIsOpened] %{d}~"/>
        <command name="XXE.open" parameter="%{d}~"/>
      </choice>
      <command name="XXE.compare" parameter="[verticalSplit] %{d}~"/>
    </sequence>
  </macro>
</command>
```

121. XXE.edit

Parameter syntax:

```
[ '[master=' [ master_document_URL ]? ']' ]?
[ '[confirm]' ]? [ '[readOnly]'|'[readWrite]' ]? [ 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.

Note that it's possible to specify several documents to be opened. In this case, it's first specified document which is made the “active” one.

Parameters:

`[master=`*`master_document_URL`*`]`

Advanced, rarely needed, option. No effect if the document is already opened in **XXE**. Informs **XXE** that the document is possibly a module which is being opened from master document *`master_document_URL`*. URL *`master_document_URL`* defaults to the URL of the document from which the `XXE.edit` command has been invoked.

`[confirm]`

When the document is not already opened in **XXE**, this option may be used to display a dialog box asking the user to confirm that she/he really wants to open this document in **XXE**. This dialog box also allows the user to specify whether the document is to be opened in read-only mode[4].

[4]The initial state of the read-only check box found in this dialog box is of course taken from parameter `[readOnly]`.

`[readOnly]`
: Allows to open in read-only mode the document specified by the other parameters.

`[readWrite]`
: Allows to open in normal read-write mode the document specified by the other parameters.

`file_name` or *`URL`*
: Opens or activates specified document.

  An URL may have a fragment. Example: `file:/home/john/docs/report122211/report.xml#conclusion`. If the element specified by fragment exists, **XXE** will automatically scroll to show this element and then it will select it.

Returns `CommandResult.DONE` having newly opened or newly activated `com.xmlmind.xml.doc.Document` as its value (for use by higher-level commands) or `CommandResult.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>
```

122. XXE.editInclusion

Parameter syntax:

```
[ '[readOnly]' ]?
```

If an *included* element or node is implicitly or explicitly selected, edit in **XXE** the document which is the source of the inclusion. Equivalent to menu item **Edit** → **Reference** → **Edit Referenced Document**.

When option `[readOnly]` has been specified and the referenced document needs to be opened in **XXE**, then this document is opened in read-only mode.

Examples:

```
XXE.editInclusion
XXE.editInclusion [readOnly]
```

123. XXE.masterDocumentControl

```
'set' | 'unset' | 'toggle' | 'state' | 'refresh' | 'reset'
```

This command allows to create a *document set* containing a master document of any kind[5] and its module documents.

By grouping a master document and its module documents, you inform XMLmind XML Editor (**XXE**) that all the module documents referenced or included, directly or indirectly, by the master document are related. When **XXE** knows that some of the opened documents are related:

- it will more thoroughly check the cross-references which may exist between these documents;
- it will make it easier creating cross-references between these documents;
- it will make it easier following cross-references between these documents;
- if a DITA map contains key definitions, then this map acts not only as a cross-reference creation/validation context for its topics, but it also acts as a *key space*;
- if the "**Easy Profiling**" add-on has been installed, then the conditional processing profile selected for the master document is automatically shared by all module documents;
- if the document view area is split in two parts, the module documents opened from a master document will appear at the opposite of this master document. This allows to use the view of the master document as a rudimentary navigation pane.

Options:

`set`

Create the document set specified by the master document being edited.

For this option to work, a `documentSetFactory` in *XMLmind XML Editor - Configuration and Deployment* element must have been declared in the configuration of the document being edited.

`unset`

Delete the document set containing the document being edited. This document may be a module document or a master document.

`toggle`

Create the document set specified by the master document being edited if this set does not already exist; delete the set otherwise. This option requires the document being edited to be a master document, not a module document.

Command `XXE.masterDocumentControl` with option `toggle` is used to implement **Tools** → **Use as Master Document**.

`state`

Returns "`on`" if the document being edited is contained in a document set. Returns "`off`" otherwise.

`refresh`

Update the document set containing the document being edited. This document may be a module document or a master document.

Command `XXE.masterDocumentControl` with option `refresh` is automatically used when the master document is saved to disk using **File** → **Save**.

`reset`

Recreate the document set containing the document being edited. This document may be a module document or a master document.

[5]DITA map, DocBook 5.1+ assembly, Ebook, DocBook modular document such a book including chapters by the means of XInclude, etc.

Command `XXE.masterDocumentControl` with option `reset` is automatically used when the master document is saved to disk using **File** → **Save As**.

Examples:

