# XMLmind XML Editor - Configuration and Deployment

**Hussein Shafie**
**Pixware**

`<xmleditor-support@xmlmind.com>`

# XMLmind XML Editor - Configuration and Deployment

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

Publication date April 2, 2013

**Abstract**

This document describes how to customize and deploy XXE.

# Table of Contents

# List of Figures

# List of Tables

# List of Examples

# Part I. Guide

# Table of Contents

# Chapter 1. Introduction

XMLmind XML Editor (XXE for short) is an XML editor designed for production use. Unlike many other XML editors, its user interface does not allow to do simple things such as:

- Open an XML document in the editor and, after this, use a dialog box to associate a DTD and/or a style sheet to the newly opened document.
- Select a DTD or schema using a file chooser and then, use another dialog box to select the root element of a new document (conforming to the chosen DTD or schema).

The above features are useful if you muse with an XML file from time to time. They are almost never needed in production use, for example, writing a book ten hours a day.

Out of the box, XXE can be used to author XHTML, DocBook and DITA[1] documents with a good personal productivity.

But if you need to achieve *excellent* productivity for a group of users in your organization or if you need to use a proprietary DTD, W3C XML Schema or RELAX NG schema, then you'll have to customize existing XXE configurations or you'll have to write a custom configuration for your proprietary schema from scratch.

In an organization, the task of writing a configuration file for XXE is ideally performed by a single person, who belongs to the group of XML authors, but who is specially motivated by becoming the *local guru*.

- The local guru really needs to understand the job of the group of XML authors which will use XXE.
- The local guru really needs to be motivated because she/he will have to read tons of documentation: XXE documentation, but also many W3C standards such as XML, CSS, XPath, etc.
- The local guru does *not* need to be a programmer, or even a member of the IT staff.

If you don't have a person with the profile of a local guru, you may consider hiring an external consultant for a few days.

[1]Simplified DocBook, Slides, etc, are available as add-ons.

# Chapter 2. Writing a configuration file for XXE

## 1. What is a configuration?

A configuration file is a declarative XML file which teaches XXE how to handle documents of a given type. Without an appropriate configuration file, XXE is not of much use: the opened document is rendered using a tree view, it cannot be validated, schema-directed editing does not work, the user is limited to the most basic editing commands.

XXE is bundled with configurations for DITA, DocBook and XHTML. More configurations (e.g. Slides) are available but they need the user to download and install the corresponding add-on (i.e. using menu item Options → Install Add-ons).

For the impatient, the most basic configuration file looks like this:

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

  <detect>
    <dtdPublicId>-//XMLmind//DTD Example//EN</dtdPublicId>
  </detect>

  <template name="Example 1" location="example1.xml" />

</configuration>
```

## 2. A configuration for the "Simple Section" document type.

This chapter describes how to write a configuration for the "Simple Section" document type:

- The root element of this kind of document is a `section`. A `section` starts with a `title`. It may contain one or more `paragraph`s or `table`s. It may also contain nested `section`s. A `paragraph` contains text in addition to `bold`, `italic`, `literal`, `break` and `image` elements.
- The namespace used for the "Simple Section" document type is "`http://www.xmlmind.com/ns/sect`".

We'll describe 3 variants of the same configuration, one based on a DTD, one based on a W3C XML schema and one based on a RELAX NG schema.

These sample configurations are found in `*XXE_install_dir*/doc/configure/samples/`:

`dtd_section_config/`
: The DTD variant.

`xsd_section_config/`
: The W3C XML Schema variant.

`rng_section_config/`
: The RELAX NG variant.

**More sample configurations**

You'll also find in *XXE_install_dir*/doc/configure/samples/:

topic_plus_tag/
: A configuration for a DITA topic *specialization*. This specialization adds a tag element to the topic DTD. A tag element has a required kind attribute. The values allowed for the kind attribute are: attribute, attvalue, element, emptytag, endtag, genentity, localname, namespace, numcharref,paramentity, pi, prefix, comment, starttag.

  This configuration has been created by customizing the stock DITA topic configuration as explained in Chapter 5, *Customizing an existing configuration* [27].

## 3. Before writing your first configuration

1. Do not forget to temporarily disable the Quick Start and Schema caches by unchecking the corresponding checkboxes in Options → Preferences, Advanced|Cached Data section. More information about these caches in Section 5.12.1, “Cached data options” in *XMLmind XML Editor - Online Help*.

   **Warning**

   If you forget to do this, XXE will fail to see your configuration and/or may not see the changes you make to the schema referenced by your configuration.

2. Using menu item Options → Install Add-ons, download and install the add-on called "*XMLmind XML Editor Configuration Pack*". This add-on contains important support files (e.g. configuration.xsd) for use when writing a configuration.

3. Start XXE with a *console* in order to see low-level error messages possibly reported by XXE when you'll test your configuration.

   On Windows, this means running XXE using **java** rather than using **javaw**:

   - If you start XXE using xxe.exe (that is, the icon in the "XMLmind XML Editor" start menu), open xxe.jstart in **notepad**, replace "jre\bin\javaw.exe" by "jre\bin\java.exe", save xxe.jstart using the UTF-8 encoding (should have been automatically detected by **notepad**).

   - If you start XXE using xxe.bat, open xxe.bat in **notepad**, replace all occurrences of "javaw" by "java" and save xxe.bat.

4. If you are interested in having a functionality in your configuration which is present in any of the stock XHTML, DocBook or DITA configuration, do not hesitate to take a look at the .xxe, .incl, .css, etc, files found in:
   - *XXE_install_dir*/addon/config/xhtml/,
   - *XXE_install_dir*/addon/config/docbook/,
   - *XXE_install_dir*/addon/config/docbook5/,
   - *XXE_install_dir*/addon/config/dita/,
   - *XXE_install_dir*/addon/config/common/css/.

## 4. Anatomy of a configuration file

File dtd_section_config/section.xxe contains:

```
<configuration name="Simple Section (DTD)"
  xmlns="http://www.xmlmind.com/xmleditor/schema/configuration"
  xmlns:cfg="http://www.xmlmind.com/xmleditor/schema/configuration">

  <detect>
    <dtdPublicId>-//XMLmind//DTD Simple Section//EN</dtdPublicId>
  </detect>

  <template location="template.xml" name="Empty Section" />

  <include location="common.incl" />

</configuration>
```

- A configuration file must conform to the `configuration.xsd` W3C XML schema.
- A configuration file must have an "`xxe`" extension.
- The `configuration` root element must have a `name` attribute. This name must be chosen in order to uniquely identify the configuration among all the other configurations.
- The `configuration` element must have a `detect` child element (see below [7]).
- Support configuration files (such as `common.incl`; see above) may be included in the `.xxe` main file. However such support configuration files must *not* have "`xxe`" extensions and their `configuration` root elements must *not* have `name` attributes.
- The `.xxe` main configuration file along with *all* support files (`.incl`, `.css`, `.xsd`, `.rng`, `.dtd`, `*catalog.xml`, `.xsl`, icons, etc) must be created in a subdirectory itself contained in one of the two XXE `addon/` directories (more information about these `addon/` directories in Section 1, “Dynamic discovery of add-ons” [37]).

  For example, it could be created in the `addon/` subdirectory of XXE user preferences directory. XXE user preferences directory is:

  - `$HOME/.xxe5/` on Linux.
  - `$HOME/Library/Application Support/XMLmind/XMLEditor5/` on the Mac.
  - `%APPDATA%\XMLmind\XMLEditor5\` on Windows XP, Vista, 7 and 8.

    Example: `C:\Documents and Settings\john\Application Data\XMLmind\XMLEditor5\` on Windows XP. `C:\Users\john\AppData\Roaming\XMLmind\XMLEditor5\` on Windows Vista, 7 and 8.

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

The `configuration.xsd` schema is found in the add-on called "*XMLmind XML Editor Configuration Pack*". This means that you can quickly and safely your configuration file using XXE (using menu item File → New and then selecting entry "XMLmind XML Editor Configuration|Empty Template") or you can create your configuration file using any text editor and from time to time validate your configuration as follows:

```
C:\> XXE_install_dir\bin\xmltool validate¬
    -s xxe_config_pack_install_dir/config/xsd/configuration.xsd¬
    my_config.xxe my_include1.incl my_include2.incl
```

More information about the **xmltool** command line utility in The **xmltool** command-line utility.

## 5. Specifying which configuration to use for a given document type

The `detect` element [71] allows to specify which configuration to use for a given document type.

It works as follows: when XXE opens a document, it evaluates in turn the `detect` element found in each configuration. When all the conditions found in a `detect` element are met, XXE stops its evaluations and associates the configuration containing the matching `detect` element to the document being opened.

Excerpts from `rng_section_config/section.xxe` (configuration based on RELAX NG):

```
<detect>
  <rootElementNamespace>http://www.xmlmind.com/ns/sect</rootElementNamespace>
</detect>
```

The above `detect` element reads as: if the namespace of the root element of the file being opened is "`http://www.xmlmind.com/ns/sect`", then its configuration is `rng_section_config/section.xxe`.

The above `detect` element could be used as well for the configurations based on the DTD and the W3C XML Schema. However, in order to show you that there are often several ways to detect a document type, we have used a different `detect` condition in the configuration based on DTD:

```
<detect>
  <dtdPublicId>-//XMLmind//DTD Simple Section//EN</dtdPublicId>
</detect>
```

## 6. Associating a schema to the opened document

In the configuration based on the DTD, a document is assumed to always start with:

```
<!DOCTYPE section PUBLIC "-//XMLmind//DTD Simple Section//EN"
"http://www.xmlmind.com/dtd/section.dtd">
```

therefore the "`-//XMLmind//DTD Simple Section//EN`" DTD is automatically associated to the opened document. This DTD is then used to validate the document and also let XXE perform its schema-directed editing.

However there is no facility equivalent to `<!DOCTYPE>` for RELAX NG schemas, therefore this association must be specified in the configuration file. Excerpts from `rng_section_config/section.xxe` (configuration based on RELAX NG):

```
<relaxng location="section.rnc" />
```

The `relaxng` element [96] allows to associate a RELAX NG schema to the opened document. Similarly, the `schema` element [99] allows to associate a W3C XML Schema to the opened document and the `dtd` element [71] allows to associate a DTD to the opened document. Note that the `relaxng`, `schema` and `dtd` elements are completely ignored by XXE when the opened document contains a reference to its schema (i.e. by the means of `<!DOCTYPE>`, `xsi:schemaLocation`, `<?xml-model>`).

For example, let's suppose that, unlike what is shown in the `template.xml` file below [8], in the configuration based on the XML Schema, the `section` root element does *not* have an `xsi:schemaLocation` attribute (which, by the way, is somewhat cleaner), then specifying:

```
<schema>
  <location>http://www.xmlmind.com/ns/sect
            section.xsd</location>
</schema>
```

would have been mandatory.

File `xsd_section_config/template.xml` (configuration based on XML Schema):

```
<section xmlns="http://www.xmlmind.com/ns/sect"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.xmlmind.com/ns/sect
                             http://www.xmlmind.com/xsd/section.xsd">
  <title></title>
  <paragraph></paragraph>
</section>
```

## 7. XML catalogs

In the section above, notice that `dtd_section_config/template.xml` contains a reference to "`http://www.xmlmind.com/dtd/section.dtd`" and `xsd_section_config/template.xml` contains a reference to "`http://www.xm-`

lmind.com/xsd/section.xsd". Well, these files do not exist! Anyway, as explained in "*XML Entity and URI Resolvers*", even a real reference to a schema file would have ended up posing interchange problems.

Nevertheless, thanks to the XML catalogs found in the configuration directories, XXE has no problem loading the local copy of `section.dtd` and the local copy of `section.xsd`.

File `dtd_section_config/catalog.xml` (configuration based on DTD):

```
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
         prefer="public">
  <public publicId="-//XMLmind//DTD Simple Section//EN"
          uri="section.dtd"/>
</catalog>
```

The above catalog associates the public DTD ID "`-//XMLmind//DTD Simple Section//EN`" referenced in a document instance to local copy `section.dtd` (local because its URI is relative to `catalog.xml`).

File `xsd_section_config/catalog.xml` (configuration based on XML Schema):

```
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <uri name="http://www.xmlmind.com/xsd/section.xsd"
       uri="section.xsd"/>
</catalog>
```

The above catalog associates the absolute URI "`http://www.xmlmind.com/xsd/section.xsd`" referenced in a document instance to local copy `section.xsd` (local because its URI is relative to `catalog.xml`).

Note that, in the case of the configuration based on RELAX NG, because a document instance never directly references its schema, there is no need for an XML catalog.

For XXE to discover and load an XML catalog, the file containing it must have a name ending with string "`catalog.xml`". Examples: `catalog.xml`, `mycatalog.xml`, `foo_catalog.xml`.

## 8. Document templates

The `template` element [104] allows to specify the location and name of a document template. It is of course allowed to have several `template` elements in the same configuration. All these document templates are listed in the dialog box displayed by File → New.

Excerpts from `rng_section_config/section.xxe` (configuration based on RELAX NG):

```
<template location="template.xml" name="Empty Section" />
```

## 9. CSS stylesheets

The `css` element [70] allows to specify the location and name of a CSS stylesheet. It is of course allowed to have several `css` elements in the same configuration, provided that all `css` elements but one have an `alternate="true"` attribute. All these CSS stylesheets are listed at the end of the View menu. The CSS stylesheet which has an `alternate="false"` attribute is the one which, by default, is used to style the document being opened.

Excerpts from `rng_section_config/common.incl` (same file for all variants):

```
<css location="section.css" name="Normal" />
```

### 9.1. Styling images

The "Simple Section" document type has an `image` element. The source of the image is specified using required attribute `source`, which contains an absolute or relative URI. An `image` element also has optional `width` and `height` attributes, which contain dimensions expressed in pixels.

This `image` element is styled using replaced content. Excerpts from `rng_section_config/section.css` (same file for all variants):

```
image {
    display: inline;
    content: image-viewport(attribute, source,
                            data-type, anyURI,
                            content-width, attr(width),
                            content-height, attr(height));
}
```

This replaced content consists in extension pseudo-function `image-viewport()` in *XMLmind XML Editor - Support of Cascading Style Sheets (W3C CSS)* or extension pseudo-function `image()` in *XMLmind XML Editor - Support of Cascading Style Sheets (W3C CSS)*. Note that `image()` is based on `image-viewport()`, except that it has a simpler syntax. For example, one could have styled the `image` element as follows:

```
image {
    display: inline;
    content: image(attr(source), attr(width), attr(height));
}
```

## 9.2. Styling tables

The "Simple Section" document type has a rather simple `table` element. A `table` contains an optional `tableHeader` row, followed by one or more `tableRow` rows. A `tableHeader` or `tableRow` element contains one or more `tableCell` cells. A `tableCell` may span several columns, which is specified using attribute `columns`.

The `table` element is styled using standard CSS rules, expect for the number of columns spanned by a cell. Excerpts from `rng_section_config/section.css` (same file for all variants):

```
table {
    display: table;
    border: 1px solid gray;
    margin-top: 1.33ex;
    margin-bottom: 1.33ex;
}

tableHeader,
tableRow {
    display: table-row;
}

tableCell {
    display: table-cell;
    border: 1px solid gray;
    padding: 0.5ex;
}

...

tableCell[columns] {
    column-span: concatenate(attr(columns));
}
```

Extension property `column-span` in *XMLmind XML Editor - Support of Cascading Style Sheets (W3C CSS)* may be used to specify the number of columns spanned by an element having `display: table-cell`. Similarly, extension property `row-span` in *XMLmind XML Editor - Support of Cascading Style Sheets (W3C CSS)* may be used to specify the number of rows spanned by an element having `display: table-cell`.

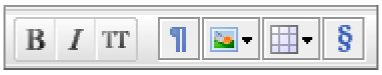
The value of property `column-span` (and `row-span`) is a positive integer. In the case of a `tableCell`, this value is obtained by parsing the value of attribute `columns` as a number. Expression `concatenate(attr(columns))` does exactly this:

1. Standard CSS pseudo-function `attr()` is used to return the value of attribute `columns`.
2. Extension pseudo-function `concatenate()` in *XMLmind XML Editor - Support of Cascading Style Sheets (W3C CSS)* concatenates all its string arguments, then parses the result of the concatenation and finally returns the parsed CSS property value.

## 10. A specific tool bar

The toolBar element [105] allows to specify a tool bar which is specific to a given document type.

**Figure 2.1. The "Simple Section" tool bar**



Excerpts from rng_section_config/common.incl (same file for all variants):

```
<toolBar>
  <button toolTip="Toggle bold"
          icon="xxe-config:common/icons/bold.png">
    <class>com.xmlmind.xmleditapp.kit.part.toolbar.TextStyleToggle</class>

    <command name="pass"
      parameter="{http://www.xmlmind.com/ns/sect}bold" />
  </button>

  ...
  <separator />
  ...
  <button toolTip="Add table" icon="xxe-config:common/icons/table_menu.png">
    <menu>
      <item label="table" command="add"
        parameter="after[implicitElement] {http://www.xmlmind.com/ns/sect}table" />
      <item label="table(header)" command="add"
        parameter="after[implicitElement]
                   #template({http://www.xmlmind.com/ns/sect}table,header)" />
    </menu>
  </button>
  ...
</toolBar>
```

A toolBar element has button and separator child elements. A button can contain a command or a menu of items, each menu item itself invoking a command.

### Tip

A button element is required to have an icon attribute. Notice in the above toolBar that all icon attributes have a value starting with "xxe-config:common/icons/".

a. "xxe-config:" is the only reliable way to refer *XXE_install_dir*/addon/config/.
b. *XXE_install_dir*/addon/config/common/icons/ contains an number of 16x16 icons commonly used for toolbars and menu items. If you are writing a configuration for XXE, you should really take a look at these icons.

So the question is now: what is a *command*? A command in *XMLmind XML Editor - Commands* is an action triggered by a user input (mouse, keyboard, drag, drop, etc) which has an effect on the active view of the document being edited. The behavior of a command is influenced by its string parameter, the current text or node selection and the schema to which the document being edited is conforming.

### Caution

The parameter of a command is a *plain string*, having vastly different meanings depending on the command. That's why, unless documented otherwise for a given command, namespace prefixes are not supported in a command parameter.

For example, in the above toolBar, you'll find qualified name s:table expressed as {http://www.xmlmind.com/ns/sect}table using Clark's notation. That is:

```
<command name="add"
  parameter="after[implicitElement] {http://www.xmlmind.com/ns/sect}table" />
```

and *not*:

```
<command name="add"
  parameter="after[implicitElement] s:table" />
```

We'll see in this tutorial that there are 3 kinds of commands: native commands (written in the Java™ programming language), *macro-commands* and *process commands* (both specified in XML in the configuration file).

XXE has dozens of native commands in *XMLmind XML Editor - Commands*. The tool bar which is specific to the "Simple Section" document type just uses one of them: `add` in *XMLmind XML Editor - Commands*.

### 10.1. Text style toggles

A tool bar normally contain plain buttons. For example, the following plain button invokes native command `convert` in *XMLmind XML Editor - Commands* to convert the implicit or explicit selection to the `bold` element:

```
<button toolTip="Convert to bold"
        icon="xxe-config:common/icons/boldText.png">
  <command name="convert"
    parameter="[implicitElement] {http://www.xmlmind.com/ns/sect}bold" />
</button>
```

However, we have chosen to add to the "Simple Section" tool bar *text style toggles* rather than plain buttons:

```
<button toolTip="Toggle bold"
        icon="xxe-config:common/icons/bold.png">
  <class>com.xmlmind.xmleditapp.kit.part.toolbar.TextStyleToggle</class>

  <command name="pass"
    parameter="{http://www.xmlmind.com/ns/sect}bold" />
</button>
```

Text style toggles emulate the behavior of the Bold, Italic, Underline, etc, toggles found in the tool bars of almost all word-processors. Such toggles are declared slightly differently than plain buttons. See Section 29.1.2, “The `TextStyleToggle` custom control” [110].

## 11. Element templates

You'll find in the command parameters of the above `toolBar`, in some cases, element qualified names such as `{http://www.xmlmind.com/ns/sect}table` and in other cases, references to *elements templates*, such as `#template({http://www.xmlmind.com/ns/sect}table,header)`.

The later notation means:

a. it's an element template;
b. this is a template for element `{http://www.xmlmind.com/ns/sect}table`;
c. the name of the element template is "`header`".

Note that element templates are uniquely identified by the combination element name/template name, and not by the template name alone.

Why specify element templates in your configuration file? By default, XXE creates the simplest possible, valid, element. For example, in the case of a table, this simplest possible, valid, `table` has just one `tableRow`, containing just one `tableCell`. In the case of a table, this is too simple to be really useful. On the contrary, we want a table to contain by default 2 rows, each row containing 2 cells. We also want to have a number of predefined table templates easily available, for example, a table having a header row. All this can be specified using the `elementTemplate` configuration [77] element.

Excerpts from `rng_section_config/common.incl` (same file for all variants):

```
<elementTemplate name="rows2" selectable="override">1
  <table xmlns="http://www.xmlmind.com/ns/sect">
    <tableRow>...ELIDED...</tableRow>
    <tableRow>...ELIDED...</tableRow>
  </table>
</elementTemplate>