```
XXE.masterDocumentControl toggle
XXE.masterDocumentControl state
```

124. XXE.new

Parameter syntax:

```
[
  category_name | '-'
  template_name | '-'
  [ save_file_or_URL | '-'  [ '[createOnly]' ]? ]?
]?
```

Creates a new document by copying a named template (that is, a document template which has been declared in an **XXE** configuration file).

When *`category_name`* and *`template_name`* are absent or specified as "`-`", the command displays the **File** → **New** dialog box to let the user choose a document template. When this is the case and when *`category_name`* is different from "`-`", the **File** → **New** dialog box preselects specified category.

Parameters:

`category_name`

Specifies the (case-insensitive) name of the *category* of the document template. A category consists in one or more segments separated by character `'/'`.

By default, the category of a document template is the name of the configuration in which this template has been specified.

Example 1: actual category : `XHTML/1.0` or `xhtml/1.0` (category "`XHTML/1.0`" includes all the templates specified in configurations "`XHTML Strict`" and "`XHTML Transitional`").

Example 2: actual category : `XHTML/5` or `xhtml/5` (category "`XHTML/5`" includes all the templates specified in configuration "`XHTML 5`").

Example 3: category which is in fact a configuration name: `DocBook` or `docbook` (the document templates of DocBook 4 are not sorted into categories).

`template_name`

Specifies the (case-insensitive, possibly localized) name of a document template. Example: "`Seite (Streng)`" ("`Page (Strict)`" in German).

Alternatively, you can specify the basename —with or without a file extension— of the file containing the document template. Doing this should work whatever your locale. XHTML/1.0 example: "`page_strict.xhtml`". DocBook example: "`refentry`".

Note that the **XXE.new** command will fail if there are several document templates in specified category having the same basename. XHTML/1.0 example: "`page_strict`". (XHTML/1.0 contains both "`page_strict.xhtml`" and "`page_strict.html`".)

`save_file_or_URL`

When this argument is specified as a filename or URL, the newly created document is immediately saved to specified location.

When this argument is specified as "`-`", the command displays the file chooser dialog box to let the user specify a save location for the newly created document. After this, the newly created document is immediately saved to specified location.

When this argument is absent, the newly created document is automatically given a save location but it is not actually saved to this location (that is, the command behaves like menu item **File** → **New**).

`[createOnly]`

Create the new document and save it to a file, but do *not* open the new document in **XXE** just after creating it.

Returns `CommandResult.DONE having` newly created `com.xmlmind.xml.doc.Document` as its value (for use by higher-level commands) or `CommandResult.FAILED` if specified template cannot be opened or `CommandResult.CANCELED` if user has canceled the command.

Examples:

```
XXE.new
XXE.new - -
XXE.new "TEI Lite" -
XXE.new - - -
XXE.new "TEI Lite" - -
XXE.new - - /tmp/doc.xml
XXE.new docbook refentry
XXE.new XHTML/1.0 "HTML Page (Transitional)"
XXE.new xhtml/1.0 page_strict.html /tmp/contact.html
XXE.new "XHTML/1.0" page_strict.html /tmp/news.html [createOnly]
```

125. XXE.open

Parameter syntax:

```
[ '[readOnly]' ]?
[
  '[reopen]' |
  '[reopenIfNewer]' |
  ('[checkIsOpened]' S file_name_or_URL) |
  file_name_or_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:

[readOnly]
: This parameter is a modifier which allows to open in read-only mode the document specified by the other 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.xml.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.

[checkIsOpened]
: The command cannot be executed unless specified document has been opened in XXE. If specified document is already opened in XXE, this command just returns it (a `com.xmlmind.xml.doc.Document` object) which may be useful to write higher-level commands.

`file_name_or_URL`
: Opens specified document.

Returns `CommandResult.DONE` having newly opened `com.xmlmind.xml.doc.Document` as its value (for use by higher-level commands), `CommandResult.FAILED` if specified document cannot be opened or `CommandResult.CANCELED` if user has canceled the command.

See XXE.save [174] for an example of use for this command.

126. XXE.save

Parameter syntax:

```
[ '[ifNeeded]' | [checkIsSaved] ]?
```

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 first example below.

[checkIsSaved]

With this option, the command does nothing at all. However it cannot be successfully executed unless the current document does not need to be saved.

This option is useful to write macro commands such as the one in the second example below.

Returns `CommandResult.FAILED` if document cannot be saved or if user has canceled the command. Otherwise, returns `CommandResult.DONE`.

Example 1:

```
  <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.

Example 2: Do not attempt to convert the DITA document to XHTML if this document needs to be saved.

```
<command name="dita.convertToXHTML">
    <macro>
      <sequence>
        <command name="XXE.save" parameter="[checkIsSaved]" />
        <command name="selectConvertedFile"
                 parameter="saveDirectoryURL" />
        <command name="dita.toXHTML" parameter='"%_"' />
      </sequence>
    </macro>
  </command>
```

127. XXE.saveAll

Parameter syntax:

```
[ '[ifNeeded]' ]?
```

Saves all the documents (which actually need to be saved) currently opened in XXE.

[ifNeeded]

Without this option, this command cannot be executed if no documents at all need to be saved.

With this option, this command does nothing but can be successfully executed even when no documents at all need to be saved.

Returns `CommandResult.FAILED` if no documents at all need to be saved (when `[ifNeeded]` is not specified) or if some of the documents which need to be saved, cannot be saved. Otherwise, returns `CommandResult.DONE`.

Example: converting a map to PDF requires all the topics referenced in this map to have been saved to disk.

```
<command name="map.convertToPDF">
  <macro>
    <sequence>
      <command name="XXE.saveAll" parameter="[ifNeeded]" />
      <command name="selectConvertedFile"
               parameter="saveFileURLWithExtension=pdf" />
      <command name="map.toPDF" parameter="%_" />
    </sequence>
  </macro>
</command>
```

128. XXE.setReadOnly

Parameter syntax:

```
[ 'true'|'false'|'toggle' ]? [ file_name | URL ]?
```

Changes the state of specified document from read-only to editable or the other way round.

Unlike command `setReadOnly` [155], this command also acts on the write lock, if any, of specified document:

- When the state of a locked document is changed from editable to read-only, the lock is automatically removed.
- When the state of a document is changed from read-only to editable (and the user wishes her/his documents to be locked), a lock is automatically added.

Parameters:

true
: Ensures that specified document is in read-only state.

false
: Ensures that specified document is in editable state.

toggle
: Changes the state of specified document from read-only to editable or the other way round. This is the default option.

file_name or *URL*
: Specifies document location. By default, applies to the current document.

Returns `null`.

129. 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 `CommandResult.FAILED` if an error occurred when saving the document or if user has canceled the command. Otherwise, returns `CommandResult.DONE`.

130. A generic, parameterizable, table editor command

Parameter syntax:

```
'insertColumnBefore'|'insertColumnAfter'|
'cutColumn'|'copyColumn'|
'pasteColumnBefore'|'pasteColumnAfter'|
'deleteColumn'|
'insertRowBefore'|'insertRowAfter'|
'cutRow'|'copyRow'|
'pasteRowBefore'|'pasteRowAfter'|
'deleteRow'|
'incrColumnSpan'|'decrColumnSpan'|
'incrRowSpan'|'decrRowSpan'
```

This command may be used to edit any table conforming to a model vaguely resembling the HTML table model (table contains rows, themselves possibly contained in row groups, etc).

Prerequisite in terms of selection	Parameter	Description
A cell or an element having a cell ancestor must be implicitly or explicitly selected.	`insertColumnBefore`	Insert a column before column containing specified cell.
	`insertColumnAfter`	Insert a column after column containing specified cell.
	`cutColumn`	Cut to the clipboard the column containing specified cell.
	`copyColumn`	Copy to the clipboard the column containing specified cell.

Prerequisite in terms of selection	Parameter	Description
	`pasteColumnBefore`	Paste copied or cut column before column containing specified cell.
	`pasteColumnAfter`	Paste copied or cut column after column containing specified cell.
	`deleteColumn`	Delete the column containing specified cell.

	`sortRows`	Sort all the rows of the table according to the string values of the cells of the “selected column”. (The “selected column” is the column containing specified cell.)
		
		Unless `sortRows` is immediately followed by (optional) parameter
		
		`'['`
		`'dictionary'|'numeric'|'lexicographic'`
		`[ ';descending'|';ascending' ]?`
		`']'`
		
		a dialog box is displayed allowing to specify the following sort options:
		
		Order
		
		Dictionary is the language-specific alphabetical order. Example: (`Charles`, `best`, `Albert`) is sorted as (`Albert`, `best`, `Charles`).
		
		Numeric. The string value of a cell is expected to start with a number. Example: (`+15.0%`, `1.50%`, `-20%`) is sorted as (`-20%`, `1.50%`, `+15.0%`).
		
		Lexicographic is the order of Unicode characters. Example: (`Charles`, `best`, `Albert`) is sorted as (`Albert`, `Charles`, `best`).
		
		Dictionary and **Numeric** orders will cause this command to fail, unless the language of the table can be determined (e.g. lookup for the `xml:lang` attribute).
		Direction
		
		Ascending means: A to Z, low to high. **Descending** means: Z to A, high to low.
		
		Note that:
		
		• Header/footer rows (e.g. `thead`) are never sorted.
		• The contents of row groups (e.g. `tbody`) are sorted separately.

A row must be explicitly selected. OR a cell or an element having a cell	`insertRowBefore`	Insert a row before row containing specified cell.
	`insertRowAfter`	Insert a row before row containing specified cell.
	`cutRow`	Cut to the clipboard the row containing specified cell.

Prerequisite in terms of selection	Parameter	Description
ancestor must be implicitly or explicitly selected.	`copyRow`	Copy to the clipboard the row containing specified cell.
	`pasteRowBefore`	Paste copied or cut row before row containing specified cell.
	`pasteRowAfter`	Paste copied or cut row after row containing specified cell.
	`deleteRow`	Delete the row containing specified cell.
A cell or an element having a cell ancestor must be implicitly or explicitly selected.	`incrColumnSpan`	Increment the number of columns spanned by specified cell.
	`decrColumnSpan`	Decrement the number of columns spanned by specified cell.
	`incrRowSpan`	Increment the number of rows spanned by specified cell.
	`decrRowSpan`	Decrement the number of rows spanned by specified cell.

Unlike the other commands contained in this reference, *this command has no fixed name*. It must be instantiated and given a name using a `command` configuration element (see Section 4, “command” in *XMLmind XML Editor - Configuration and Deployment*). It must also be parametrized using a simple specification contained in a `property` configuration element. See example below:

DITA `simpletable` example:

```
<command name="dita.simpleTableEdit">❶
  <class>com.xmlmind.xmleditapp.cmd.table.GenericTableEdit</class>
</command>

<property name="dita.simpleTableEdit.tableSpecification">❷
  table=simpletable
  row=sthead:header strow
  cell=stentry
</property>
```

❶ This creates an instance of generic, parameterizable, table editor command `com.xmlmind.xmleditapp.cmd.table.GenericTableEdit` called `dita.simpleTableEdit`.

❷ Because the table editor command is called `dita.simpleTableEdit`, a property called `dita.simpleTableEdit.tableSpecification` should exist too. *The value of this property maps element names and attribute names to roles understood by the generic table editor command.*

Example 1: "`cell=th td`" specifies that an element with name `th` or `td` should be considered by the generic table editor as being a cell.

Example 2: "`rowSpan=morerows+1`" specifies that attribute `morerows`, if found in cell elements, contains the number of additional rows spanned by the cell.

In the above example, the fact that the `rowGroup=`, `rowSpan=` and `columnSpan=` lines are missing means that this table model does not have the concept of row groups and that there are no attributes which could be used to specify the number of rows and the number of columns spanned by a cell.

The syntax of a table specification is:

```
spec -> table_spec row_group_spec? row_spec cell_spec
        row_span_spec? column_span_spec?

table_spec -> table= element_name_list \n

row_group_spec -> rowGroup= element_name_role_list \n

row_spec -> row= element_name_role_list \n

cell_spec -> cell= element_name_list \n

row_span_spec -> rowSpan= attribute_name_list \n

column_span_spec -> columnSpan= attribute_name_list \n

element_name_role_list -> name_role {S name_role}*

name_role -> name{role}?

role -> :header | :footer | :body

element_name_list -> name {S name}*

attribute_name_list -> name{+1}? {S name{+1}?}*

name = non_qualified_name | {namespace_URI}local_part
```

table=

Specifies the names of the elements which must be considered as being tables, that is, row group containers or, directly, row containers (like in HTML 3.2 tables).

rowGroup=

Specifies the names of the elements which must be considered as being row groups, that is, row containers. May be omitted if not relevant.

The name of an element may be optionally followed by `:header` if the corresponding row group is a table header, `:footer` if the corresponding row group is a table footer and `:body` if the corresponding row group is a table body. By default, a row group is considered to be a table body.

row=

Specifies the names of the elements which must be considered as being rows, that is, cell containers.

The name of an element may be optionally followed by `:header` if the corresponding row is a table header, `:footer` if the corresponding row is a table footer and `:body` if the corresponding row is a normal row. By default, a row is considered to be a normal row.

cell=

Specifies the names of the elements which must be considered as being cells.

rowSpan=

Specifies the names of the attributes which are used to specify the number of rows spanned by a cell. May be omitted if not relevant.

Use `+1` to specify that the attribute contains an *additional* number of rows rather than the actual number of rows spanned by a cell.

columnSpan=

Specifies the names of the attributes which are used to specify the number of columns spanned by a cell. May be omitted if not relevant.

Use `+1` to specify that the attribute contains an *additional* number of rows rather than the actual number of rows spanned by a cell.

Example 1: the specification of an XHTML table would be:

```
table={http://www.w3.org/1999/xhtml}table
rowGroup={http...ml}tbody {http...ml}thead:header {http...ml}tfoot:footer
row={http...ml}tr
cell={http...ml}td {http...ml}th
rowSpan=rowspan
columnSpan=colspan
```

Notice how the XHTML namespace is specified before the local name of each element.

Example 2: a partial specification for CALS (DocBook 4; no namespace) tables would be:

```
table=tgroup entrytbl
rowGroup=tbody thead:header tfoot:footer
row=row
cell=entry
rowSpan=morerows+1
```

The fact that the `columnSpan=` line is missing means that there is no *attribute* which could be used to specify the number of columns spanned by a cell.

Appendix A. Description of the XML differencing algorithm implemented by the Compare tool

1. Comparison with other approaches

	Generic XML Differencing Tool	XMLmind Compare Tool	Change Tracking
Main function	Show *what* has been changed.	Show *what* has been changed. Inserting remarks in *XMLmind XML Editor - Online Help* in a revision allows to specify by whom, when and why.	Show *how* changes have been made, by whom, when and possibly why.
How does it work?	Compare two arbitrary XML files.	Compare two revisions of the same initial document (in which the comparison of revisions has been enabled in *XMLmind XML Editor - Online Help*).	Record insertions and deletions in the revised document, typically in the form of proprietary processing-instructions.
Needs to be activated for a given document	No.	Yes.	Yes.
Performance penalty when loading and saving the document	No.	Not significant.	Possibly yes, when the number of changes becomes large.
Performance penalty when editing the document	No.	No.	Possibly yes, when the number of changes becomes large.
Increases the size of the revised document	No.	Yes. The increase in size depends on the number of elements contained in the document. See Section 2, “Elements are given serial numbers” [183].	Yes. The increase in size depends on the number of changes made to the document.
Detects all changes	Highly depends on the quality of the underlying algorithm. Yes, for some algorithms.	Yes.	No, cannot handle attributes.
Allows to accept or reject one or more changes	Yes.	Yes.	Yes.
Validity of the document after accepting or	Well-formed, may be valid.	Well-formed. In practice, generally valid.	Valid.

	Generic XML Differencing Tool	XMLmind Compare Tool	Change Tracking
rejecting one or more changes			

See also: "*Approaches to change tracking in XML*" by Robin La Fontaine.

2. Elements are given serial numbers

The XML differencing algorithm implemented in the **Compare** tool in *XMLmind XML Editor - Online Help* requires all the elements to have a globally unique ID. This GUID, which is called a *serial number*, is assigned by XMLmind XML Editor (**XXE** for short) to an element during its lifetime and will not change no matter how you'll modify the content of this element.

Menu item **Tools** → **Revisions** → **Enable the Comparison of Revisions** in *XMLmind XML Editor - Online Help* merely instructs **XXE** to systematically assign a serial number to all the elements contained in the document being edited.

These serial numbers are all stored in `<?xxe-sn>` processing-instructions. Example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<section version="5.0" xmlns="http://docbook.org/ns/docbook">
  <?xxe-sn c09ov3m4pt -1udd9r6h3v1fj?>

  <title><?xxe-sn c09ov3m4pt -1udd9r6h3v1fi?>Pangrams</title>

  <para><?xxe-sn c09ov3m4pt -1udd9r6h3v1fh?>"The quick brown fox jumps over
  the lazy dog"
  ...
</section>
```

The data contained in a `<?xxe-sn>` processing-instruction is a globally unique 128-bit ID formatted as follows: *`most_significant_64_bits_in_base36`* SPACE *`least_significant_64_bits_in_base36`*.

Note that enabling the comparison of revisions should not slowdown **XXE** perceptibly. That's why there is an option in *XMLmind XML Editor - Online Help* which allows to automatically enable the comparison of revisions in all the documents created and edited using **XXE**.

3. A simple XML differencing algorithm

The XML differencing algorithm implemented in the **Compare** tool in *XMLmind XML Editor - Online Help* may be described as follows:

1. Begin by comparing the root element of the original document to the root element of the revised document.

2. If the element in the original document (let's call it the original element) and the element in the revised document (let's call it the revised element) have the same serial number, then compare their contents. Otherwise consider that these elements are completely different.

3. Trivially compare the attributes of the original element to the attributes of the revised element.

4. The child nodes of an element are converted to a sequence of *comparable items* prior to be compared:
   - A text item is added to the sequence for each *word* contained in the element[1].
   - A serial number item is added for each child element contained in the element.
   - A comment item is added for each XML comment contained in the element.
   - A processing-instruction item is added for each processing-instruction contained in the element.
   - An inclusion directive item is added for each range of included nodes contained in the element.

   Example:

   ```
   <p>The <i>quick <b>brown</b></i> fox jumps over the <b>lazy</b> dog.</p>
   ```

   gives:

   ```
   "The ", element_7223, " fox", " jumps", " over", " the ", element_10087, " dog."
   ```

5. The sequence of items of the original element is compared to the sequence of items of the revised element using the well-known Unix **diff** algorithm[2] here applied to comparable items rather than to text lines:
   - Two text items are equal if they contain exactly the same text.
   - Two serial number items are equal if they have the same serial number.
   - Two comment items are equal if they contain exactly the same text.
   - Two processing-instructions items are equal if they have the same target and contain exactly the same text.
   - Two inclusion directive items are equal they have exactly the same XML contents. For example, `<xi:include href="vars.xml" xpointer="copyright"/>` and `<xi:include xpointer="copyright" href="vars.xml"/>` are equal, while `<xi:include href="vars.xml" xpointer="copyright"/>` and `<xi:include href="vars.xml" xpointer="notice"/>` differ.