<elementTemplate name="header">2
  <table xmlns="http://www.xmlmind.com/ns/sect">
    <tableHeader>...ELIDED...</tableHeader>
    <tableRow>...ELIDED...</tableRow>
    <tableRow>...ELIDED...</tableRow>
  </table>
</elementTemplate>

<elementTemplate name="image" selectable="false">3
  <paragraph xmlns="http://www.xmlmind.com/ns/sect"><image source="???"/></paragraph>
</elementTemplate>
```

1 Attribute `selectable="override"` means: use this template instead of the 1 row/1cell table generated by default by XXE. This also implies that this element template will be listed simply as "`table`", and not as "`table(rows2)`", in the Edit tool in *XMLmind XML Editor - Online Help*.

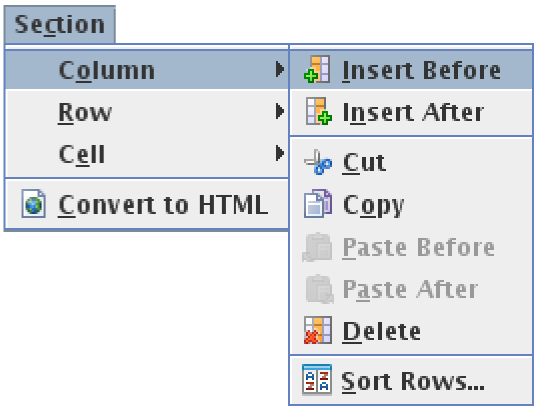
2 The default value of `attribute` selectable is `true`. This makes the corresponding element template available for the Edit tool. In the above case, this element template will be listed as "`table(header)`".

3 Attribute `selectable="false"` means: do not make this element template available for the Edit tool, as it is just needed here in this configuration file (i.e. in a `menu`, `toolBar`, `macro`, etc).

## 12. A specific menu

The `menu` element [90] allows to specify a menu which is specific to a given document type.

**Figure 2.2. The "Simple Section" menu**



Excerpts from `rng_section_config/common.incl` (same file for all variants):

```
<menu label="Se_ction">
  <menu label="C_olumn" name="tableColumnMenu">
    <item label="_Insert Before"
          icon="xxe-config:common/icons/insertColumnBefore.png"
          command="sect.tableEdit" parameter="insertColumnBefore"/>
    <item label="I_nsert After"
          icon="xxe-config:common/icons/insertColumnAfter.png"
          command="sect.tableEdit" parameter="insertColumnAfter"/>
  ...
  </menu>
  <separator />
  <item label="_Convert to HTML"
        icon="xxe-config:common/mime_types/html.png"
        command="sect.convertToHTML" />
</menu>
```

A `menu` element has `item`, `menu` and `separator` child elements, each menu `item` invoking a command. A `menu` element is required to have a `label` attribute. In the value of the `label` attribute, character underscore (`'_'`) may be used to specify the mnemonic of a menu or an item.

## 12.1. About the table editing command

### What if you have reused standard HTML or CALS tables in your own custom schema?

What to do in this case is explained in next chapter Chapter 3, *Using HTML4 tables or CALS tables in your own custom schema* [21].

The above `menu` element contains 3 sub-menus called Column, Row and Cell. All the items of these sub-menus invoke the same `sect.tableEdit` table editing command, albeit with different parameters, each parameter specifying the desired operation (e.g. "`insertColumnBefore`").

XXE has a native, generic, parameterizable, table editing command in *XMLmind XML Editor - Commands* powerful enough to edit all kinds of custom tables. However in order to be able to use this command in your configuration, you must

1. instantiate this command and give a name to your instance by the means of the `command` configuration element [68];
2. parameterize your instance by the means of a `property` configuration element [94].

This is done as follows. Excerpts from `rng_section_config/common.incl` (same file for all variants):

```
<command name="sect.tableEdit">
  <class>com.xmlmind.xmledit.cmd.table.GenericTableEdit</class>
</command>

<property name="sect.tableEdit.tableSpecification">
  table={http://www.xmlmind.com/ns/sect}table
  row={http://www.xmlmind.com/ns/sect}tableRow {http://www.xmlmind.com/ns/sect}tableHeader:header
  cell={http://www.xmlmind.com/ns/sect}tableCell
  columnSpan=columns
</property>
```

If the name of your table editing command is *`foo`*, then the name of the corresponding property must be *`foo`*`.tableSpecification`. The content of the property is simply the description of the element and attribute names used by your custom table.

### Caution

The above `command` element defines a command called `sect.tableEdit`. Why this "`sect.`" prefix? Why not simply "`tableEdit`"?

The commands defined in a configuration are not local to this configuration. *All commands have a global scope*. If you call your command `tableEdit`, then there are chances that you'll overwrite another command, defined in another configuration, also called `tableEdit` (or the other way round, depending on the order which is used by XXE to load the configurations).

Therefore you must always give a prefix which unique to your configuration to the names of the commands defined in this configuration. For example, the stock configurations use these prefixes: "`xhtml.`", "`docb.`", "`db5.`", "`dita.`", "`ditamap.`".

## 13. Specific keyboard shortcuts

The `binding` element [61] allows to specify a *binding* which is specific to a given document type. A `binding` element binds a user input, for example a sequence of key strokes, to a command.

Excerpts from `rng_section_config/common.incl` (same file for all variants):

```
<property name="$c blockList">
  {http://www.xmlmind.com/ns/sect}paragraph=paragraph
</property>

<binding>
  <keyPressed code="ENTER" />1
  <command name="insertNewlineOrSplitBlock" />
</binding>

...

<binding>
  <keyPressed code="DELETE" />
  <command name="deleteSelectionOrJoinBlockOrDeleteChar" />
</binding>

<binding>
  <keyPressed code="BACK_SPACE" />
  <command name="deleteSelectionOrJoinBlockOrDeleteChar"
           parameter="backwards" />
</binding>

...

<binding>
  <keyPressed code="ENTER" modifiers="shift" />2
  <command name="sect.insertBreak" />
</binding>
```

1 When a user presses the ENTER, DELETE, or BACK_SPACE key while the caret is found inside a `paragraph` element, we want XXE to behave like any word-processor. This is precisely what do commands `insertNewlineOrSplitBlock in` *`XMLmind XML Editor - Commands`* and `deleteSelectionOrJoinBlockOrDeleteChar` in *XMLmind XML Editor - Commands*. However for these commands to work, the elements behaving like paragraphs (or list items) must have been specified. For the "Simple Section" configuration, this specification is simply:

```
<property name="$c blockList">
  {http://www.xmlmind.com/ns/sect}paragraph=paragraph
</property>
```

Notice "`$c blockList`", which is a shorthand for "*`configuration_name`* `blockList`".

2 When the user presses Shift+ENTER, we want XXE to add a `break` element at caret location. Moreover, if the newly inserted `break` element is not immediately followed by a text node, we want XXE to automatically add a text node and move the caret to this new empty text node.

## 13.1. About macro commands

In the excerpts from `rng_section_config/common.incl` below, `sect.insertBreak` is a macro-command. A macro-command in *XMLmind XML Editor - Commands* is a command specified in XML as combination of other commands (of any kind: native command, macro-command or process command).

Let's examine `sect.insertBreak`:

```
<command name="sect.insertBreak">
  <macro undoable="true">
    <sequence>1
      <command name="insert"
        parameter="into {http://www.xmlmind.com/ns/sect}break" />
      <command name="insertNode" parameter="textAfter" />2
      <command name="cancelSelection" />3
    </sequence>
  </macro>
</command>
```

1 The above macro uses the simplest and most common form of combination of other commands: the sequence. It invokes in turn the following native commands: `insert` in *XMLmind XML Editor - Commands*, `insertNode` in *XMLmind XML Editor - Commands* and `cancelSelection` in *XMLmind XML Editor - Commands*.

2 The newly inserted `break` element is automatically selected at the end of the execution of command `insert`. If there is already a text node after the selected `break` element, then command `insertNode` silently fails and the sequence of commands is stopped at this point. This kind of failure is harmless and we can even say that, as the developers of the above macro, we count on this behavior.

3 The above macro has two “nice touches”:

a. Command `cancelSelection` is invoked to get rid of the red box around the newly inserted `break` element. Not strictly needed but nice to have.
b. The macro has attribute `undoable="true"`. Without this attribute, after `sect.insertBreak` has been executed in full, the user would have to invoke Edit → Undo (**Ctrl**+**Z**) twice: one time to remove the text node and a second time to remove the `break` element.

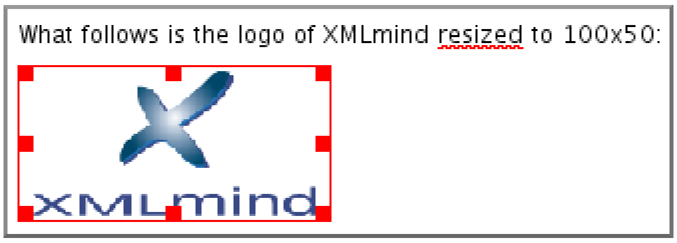
## 14. Interactively resizing an image

In the above section, we have explained how to bind a keystroke: Shift+ENTER to a command: `sect.insertBreak`. Similarly, interactively resizing an image may be implemented by binding an *application event*: `resize-image` [66] to a command: `resizeImage`.

```
<binding>
  <appEvent name="resize-image" />
  <command name="resizeImage"
           parameter="height=%{height} width=%{width}" />
</binding>
```

`Resize-image` events are emitted when the user first clicks inside an image in order to display handles around it and then drag one of those handles.

### Figure 2.3. Resize handles around an image


What follows is the logo of XMLmind resized to 100x50:

In the case of the "Simple Section" configuration, dragging a resize handle invokes command `resizeImage` in *XMLmind XML Editor - Commands*. This command resizes an image by modifying its `width` and `height` attributes.

It's important to understand that the resize image command being invoked must correspond to the image element. If the image element had no `width` and `height` attributes or if these attributes contained inches rather than pixels, built-in command `resizeImage` would not be usable here.

For example, it's possible to interactively resize a column of a table by binding application event `resize-table-column` [67] to the proper command. However the `table` element of the "Simple Section" document type has no child element or attribute allowing to specify the width of a column. Therefore, in the case of the "Simple Section" configuration, there is no way to specify how a user can resize a table column by dragging the corresponding column separator.

## 15. Miscellaneous configuration elements

The `spellCheckOptions` element [102] allows to specify spell-checker options, for example whether the on-the-fly spell-checking should be turned on by default, which elements should not be checked for spelling, etc. Excerpts from `rng_section_config/common.incl` (same file for all variants):

```
<spellCheckOptions useAutomaticSpellChecker="true"
                   languageAttribute="xml:lang"
                   skippedElements="s:literal" />
```

The `documentResources` element [74] allows to specify how to determine the resource files attached to the document being edited. In the case of a "Simple Section" document, the only resource files are the image files pointed to by the `image` elements. Excerpts from `rng_section_config/common.incl` (same file for all variants):

```
<documentResources>
  <resource path="//s:image/@source" />
</documentResources>
```

Knowing which resource files are attached to the document being edited is needed to implement File → Save As and also the conversion of the document being edited to other formats (see below [18]).

## 16. Converting an XML document to other formats

For the purpose of this tutorial, we'll explain how to convert the "Simple Section" document being edited to a single HTML page.

A XSLT stylesheet will be used for this task. Excerpts from `rng_section_config/html.xsl` (same file for all variants):

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:s="http://www.xmlmind.com/ns/sect"
                exclude-result-prefixes="s"
                version="1.0">
  <xsl:output method="html"
              encoding="ISO-8859-1"
              indent="no"/>
  ...
  <xsl:template match="s:bold">
    <b>
      <xsl:call-template name="processCommonAttributes"/>
      <xsl:apply-templates/>
    </b>
  </xsl:template>
  ...
</xsl:stylesheet>
```

It should be noted that XXE has no high-level construct (e.g. an hypothetical `convertDocument` configuration element) allowing to integrate an XSLT stylesheet. Instead, XXE relies on its `menu`, `toolBar`, `binding`, `macro` command, `process` command, etc, to do that.

For example, our "Section" `menu` element ends with:

```
  ...
  <separator />
  <item label="_Convert to HTML"
        icon="xxe-config:common/mime_types/html.png"
        command="sect.convertToHTML" />
</menu>
```

Macro-command `sect.convertToHTML` allows the user to choose a save file for the HTML page which will be the result of the conversion (by the means of native command `selectConvertedFile` in *XMLmind XML Editor - Commands*) and then invokes process command `sect.toHTML` which will perform the conversion. Excerpts from `rng_section_config/common.incl` (same file for all variants):

```
<command name="sect.convertToHTML">
  <macro>
    <sequence>
      <command name="selectConvertedFile"
               parameter="saveFileURLWithExtension=html" />
      <command name="sect.toHTML" parameter='"%_"' />1
      <command name="preview" parameter="[lastConverted]" />2
    </sequence>
  </macro>
</command>
```

In the above macro,

1 The command parameter contains `"%_"`. `%_` is a *macro-variable* in *XMLmind XML Editor - Commands* and it is quoted using double-quote characters in case it contains whitespace. Variable `%_` contains the result returned by the last executed command. In the above case, the last executed command is `selectConvertedFile` and the result it returns is the URI of the save file selected by the user.

2 Command `preview` in *XMLmind XML Editor - Commands* allows to invoke a third-party helper application (e.g. a Web browser) in order to preview the result of the conversion.

## 16.1. About process commands

Like macro-commands, a *process command* in *XMLmind XML Editor - Commands* is a command specified in XML. But a macro-command acts on elements, attributes and on the text or node selection, while a process command *acts on files*.

Moreover the current working directory (that is, directory "`.`") of a process command is always a temporary directory especially created for the execution of the process command. This means that all files and directories created in this temporary directory will go away once the execution of the process command is complete.

Excerpts from `rng_section_config/common.incl` (same file for all variants):

```
<command name="sect.toHTML">
  <process>
    <mkdir dir="images" />1

    <copyDocument to="__doc.xml">2
      <resources match="^[a-zA-Z][a-zA-Z0-9.+-]*:/.+" />3

      <resources match=".+\.(png|jpg|jpeg|gif)" copyTo="images" />4
    </copyDocument>

    <transform stylesheet="html.xsl" file="__doc.xml" to="__doc.html" />5

    <upload base="%0">6 7
      <copyFile file="__doc.html" to="%0" />
      <copyFiles files="images/*" toDir="images" />
    </upload>
  </process>
</command>
```

1 A `process` element has a number of child elements which are verbs acting on files and directories. Among these verbs, you'll find `mkdir` in *XMLmind XML Editor - Commands*.

Here we use `mkdir` to create an `images/` directory. The `images/` directory will contain a copy of some of the image files referenced in the "Simple Section" document being converted.

2 Child element `copyDocument` in *XMLmind XML Editor - Commands* is used to copy the document being converted to `__doc.xml`. Note that in the case the document being converted contains inclusions (XInclude, DITA `conref`, etc), these inclusions are fully transcluded in order to ease the job of the XSLT engine.

3 This `resources` element in *XMLmind XML Editor - Commands* specifies that resource files referenced in the document being converted by the means of absolute URIs (e.g. `<image source="http://www.acm.com/logo.gif"/>`) should be ignored.

4 This other `resources` element specifies that resource files referenced in the document being converted by the means of relative URIs ending with "`.png`", "`.jpg`", etc, (e.g. `<image source="photos/photo43.jpg"/>`) should be copied to the `images/` directory.

Of course, this implies a fixup of the corresponding reference in `__doc.xml` (e.g. `<image source="photos/photo43.jpg"/>` becomes `<image source="images/photo43.jpg"/>`)

5 Child element `transform` in *XMLmind XML Editor - Commands* invokes the XSLT engine (Saxon 6.5 or Saxon 9 depending on the version of the XSLT stylesheet) in order to transform `__doc.xml` to `__doc.html` using stylesheet `html.xsl`.

6 Child element `upload` in *XMLmind XML Editor - Commands* is used to upload file `__doc.html` and directory `images/` to the location specified by `%0`.

By upload, we really mean *upload* and not simply copy these files somewhere else on the local file system. For example, if you install the add-on called "*FTP virtual drive plug-in*", you'll be able to select an "`ftp:`" (or "`ftps:`" or "`sftp:`") URI using `selectConvertedFile` and then upload file `__doc.html` and directory `images/` to the FTP server you have chosen.

7 What is `%0`?

All commands may be passed a parameter. This parameter may be considered to contain a number of *arguments* separated with whitespace (an argument itself containing whitespace should be quoted using single or double quotes). Variable `%0` specifies the first argument, `%2` the second one, etc, up to `%9`. For example, if a macro or process command is passed parameter "`one 'two three' four`", then `%0` contains "`one`", `%1` contains "`two three`" and `%2` contains "`four`".

Process command `sect.toHTML` is passed a parameter containing the quoted URI of the save file, hence `%0` contains the URI of the save file.

## 17. Packaging your configuration for XXE add-on manager

Packaging your configuration for XXE add-on manager is optional[1]. However, if you deploy XXE as a desktop application, this packaging allows your users to easily install, upgrade or uninstall your configuration using menu item Options → Install Add-ons.

An add-on is simply a Zip archive ("`zip`" file extension) where all the files and directories comprising the add-on are contained in a single topmost directory. Among these files, XXE must find a single file having a "`xxe_addon`" extension. For example, let's suppose the configuration based on the XML schema is found in `xsd_section_config.zip`. Unzipping this file would give:

```
C:\> unzip xsd_section_config.zip
   creating: xsd_section_config/
  inflating: xsd_section_config/common.incl
  inflating: xsd_section_config/section.xsd
  ...
  inflating: xsd_section_config/xsd_section_config.xxe_addon
  ...
  inflating: xsd_section_config/catalog.xml
```

File `xsd_section_config.xxe_addon` contains:

```
<a:addon xmlns="http://www.w3.org/1999/xhtml"
         xmlns:html="http://www.w3.org/1999/xhtml"
         xmlns:a="http://www.xmlmind.com/xmleditor/schema/addon">
  <a:category>
    <a:configuration />
  </a:category>

  <a:name>"Simple Section (XML Schema)" Configuration</a:name>

  <a:version>1.0.0</a:version>

  <a:abstract>Part of the tutorial explaining how to write a configuration for
  XMLmind XML Editor.</a:abstract>
</a:addon>
```

The easiest way to create an `.xxe_addon` file is to use XXE:

[1]This is even useless if you deploy XXE as an applet or using Java™ Web Start.

1. Download and install the add-on called "*XMLmind XML Editor Configuration Pack*" using Options → Install Add-ons.
2. Restart XXE as instructed.
3. Select File → New and choose XMLmind XML Editor Add-on|Single Add-on.

Make sure to carefully choose a unique, descriptive, *stable* name for your add-on. Changing the name of an add-on from one release to another is likely to annoy your users. For example, this breaks the upgrade facility offered by XXE add-on manager.

More information in "*XMLmind XML Editor - Developer's Guide*", "*Packaging an add-on for XMLmind XML Editor integrated add-on manager*".

# Chapter 3. Using HTML4 tables or CALS tables in your own custom schema

If you create a custom schema and need general purpose tables for it, you'll probably choose the well-known HTML4 or CALS[1] tables.

### Tip

If this is not the case and if you have created your own table model, then you can still use the generic, parameterizable, table editor documented in Section 117, “A generic, parameterizable, table editor command” in *XMLmind XML Editor - Commands*. Note that, for this generic table editor to work with your table model, your table model needs to vaguely resemble the HTML table model (table contains rows, themselves possibly contained in row groups, etc).

Including the definition of table elements in your custom schema will not be described in this chapter. Instead this chapter will explain:

- how to properly render HTML4 or CALS tables on screen by using a CSS style sheet;
- how to include table editing commands in your custom configuration for XXE.

### Important

All the CSS style sheets and all the commands described below have been designed to properly work whatever is the namespace you have chosen for your schema and/or for the table elements.

## 1. HTML4 tables

The corresponding support code is contained in `XXE_install_dir/addon/config/common/xhtml.jar`.

### Procedure 3.1. Procedure

1. Add this snippet at the top of your CSS style sheet:

   ```
   @import url(xxe-config:common/css/xhtml_table.imp);
   ```

   If you use a namespace (e.g. `http://acme.com/ns`) for all the elements defined in your schema, including for table elements, add this snippet instead. This is not strictly needed but this will speed up the rendering of XML elements on screen:

   ```
   @namespace "http://acme.com/ns";
   @import url(xxe-config:common/css/xhtml_table.imp);
   ```

2. Add this snippet in your custom configuration for XXE. In the example below, you have chosen to prefix all the custom commands declared in your configuration using prefix "`my.`".