6. Compare each child element of the original element to the child element element having the same serial number in the revised element. Proceed as explained starting from the Compare attributes [183] step.

Notes:

- The above algorithm is fast and 100% accurate by design.

- The comparison of attributes, comments and processing-instructions is not as fine-grained as the comparison of elements. For example, if attribute `class` is "`ui-widget`" in the original element and "`ui-widget ui-state-highlight`" in the revised element, the algorithm will tell you that attribute `class` has changed. It will not tell you that word "`ui-state-highlight`" has been added at the end of attribute `class`.

- Included contents (also called transcluded contents) found in the original element and in the revised element are never compared. Instead, the corresponding *inclusion directives* (`xi:include`, DITA `conref`, etc) implicitly[3] found in the original element and in the revised element are compared.

[1] If the element has or inherits `xml:space="preserve"`, a text item is added for each *text line* contained in the element.
[2] "*A file comparison program*" by Webb Miller and Eugene W. Myers, 1985.
[3] By default, inclusion directives are always transcluded by **XXE**. Hence such directives do not really exist in the document being edited. Instead, they are recreated by **XXE** each time the document is saved to disk.

Appendix B. Format of the revision history

Menu item **Tools** → **Revisions** → **Store All Revisions in the Document** in *XMLmind XML Editor - Online Help* (and also this option in *XMLmind XML Editor - Online Help*) allows to store the entire revision history of a document in the XML file containing this document.

The revision history is stored in a single `<?xxe-revisions>` processing-instruction found at the end of the document. Short example:

```
  ambiguë d&#8217;un c&#339;ur qui au zéphyr préfère les jattes de
  kiwi</foreignphrase>".</para>
</section>
<?xxe-revisions
#3 2015-09-04T14:35:48Z charles
#2 2015-09-04T14:29:20Z bart
1sPEAAABiXgAEocmAAAHBXOHExMIQwsAhRaEVw==

#1 2015-09-04T14:26:28Z john
1sPEAAABhyYAgQmHJQA/LBhmIjxmb3JlaWduOj0iZnJlPz5Qb3J0ZXogY2UgdmlldXggd2hpc2t5
IGF1IGp1Z2UgYmxvbmQgZnVtZTwvPiJzhAwTB0OBKlMx3zMGEwqTBxMEAQljBvczBAEEUyQBJzME
AQcjDgECkwpjCwCEPBREMoEthAI8g3QhFg+CIoETcJSCBw==
?>
```

The format of the `<?xxe-revisions>` processing-instruction is described as follows:

- This processing-instruction contains one or more *revision*s separated by newlines.
- A revision consists in a *header* followed by a *binary delta* encoded in base 64.
- A header, for example:

  ```
  #1 2015-09-04T14:26:28Z john
  ```

  contains the revision number (`1`), the date this revision was saved to disk (`2015-09-04T14:26:28Z`) and the author of this revision (`john`).

- First header is about the most recent revision. Hence it is not followed by a binary delta.

  For example, let's suppose the above snippet comes from document `doc.xml`. Most recent revision is #3 and corresponds to the contents of `doc.xml`. It has been created by `charles` on `2015-09-04T14:35:48Z`.

- The binary delta of revision #*N* is the result of applying the VCDIFF encoding algorithm between a normalized XML form of the content of revision #*N+1* and a normalized XML form of the content of revision #*N*.

  For example, the normalized XML form of `doc.xml` is simply the UTF-8 bytes of `doc.xml` saved unindented (of course, without the `<?xxe-revisions>` processing-instruction). The result of applying the VCDIFF decoding algorithm to the UTF-8 bytes of `doc.xml`, that is, revision #3, and the binary delta "`1sPEAAABiXgAEocmAAAHBXOHExMIQwsAhRaEVw==`" results in the UTF-8 bytes of revision #2 of `doc.xml`.