   ```
   <command name="my.tableEdit">
     <class>com.xmlmind.xmleditext.xhtml.table.HTMLTableEdit</class>
   </command>
   ```

   After that, you can reference the above table commands in your custom menu, custom tool bar or custom bindings. Example:

   ```
   <menu label="M_yDoc">
     <item label="Insert Column _Before"
           icon="xxe-config:common/icons/insertColumnBefore.png"
   ```

[1]That is, DocBook tables up to V4.2. DocBook V4.3+ supports both HTML4 and CALS tables.

```
        command="my.tableEdit" parameter="insertColumnBefore"/>
  ...
```

3. Add this other snippet to your custom configuration. Doing so will allow you to resize a table column by dragging its column separator [67].

```
<binding>
  <appEvent name="resize-table-column" />
  <command name="my.resizeTableColumn"
           parameter="%{resizedColumn} %{columnCount}
                      %{oldColumnWidths} %{newColumnWidths}" />
</binding>

<command name="my.resizeTableColumn">
  <class>com.xmlmind.xmleditext.xhtml.table.ResizeTableColumn</class>
</command>
```

## 1.1. HTML4 table editor command

| **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. |
| | `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.) |
| A row must be explicitly selected.<br><br>OR a cell or an element having a cell ancestor must be implicitly or explicitly selected. | `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. |
| | `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. |

## 2. CALS tables

The corresponding support code is contained in `XXE_install_dir/addon/config/common/docbook.jar`.

### Procedure 3.2. Procedure

1. Add this snippet at the top of your CSS style sheet:

   ```
   @import url(xxe-config:common/css/cals_table.imp);
   ```

   If you use a namespace (e.g. `http://acme.com/ns`) for all the elements defined in your schema, including for table elements, add this snippet instead. This is not strictly needed but this will speed up the rendering of XML elements on screen:

   ```
   @namespace "http://acme.com/ns";
   @import url(xxe-config:common/css/cals_table.imp);
   ```

2. Add this snippet in your custom configuration for XXE. In the example below, you have chosen to prefix all the custom commands declared in your configuration using prefix "`my.`".

   ```
   <command name="my.tableEdit">
     <class>com.xmlmind.xmleditext.docbook.table.CALSTableEdit</class>
   </command>
   ```

   After that, you can reference the above table commands in your custom menu, custom tool bar or custom bindings. Example:

   ```
   <menu label="M_yDoc">
     <item label="Insert Column _Before"
           icon="xxe-config:common/icons/insertColumnBefore.png"
           command="my.tableEdit" parameter="insertColumnBefore"/>
     ...
   ```

3. Add this other snippet to your custom configuration. Doing so will allow you to resize a table column by dragging its column separator [67].

   ```
   <binding>
     <appEvent name="resize-table-column" />
     <command name="my.resizeTableColumn"
              parameter="%{resizedColumn} %{columnCount}
                         %{oldColumnWidths} %{newColumnWidths}" />
   </binding>

   <command name="my.resizeTableColumn">
     <class>com.xmlmind.xmleditext.docbook.table.ResizeTableColumn</class>
   </command>
   ```

4. File `docbook.jar` also contains a *validation hook* which ensures that the `cols` attribute of elements `tgroup` and `entrytbl` is always set to a correct value before a DocBook document is validated and hence, saved to disk.

   Using commands `tableColumn` and `tableRow` also ensures that the `cols` attribute is up to date. However it is strongly recommended to add this validation hook to your custom configuration. This is done by adding this snippet:

   ```
   <validateHook name="cols_checker">
     <class>com.xmlmind.xmleditext.docbook.table.ValidateHookImpl</class>
   </validateHook>
   ```

## 2.1. CALS table editor command

The parameters supported by this table editor command are identical to those of the HTML4 table editor command [22].

# Chapter 4. Customizing mouse and key bindings used by XXE

## 1. Bindings specific to a document type

A configuration file such as `XXE_install_dir/addon/config/docbook/docbook.xxe` can contain binding [61] elements. A binding element specifies:

- a keystroke or a sequence of keystrokes which triggers a command,
- OR a mouse input which triggers a command or displays a custom popup menu.

For example, adding the following binding element to `docbook.xxe` will allow to convert selected text to emphasis (with role attribute set to bold) by pressing on function key **F5**:

```
<binding>
  <keyPressed code="F5" />
  <command name="docb.convertToBold" />
</binding>

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

It is recommended to add custom bindings into a separate file and to include this file in configurations files bundled with XXE rather than directly modifying the bundled configuration files.

For example, if you want to use the **F5** key for converting text to emphasis in all documents belonging to the DocBook family (DocBook, Simplified DocBook, Slides), add the elements of the previous example to a file called `/opt/xxe-custom/extrabindings.incl` and include this file in `XXE_install_dir/addon/config/docbook/docbook.xxe`.

```
<include location="file:///opt/xxe-custom/extrabindings.incl" />
```

In next chapter [27], we will learn how to that without modifying the bundled configuration files.

### Important

XXE *does not allow bindings defined in document type specific configuration files to override its menu accelerators.*

Example 1: you cannot bind **Ctrl**+**Q** to command `docb.convertToBold` because **Ctrl**+**Q** is used to quit XXE.

Example 2: you cannot bind **Ctrl**+**I** to command `docb.convertToBold` because, by default, **Ctrl**+**I** triggers command "`insert`" with parameter "`into`" (menu item Edit → Insert).

## 2. Generic bindings

What if you want add bindings which are not specific to a document type. Do you really have to include them in all configuration files?

What if you really *hate* some of the default bindings of XXE? Do you really have to stop using XXE?

The answer is no to both questions. Simply add your generic bindings to a file called `customize.xxe` anywhere XXE can find it. For example, create this file in the `addon/` subdirectory of your user preferences directory. XXE user preferences directory is:

- *`$HOME`*`/.xxe5/` on Linux.
- *`$HOME`*`/Library/Application Support/XMLmind/XMLEditor5/` on the Mac.
- *`%APPDATA%`*`\XMLmind\XMLEditor5\` on Windows XP, Vista, 7 and 8.

  Example: `C:\Documents and Settings\john\Application Data\XMLmind\XMLEditor5\` on Windows XP. `C:\Users\john\AppData\Roaming\XMLmind\XMLEditor5\` on Windows Vista, 7, and 8.

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

For more information about how XXE finds its configuration files, please read Section 1, “Dynamic discovery of add-ons” [37].

If several configuration files called `customize.xxe` are found, their contents are merged with a higher priority to `customize.xxe` files found in the user preferences directory.

File `customize.xxe` may also be used to specify parameterGroup [95], imageToolkit [80], spreadsheetFunctions [103], property [94].

A *very useful*[1] sample `customize.xxe` may be downloaded and installed using XXE add-on manager (Options → Install Add-ons). Excerpt of this sample `customize.xxe`:

```
                   . . .
<binding>
  <keyPressed code="ESCAPE" />
  <charTyped char="l" />
  <command name="convertCase" parameter="lower" />
</binding>

<binding>
  <keyPressed code="ESCAPE" />
  <charTyped char="u" />
  <command name="convertCase" parameter="upper" />
</binding>

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

<binding>
  <keyPressed code="ESCAPE" />
  <charTyped char="!" />
  <command name="insertCommandOutput" />
</binding>
                   . . .
```

### Important

Defining a binding in `customize.xxe` prevents XXE from using the same keystroke as a menu accelerator. For example, if you bind a command such as "`recordMacro toggle`" to **Ctrl**+**O**, then menu item File → Open will loose its customary shortcut.

[1]Yours truly cannot use XXE without it.

# Chapter 5. Customizing an existing configuration

This chapter is not a tutorial. It will merely give you some recipes. If you want to understand what you are doing, please refer to Writing a configuration file for XXE [5].

Let's suppose you want to customize one of the DITA[1], DocBook 5, DocBook 4 or XHTML configurations, here's what to do.

1. Create in *XXE_user_preferences_dir*/addon/[2] a subdirectory which will contain all the files comprising your customization.

   The name of this directory is not important. Let's suppose you have created *XXE_user_preferences_dir*/addon/custom/.

2. Copy one of the following template files depending on which configuration you want to customize:

   | Configuration Name | Procedure |
   |---|---|
   | DITA | Copy 0topic.xxe to custom/. |
   | DITA Map | Copy 0map.xxe to custom/. |
   | DITA BookMap | Copy 0bookmap.xxe to custom/. |
   | DocBook v5+ | Copy 0docbook5.xxe to custom/. |
   | DocBook | Copy 0docbook.xxe to custom/. |
   | XHTML Strict | Copy 0xhtml_strict.xxe to custom/. |
   | XHTML Transitional | Copy 0xhtml_loose.xxe to custom/. |
   | XHTML 1.1 | Copy 0xhtml11.xxe to custom/. |
   | XHTML 5 | Copy 0xhtml5.xxe to custom/. |

   For example, 0docbook5.xxe[3] looks like this:

```
<configuration [69] name="DocBook v5+"
  xmlns="http://www.xmlmind.com/xmleditor/schema/configuration"
  xmlns:cfg="http://www.xmlmind.com/xmleditor/schema/configuration"
  xmlns:db="http://docbook.org/ns/docbook"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:xi="http://www.w3.org/2001/XInclude"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:html="http://www.w3.org/1999/xhtml">

  <include [83] location="docbook5-config:docbook5.xxe" />

</configuration>
```

3. This step involves adding one or more configuration elements after the include element. This is done using any XML or text editor. Each of the following sections describes a common customization step.

4. Restart XXE.

[1]The configuration associated to DITA topics is called DITA. The configuration associated to DITA maps is called DITA Map. The configuration associated to DITA bookmaps is called DITA BookMap.
[2]*XXE_user_preferences_dir* is documented in 1 [37].
[3]The ``funny" name, starting with a "0", has its utility if you happen to create your customization in *XXE_install_dir*/addon/ rather than in *XXE_user_preferences_dir*/addon/.

## 1. Adding a custom document template

If you want to add a new document template which would be listed in the dialog box displayed by File → New:

1. Create this document template, preferably using XMLmind XML Editor. At least, make sure that the created file is valid by opening it in XXE.
2. Copy this file to `custom/`.
3. Let's suppose this file is called `template1.xml` and that you want your document template to be listed in the dialog box as "`Template #1`".

   Add the following template [104] configuration element to your custom `.xxe` file (e.g. `0docbook5.xxe`):

   ```
   <template [104] name="Template #1"
             location="template1.xml" />
   ```

## 2. Replacing an existing document template

Add the following template [104] configuration element to your custom `.xxe` file:

```
<template [104] name="Name of the template to be replaced"
          location="template1.xml" />
```

Specify the *English* name [28] of the template as listed by the File → New dialog box. XHTML example: "`Page`" (not "`Seite`").

**How to see the English names of configuration elements?**

When you want to remove or replace an existing configuration element, you need to refer to it by its English name. You cannot refer to it by its localized name.

Now, how to learn what is the English name of a given configuration element? The obvious solution is to look in the bundled configuration files:

| Configuration Name | Bundled Configuration Files |
|---|---|
| DITA<br><br>DITA Map<br><br>DITA BookMap | `XXE_install_dir/addon/config/dita/*.xxe`, `*.incl`. |
| DocBook v5+ | `XXE_install_dir/addon/config/docbook5/*.xxe`, `*.incl`. |
| DocBook | `XXE_install_dir/addon/config/docbook/*.xxe`, `*.incl`. |
| XHTML Strict<br><br>XHTML Transitional<br><br>XHTML 1.1<br><br>XHTML 5 | `XXE_install_dir/addon/config/xhtml/*.xxe`, `*.incl`. |

Given the fact that the names of configuration elements are often displayed by the GUI of XXE (the name of document templates are listed in the File → New dialog box, the names of CSS style sheets are listed in the View menu, etc), a simpler solution consists in temporarily switching to the English locale. In order to do this, use Options → Preferences, General section, Locale combobox. More information in Locale in *XMLmind XML Editor - Online Help*.

## 3. Removing an existing document template

Add the following template [104] configuration element to your custom `.xxe` file:

```
<template [104] name="Name of the template to be removed" />
```

Specify the *English* name [28] of the template as listed by the File → New dialog box. XHTML example: "`Page`" (not "`Seite`").

## 4. Adding a custom CSS style sheet

Procedure:

1. Copy one of the following files depending on which configuration you want to customize:

| Configuration Name | Procedure |
|---|---|
| DITA | Copy topic.css to `custom/`. |
| DITA Map | Copy map.css to `custom/`. |
| DITA BookMap | Copy bookmap.css to `custom/`. |
| DocBook v5+ | Copy docbook5.css to `custom/`. |
| DocBook | Copy docbook.css to `custom/`. |
| XHTML Strict<br><br>XHTML Transitional<br><br>XHTML 1.1<br><br>XHTML 5 | Copy xhtml.css to `custom/`. |

   For example, `xhtml.css` looks like this:

```
@import url(xhtml-config:css/xhtml.css);
```

2. Edit this file using a text editor and add one or more CSS rules after the `@import` directive.

   XHTML example:

```
p {
    color: red;
}
```

   DocBook, DocBook v5+ example:

```
para {
    color: red;
}
```

3. Check your CSS file using the **csscheck** command-line utility. This utility is found in *`XXE_install_dir`*`/bin/`. Example:

```
$ /opt/xxe/bin/csscheck stylesheet1.css
```

4. Let's suppose this file is called `stylesheet1.css` and that you want your style sheet to be listed in the View menu as "`Style sheet #1`".

   Add the following css [70] configuration element to your custom `.xxe` file:

```
<css [70] name="Style sheet #1"
     location="stylesheet1.css"
     alternate="true" />
```

5. If you want to make your custom CSS style sheet the default one, add the following windowLayout [115] configuration element:

```
<windowLayout [115]>
  <center css="Style sheet #1" />
</windowLayout>
```

## 5. Replacing an existing CSS style sheet

Add the following css [70] configuration element to your custom `.xxe` file:

```
<css [70] name="Name of the CSS style sheet to be replaced"
     location="stylesheet1.css"
     alternate="true or false: copy the original value" />
```

Specify the *English* name [28] of the CSS style sheet as listed in the View menu.

## 6. Removing an existing CSS style sheet

Add the following css [70] configuration element to your custom `.xxe` file:

```
<css [70] name="Name of the CSS style sheet to be removed" />
```

Specify the *English* name [28] of the CSS style sheet as listed in the View menu.

## 7. Adding buttons to the tool bar

1. Add the following toolBar [105] configuration element to your custom `.xxe` file:

```
<toolBar [105]>
  <insert />
</toolBar>
```

2. After the `insert` element, add one or more `separator` and/or `button` elements. Example:

```
<toolBar>
  <insert />
  <separator />
  <button toolTip="TEST" icon="xxe-config:common/icons2/help.gif">
    <command name="alert" parameter="TEST"/>
  </button>
</toolBar>
```

## 8. Adding items to the menu

1. Add the following menu [90] configuration element to your custom `.xxe` file:

```
<menu [90] label="-">
  <insert />
</menu>
```

Attribute `label` is required. The value `-` simply means that you do not want to change the original label of the menu.

2. After the `insert` element, add one or more `separator` and/or `item` and/or `menu` elements. Example:

```
<menu label="_DocBook">
  <insert />
  <separator />
  <item label="TEST #_1" icon="xxe-config:common/icons2/help.gif"
        command="alert" parameter="TEST #1" />
  <separator />
  <menu label="SUBMENU">
    <item label="TEST #_2"
          command="alert" parameter="TEST #2" />
  </menu>
</menu>
```

- The `icon` attribute is optional for `item` elements.
- The "_" character in the `label` attribute is optional. It is used to specify the position of the menu mnemonic, if any.

## 9. Parametrizing the XSLT style sheets used in the Convert Document submenu

Add one or more parameterGroup [95] configuration elements to your custom `.xxe` file:

```
<parameterGroup [95] name="Name of the parameter group">
  <parameter name="Name of parameter #1">Value or parameter #1</parameter>
  <parameter name="Name of parameter #2">Value or parameter #2</parameter>
  <parameter name="Name of parameter #3">Value or parameter #3</parameter>
</parameterGroup>
```

Which parameters to specify is found by reading the documentation of the XSLT style sheets. For example, the reference manual of the DocBook XSLT style sheets is: DocBook XSL Stylesheet Documentation.

| Configuration Name | Convert to | Name of the `parameterGroup` |
|---|---|---|
| DITA<br><br>DITA Map<br><br>DITA BookMap | XHTML multi-page | `dita.toXHTML.transformParameters` |
| | XHTML single page | `dita.toXHTML1.transformParameters` |
| | HTML Help | `dita.toHTMLHelp.transformParameters` |
| | Java Help | `dita.toJavaHelp.transformParameters` |
| | RTF, WordprocessingML, OpenDocument, OOXML | `dita.toRTF.transformParameters` |
| | PDF, PostScript | `dita.toPS.transformParameters` |
| DocBook v5+ | HTML multi-page | `db5.toHTML.transformParameters` |
| | HTML single page | `db5.toHTML1.transformParameters` |
| | HTML Help | `db5.toHTMLHelp.transformParameters` |
| | Java Help | `db5.toJavaHelp.transformParameters` |
| | Eclipse Help | `db5.toEclipseHelp.transformParameters` |
| | RTF, WordprocessingML, OpenDocument, OOXML | `db5.toRTF.transformParameters` |
| | PDF, PostScript | `db5.toPS.transformParameters` |
| DocBook | HTML multi-page | `docb.toHTML.transformParameters` |
| | HTML single page | `docb.toHTML1.transformParameters` |
| | HTML Help | `docb.toHTMLHelp.transformParameters` |
| | Java Help | `docb.toJavaHelp.transformParameters` |

| Configuration Name | Convert to | Name of the `parameterGroup` |
|---|---|---|
| | Eclipse Help | `docb.toEclipseHelp.transformParameters` |
| | RTF, WordprocessingML, OpenDocument, OOXML | `docb.toRTF.transformParameters` |
| | PDF, PostScript | `docb.toPS.transformParameters` |
| XHTML Strict<br>XHTML Transitional<br>XHTML 1.1<br>XHTML 5 | RTF, WordprocessingML, OpenDocument, OOXML | `xhtml.toRTF.transformParameters` |
| | PDF, PostScript | `xhtml.toPS.transformParameters` |

Example: Use UTF-8 encoding when convert DocBook documents to multi-page HTML:

```
<parameterGroup name="docb.toHTML.transformParameters">
  <parameter name="chunker.output.encoding">UTF-8</parameter>
  <parameter name="saxon.character.representation">native;decimal</parameter>
</parameterGroup>
```

Example: When converting DocBook v5+ document to RTF, WordprocessingML, OpenDocument, OOXML or to PDF, PostScript, style `variablelist` like XXE does it on screen. That is, do not put the term and its definition side by side.

```
<parameterGroup name="db5.toRTF.transformParameters">
  <parameter name="variablelist.as.blocks">1</parameter>
</parameterGroup>

<parameterGroup name="db5.toPS.transformParameters">
  <parameter name="variablelist.as.blocks">1</parameter>
</parameterGroup>
```

## 10. Customizing the XSLT style sheets used in the Convert Document submenu

In order to do this, you need to use a custom XSLT style sheet instead of the stock one. Of course, the custom XSLT style sheet includes the stock one, so you can concentrate on your customizations.

Once you have created your custom XSLT style sheet, you have to specify to XXE that it must use it instead on the stock one. This is done by the means of a system property having the proper name and value.

1. Copy one of the following template files depending on which configuration you want to customize and on which format you want to generate:

| Configuration Name | Convert to | Procedure |
|---|---|---|
| DITA<br>DITA Map<br>DITA BookMap | XHTML multi-page | Copy xhtml.xsl to `custom/`. |
| | XHTML single page | Copy xhtml.xsl to `custom/`. |
| | HTML Help | Copy htmlhelp.xsl to `custom/`. |
| | Java Help | Copy javahelp.xsl to `custom/`. |
| | RTF, WordprocessingML, OpenDocument, OOXML | Copy fo.xsl to `custom/`. |
| | PDF, PostScript | Copy fo.xsl to `custom/`. |
| DocBook v5+ | HTML multi-page | Copy chunk.xsl to `custom/`. |

| Configuration Name | Convert to | Procedure |
|---|---|---|
| | HTML single page | Copy html.xsl to `custom/`. |
| | HTML Help | Copy htmlhelp.xsl to `custom/`. |
| | Java Help | Copy javahelp.xsl to `custom/`. |
| | Eclipse Help | Copy eclipse.xsl to `custom/`. |
| | RTF, WordprocessingML, OpenDocument, OOXML | Copy fo.xsl to `custom/`. |
| | PDF, PostScript | Copy fo.xsl to `custom/`. |
| DocBook | HTML multi-page | Copy chunk.xsl to `custom/`. |
| | HTML single page | Copy html.xsl to `custom/`. |
| | HTML Help | Copy htmlhelp.xsl to `custom/`. |
| | Java Help | Copy javahelp.xsl to `custom/`. |
| | Eclipse Help | Copy eclipse.xsl to `custom/`. |
| | RTF, WordprocessingML, OpenDocument, OOXML | Copy fo.xsl to `custom/`. |
| | PDF, PostScript | Copy fo.xsl to `custom/`. |
| XHTML Strict<br><br>XHTML Transitional<br><br>XHTML 1.1<br><br>XHTML 5 | RTF, WordprocessingML, OpenDocument, OOXML | Copy fo.xsl to `custom/`. |
| | PDF, PostScript | Copy fo.xsl to `custom/`. |

For example, DocBook v5+ `chunk.xsl` looks like this:

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

  <xsl:import href="docbook5-config:xsl/html/chunk.xsl"/>

</xsl:stylesheet>
```

2. Edit this file using an XML or text editor and add one or more XSLT elements after the `xsl:import` element.

   DocBook `html.xsl` example: Use the UTF-8 encoding instead of default ISO-8859-1 when converting a DocBook document to *single page* HTML[4]:

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

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

  <xsl:output method="html"
              encoding="UTF-8"
              indent="no"
              saxon:character-representation="native;decimal"/>
</xsl:stylesheet>
```

[4]XSLT style sheet parameter `chunker.output.encoding` does not work in this case.

DocBook `fo.xsl` example: add more information to the title page of book:

```
<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>
```

3. Add one of the following property [94] configuration element to your custom `.xxe` file:

| Configuration Name | Convert to | Property Configuration Element |
|---|---|---|
| DITA<br><br>DITA Map<br><br>DITA BookMap | XHTML multi-page | `<property name="dita.toXHTML.transform"`<br>`  url="true">xhtml.xsl</property>` |
| | XHTML single page | `<property name="dita.toXHTML1.transform"`<br>`  url="true">xhtml.xsl</property>` |
| | HTML Help | `<property name="dita.toHTMLHelp.transform"`<br>`  url="true">htmlhelp.xsl</property>` |
| | Java Help | `<property name="dita.toJavaHelpStep1.transform"`<br>`  url="true">javahelp.xsl</property>` |
| | RTF, WordprocessingML, OpenDocument, OOXML | `<property name="dita.toRTF.transform"`<br>`  url="true">fo.xsl</property>` |
| | PDF, PostScript | `<property name="dita.toPS.transform"`<br>`  url="true">fo.xsl</property>` |
| DocBook v5+ | HTML multi-page | `<property name="db5.toHTML.transform"`<br>`  url="true">chunk.xsl</property>` |
| | HTML single page | `<property name="db5.toHTML1.transform"`<br>`  url="true">html.xsl</property>` |
| | HTML Help | `<property name="db5.toHTMLHelp.transform"`<br>`  url="true">htmlhelp.xsl</property>` |
| | Java Help | `<property name="db5.toJavaHelpStep1.transform"`<br>`  url="true">javahelp.xsl</property>` |
| | Eclipse Help | `<property name="db5.toEclipseHelp.transform"`<br>`  url="true">eclipse.xsl</property>` |

| Configuration Name | Convert to | Property Configuration Element |
|---|---|---|
| | RTF, WordprocessingML, OpenDocument, OOXML | `<property name="db5.toRTF.transform" url="true">fo.xsl</property>` |
| | PDF, PostScript | `<property name="db5.toPS.transform" url="true">fo.xsl</property>` |
| DocBook | HTML multi-page | `<property name="docb.toHTML.transform" url="true">chunk.xsl</property>` |
| | HTML single page | `<property name="docb.toHTML1.transform" url="true">html.xsl</property>` |
| | HTML Help | `<property name="docb.toHTMLHelp.transform" url="true">htmlhelp.xsl</property>` |
| | Java Help | `<property name="docb.toJavaHelpStep1.transform" url="true">javahelp.xsl</property>` |
| | Eclipse Help | `<property name="docb.toEclipseHelp.transform" url="true">eclipse.xsl</property>` |
| | RTF, WordprocessingML, OpenDocument, OOXML | `<property name="docb.toRTF.transform" url="true">fo.xsl</property>` |
| | PDF, PostScript | `<property name="docb.toPS.transform" url="true">fo.xsl</property>` |
| XHTML Strict<br><br>XHTML Transitional<br><br>XHTML 1.1<br><br>XHTML 5 | RTF, WordprocessingML, OpenDocument, OOXML | `<property name="xhtml.toRTF.transform" url="true">fo.xsl</property>` |
| | PDF, PostScript | `<property name="xhtml.toPS.transform" url="true">fo.xsl</property>` |

## 11. Using a custom CSS style sheet to style the HTML files generated by the Convert Document submenu

Procedure:

1. Copy your custom CSS style sheet to the `custom/` directory.

   Let's suppose the name of the custom CSS style sheet is `fancy.css`.

2. Add one or more of the following property [94] configuration element to your custom `.xxe` file, depending on the kind of HTML files you want to style (HTML Help, Java Help, Eclipse Help and Epub are all HTML-based formats):

| Configuration Name | Convert to | Property Configuration Element |
|---|---|---|
| DITA<br><br>DITA Map | XHTML multi-page | `<property name="dita.toXHTML.resource.css" url="true">fancy.css</property>` |

| Configuration Name | Convert to | Property Configuration Element |
|---|---|---|
| DITA BookMap | XHTML single page | `<property name="dita.toXHTML1.resource.css" url="true">fancy.css</property>` |
| | HTML Help | `<property name="dita.toHTMLHelp.resource.css" url="true">fancy.css</property>` |
| | Java Help | `<property name="dita.toJavaHelpStep1.resource.css" url="true">fancy.css</property>` |
| DocBook v5+ | HTML multi-page | `<property name="db5.toHTML.resource.css" url="true">fancy.css</property>` |
| | HTML single page | `<property name="db5.toHTML1.resource.css" url="true">fancy.css</property>` |
| | HTML Help | `<property name="db5.toHTMLHelp.resource.css" url="true">fancy.css</property>` |
| | Java Help | `<property name="db5.toJavaHelpStep1.resource.css" url="true">fancy.css</property>` |
| | Eclipse Help | `<property name="db5.toEclipseHelp.resource.css" url="true">fancy.css</property>` |
| | Epub | `<property name="db5.toEpub.resource.css" url="true">fancy.css</property>` |
| DocBook | HTML multi-page | `<property name="docb.toHTML.resource.css" url="true">fancy.css</property>` |
| | HTML single page | `<property name="docb.toHTML1.resource.css" url="true">fancy.css</property>` |
| | HTML Help | `<property name="docb.toHTMLHelp.resource.css" url="true">fancy.css</property>` |
| | Java Help | `<property name="docb.toJavaHelpStep1.resource.css" url="true">fancy.css</property>` |
| | Eclipse Help | `<property name="docb.toEclipseHelp.resource.css" url="true">fancy.css</property>` |
| | Epub | `<property name="docb.toEpub.resource.css" url="true">fancy.css</property>` |

# Chapter 6. Deploying XXE

## 1. Dynamic discovery of add-ons

This section describes how XXE discovers and loads *add-ons* (that is, extensions) of all types:

- configuration files,
- XML catalogs,
- translations of XXE messages (menu labels, button labels, error messages, etc) to languages other than English,
- spell-checker dictionaries,
- XSL-FO processor, image toolkit and virtual drive plug-ins,
- customizations of the XXE GUI.

Understanding this is important before learning how to deploy XXE.

### About the integrated add-on manager

What is described in this chapter is not related to XXE integrated add-on manager (menu item Options → Install Add-ons).

The integrated add-on manager is just a facility which, in order to install add-ons, follows the rules described in this chapter.

For example, in order to install an add-on packaged as a Zip archive, the add-on manager simply unzips this archive in one of the two `addon/` directories scanned by XXE at startup time.

## 1.1. The lookup phase during XXE startup

During its startup:

1. XXE recursively scans the `addon/` subdirectory of XXE user preferences directory searching it for files containing add-ons.

   XXE user preferences directory is:

   - *`$HOME`*`/.xxe5/` on Linux.
   - *`$HOME`*`/Library/Application Support/XMLmind/XMLEditor5/` on the Mac.
   - *`%APPDATA%`*`\XMLmind\XMLEditor5\` on Windows XP, Vista, 7 and 8.

     Example: `C:\Documents and Settings\john\Application Data\XMLmind\XMLEditor5\` on Windows XP. `C:\Users\john\AppData\Roaming\XMLmind\XMLEditor5\` on Windows Vista, 7 and 8.

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

   ### Tip

   This `addon/` subdirectory is *recursively* scanned by XXE at startup time. Therefore, feel free to organize it as you want.

2. If the `XXE_ADDON_PATH` variable is set to a non empty string, the content of this variable must be a list of *directory* names separated by character "`;`" (even on Unix). All the *directories* referenced in this list are recursively scanned by XXE.

   - File names and "`file://`" URLs are both supported. Windows example:

     ```
     C> set XXE_ADDON_PATH=C:\xxe\doc\configure\samples\example1;\
     file:///C:/xxe/doc/configure/samples/example2
     ```

   - If this path ends with "`;+`", the `addon/` subdirectory of XXE installation directory is also scanned at startup time. Otherwise, the default add-ons (XHTML configuration, DocBook configuration, etc) are ignored.

   - Form `@`*`absolute URL`* is also supported.

     *`Absolute URL`* specifies the location of a text file containing a list of (generally relative) URLs to be scanned by XXE. The URLs in this list are separated by white space.

     Example, `sample_configs.list`:

     ```
     example1
     example1/example1.css
     example1/example1.dtd
     example1/example1.xml
     example1/example1.xxe
     example1/example1_catalog.xml
     example2
     example2/example2.css
     example2/example2.xml
     example2/example2.xsd
     example2/example2.xxe
     example2/example2_catalog.xml
     ```

     Unix example:

     ```
     $ export XXE_ADDON_PATH="@http://www.foo.com/xxe/sample_configs.list;+"
     ```

3. If the `XXE_ADDON_PATH` is not set or is set to an empty string or ends with "`;+`", XXE also recursively scans the `addon/` subdirectory of its installation directory searching it for files containing add-ons.

   ### Tip

   This `addon/` subdirectory is *recursively* scanned by XXE at startup time. Therefore, feel free to organize it as you want.

4. Jar files (`.jar` files containing compiled Java™ code) found anywhere inside a directory or a file list scanned by XXE during its startup are automatically added to the `CLASSPATH` of XXE.

   ## About Jar files containing native libraries

   Some Jar files may contain native libraries. For example: `hunspell.dll` for Windows 32-bit and `libhunspell64.so` for Linux Intel™ 64-bit.

   When this is the case, it is recommended to create one Jar file per OS/architecture and to give these Jar files filenames following the convention explained below. For example, `hunspell.dll` should be contained in `hunspell--Windows-x86.jar` and `libhunspell64.so` should be contained in `hunspell--Linux-amd64.jar`.

   By doing this, you'll instruct XXE, for example, to ignore `hunspell--Linux-amd64.jar` and just consider `hunspell--Windows-x86.jar` when it is started on Windows.

   Filename syntax:

```
jar_basename -> jar_name '--' os_name [ '-' os_arch ]? '.jar'
```

- *`os_name`* must match the value of Java™ system property `os.name` (though for Windows, you may skip the "`XP`", "`Vista`", "`7`", "`8`" suffix and keep just the "`Windows`" prefix).
- *`os_arch`* must match the value of Java system property `os.arch`.

Examples:

| ***`os_name`*** | ***`os_arch`*** |
|---|---|
| `Windows` | Intel 32-bit: `x86` |
| | Intel 64-bit: `amd64` |
| `Mac OS X` | Intel 32-bit: `i386` |
| | Intel 64-bit: `x86_64` |
| `Linux` | Intel 32-bit: `i386` |
| | Intel 64-bit: `amd64` |

The deploywebstart command-line tool [43] also understands the above filename convention and will wrap these Jar files into appropriate `resource` elements.

## 1.2. Files containing the add-ons

Configuration file

XXE configuration files are XML files:

- with a file name ending with "`.xxe`",
- validated by XML schema with `http://www.xmlmind.com/xmleditor/schema/configuration` as its target namespace,
- with a root element named `configuration`,
- this root element having a `name` attribute,
- containing a `detect` element.

Several configurations may have the same name. For example, a user may have defined its own configuration named "`DocBook`" including bundled configuration also named "`DocBook`" but adding element templates and keyboard shortcuts (see include [83], elementTemplate [77], binding [61]). In such case, only one configuration named "`DocBook`" is kept by XXE: the configuration with highest priority.

Configurations loaded from the `addon/` subdirectory of user preferences directory have priority over configurations loaded from the value of environment variable `XXE_ADDON_PATH` which in turn have priority over configurations loaded from the `addon/` subdirectory of XXE installation directory.

Configurations having the same priority are sorted using their file basenames. Example: `file:///opt/xxe/foo/docbook.xxe` is tested before `file:///opt/xxe/bar/sdocbook.xxe` when trying to detect the class of a document because `docbook.xxe` lexicographically precedes `sdocbook.xxe`.

XML catalogs

XML catalogs are XML files:

- with a file name ending with "`atalog.xml`",
- which conform to the OASIS catalog DTD.

Example:

```
<?xml version="1.0" ?>
<!DOCTYPE catalog PUBLIC "-//OASIS//DTD XML Catalogs V1.0//EN"
  "http://www.oasis-open.org/committees/entity/release/1.0/catalog.dtd">

<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
         prefer="public">

  <public publicId="-//W3C//DTD SVG 1.1//EN"
          uri="common/dtd/svg11/svg11.dtd"/>

</catalog>
```

Note that specifying the above `<!DOCTYPE>` will *not* cause the XML catalog parser to download XML Catalog DTD, `catalog.dtd`, from the Web.

XXE uses XML Catalogs not only to resolve the locations of the DTD and other external entities, but also to resolve URLs found in the following places:

- Schema locations in `xsi:schemaLocation` and in `xsi:noNamespaceSchemaLocation`.
- Schema locations in `xs:include`, `xs:redefine`, `xs:import`.
- Schema locations in `<?xml-model href="..."?>`.
- Document locations passed to the `document()` XPath function.
- All XXE configuration elements referencing an URL. Example: `<include location="..."/>`.
- CSS style sheet locations in `@import`.
- CSS style sheet locations in `<?xml-stylesheet href="..."?>`.
- XSLT style sheets in the `transform` child element of a `process` command.
- Resources in the `copyProcessResource` child element of a `process` command.
- XSLT style sheets included or imported by other XSLT style sheets (that is, the XML Catalogs used by XXE are passed to Saxon, the XSLT engine bundled with XXE).
- The `href` attribute of `xi:include` elements (XInclude).

Translations of XXE messages (menu labels, button labels, error messages, etc) to languages other than English

Translations of XXE messages are contained in Java™ jars:

- with a file name ending with "`.jar`",
- having a basename which is the two-letter ISO code of the language followed by "`_translation`" (e.g. `de_translation.jar`, `it_translation.jar`, `cs_translation.jar`, `es_translation.jar`, etc). Not mandatory, just recommended.

Spell-checker dictionaries

Spell-checker dictionaries are contained in Java™ jars:

- with a file name ending with "`.dar`",
- having a basename which is the ISO code of a language (e.g. `fr`, `fr-CH`, `en`, `en-US`, etc).

  This naming pattern is highly recommended for dictionaries found in the local file system. This naming pattern is *mandatory* for dictionaries centralized on an HTTP or an FTP server.

XSL-FO processor plug-ins

XSL-FO processor plug-ins are contained in Java™ jars:

- with a file name ending with "`_foprocessor.jar`",
- implementing service `com.xmlmind.foprocessor.FOProcessor`.

The exact structure of a plug-in jar (manifest, service providers, etc) is described in Chapter 10, *Writing a plug-in* in *XMLmind XML Editor - Developer's Guide*.

Image toolkit plug-ins
: Image toolkit plug-ins are contained in Java™ jars:

- with a file name ending with "`_imagetoolkit.jar`",
- implementing service `com.xmlmind.xmledit.imagetoolkit.ImageToolkit`.

Virtual drive plug-ins
: Virtual drive plug-ins are contained in Java™ jars:

- with a file name ending with "`_vdrive.jar`",
- implementing service `com.xmlmind.xmleditapp.vdrive.DriveFactory`.

Customizations of XXE GUI
: Such customizations are contained in XML files called `customize.xxe_gui` and conforming to the "`http://www.xmlmind.com/xmleditor/schema/gui`" W3C XML Schema.

Such GUI specification files are described in XMLmind XML Editor - Customizing the User Interface.

If during its start-up, XXE finds several `customize.xxe_gui` files, it will merge their contents with the *base* GUI specification (by default, `xxe-gui:app/Professional.xxe_gui`, which is a resource contained in `xxe_app.jar`).

## 2. Centralizing add-ons on a HTTP server

1. Install XXE on the server. Example: `/opt/xxe/` on a server called `rapido`.
2. Customize XXE if needed to. Example:
   - Create directory `/opt/xxe/addon/custom/`. This directory will contain all the add-ons you want to deploy.
   - Unzip one or more add-on distributions in `/opt/xxe/addon/custom/`.

     You'll find the add-on distributions packaged by XMLmind in http://www.xmlmind.com/xmleditor/addons.shtml.

     **Tip**

     This customization of the XXE distribution can be done by hand by unpacking add-on distributions anywhere inside `/opt/xxe/addon/`, or more simply by using the integrated add-on manager (Options → Install Add-ons).

     In the latter case, just make sure to check "Install add-ons in XXE installation directory" in the Preferences dialog box (Options → Preferences, Install add-ons section) before using the add-on manager.
3. Test your customized distribution by running `/opt/xxe/bin/xxe` on the server.

### Note

If you use the RenderX XEP plug-in, make sure that you have finished its installation by converting at least a document to PDF or PostScript®.

4. In `/opt/xxe/addon/`, run command "`find custom > custom.list`" to create text file `custom.list`. The following example assumes that you have unpacked `xfc_foprocessor.zip` in `custom/`.

   ```
   /opt/xxe/addon$ find custom > custom.list

   /opt/xxe/addon$ cat custom.list

   custom
   custom/xfc_foprocessor
   custom/xfc_foprocessor/xfc.LICENSE
   custom/xfc_foprocessor/xfc.README
   custom/xfc_foprocessor/xfc.jar
   custom/xfc_foprocessor/xfc_foprocessor.jar
   custom/xfc_foprocessor/xfc_foprocessor.xxe_addon
   ...
   ```

5. Publish your customized distribution on your intranet using a HTTP server. Apache example:

   a. Add a similar snippet to `/etc/httpd.conf`:

      ```
      <Directory /opt/xxe/>
        AllowOverride None
        Order Deny,Allow
        Deny from All
        Allow from my_company.com

        Options Indexes Includes
      </Directory>
      Alias /xxe /opt/xxe/
      ```

   b. Restart apache:

      ```
      # cd /etc/rc.d
      # ./apache restart
      ```

6. Now, the hardest part: make sure that the PCs of the all future XXE users on your intranet have the following environment variable always properly set (for example: add it to `autoexec.bat`).

   ```
   set XXE_ADDON_PATH="@http://rapido.my_company.com/xxe/addon/custom.list;+"
   ```

   Notice that you can update or upgrade the distribution on the server side without having to change this environment variable on the client side.

7. Tell all your XXE users to download a copy of the XXE installer (that is, `xxe-perso-`*`NNN`*`-setup.exe` or `xxe-pro-`*`NNN`*`-setup.exe`) from your intranet and to install it on their PCs.

## 3. Deploying XXE using Java™ Web Start

This section assumes that the reader knows what is Java™ Web Start.

### Important

XXE requires `<security><all-permissions/></security>` in order to run. This implies that all `.jar` files must be digitally signed using the same certificate. Fortunately, the `deploywebstart` command-line utility makes this a snap. See below.

## 3.1. The deploywebstart command-line tool

Usage: `deploywebstart` *`?options?`*

Basic options are:

-codebase *`url`*
: Base URL for all relative URLs in `xxe.jnlp`.

  Default: none when XXE is deployed as an applet (because in such case, specifying a codebase is not strictly needed); `http:`*`//machine name on which deploywebstart was run/`*`xxe` when XXE is deployed using Java™ Web Start.

-keystore *`url`*
: Keystore location.

  Default: *`XXE_install_dir/`*`webstart/testkeystore`

-storetype *`type`*
: Type of the keystore.

  Default: system dependant, generally `jks`

-storepass *`password`*
: Password for keystore.

  Default: `teststorepass`

-keypass *`password`*
: Password for private key.

  Default: `testkeypass`

-alias *`alias`*
: Alias of keystore entry.

  Default: login name of person running `deploywebstart`.

-index
: Generate a simple `index.html`.

Applet-specific options are:

-applet xxe|viewer|editor1|editor2|*`app_class`*, -jsapplet xxe|viewer|editor1|editor2|*`app_class`*
: Deploy application having specified class name as an applet rather than using Java™ Web Start.

  "`xxe`" is a shorthand for "`com.xmlmind.xmleditapp.app.Application`".

  "`viewer`" is a shorthand for "`com.xmlmind.xmleditapp.applet.ViewerApp`".

  "`editor1`" is a shorthand for "`com.xmlmind.xmleditapp.applet.Editor1App`".

  "`editor2`" is a shorthand for "`com.xmlmind.xmleditapp.applet.Editor2App`".

  Option `-jsapplet` is a variant of option `-applet`. Instead of generating an `applet` tag in `index.html`, it includes in `index.html` some smart JavaScript code which, depending on the Web browser host, will dynamically generate the proper `object` or `applet` tag.

-width *`pixels_or_percentage`*
: Width of the applet.

Default: 100%.

-height *pixels_or_percentage*
: Height or the applet.

  Default: 600.

Advanced options are:

-selfsigner *dname*
: Specifies a distinguished name (dname) for *testkeystore*. Ignored unless `testkeystore` is used. That is, this option is ignored when a real certificate is used.

  The syntax for distinguished names (dname) is:

```
CN=cName,OU=orgUnit,O=org,L=city,S=state,C=countryCode
```

  where:

  *cName*
  : common name of a person, e.g., '`Susan Jones`'.

  *orgUnit*
  : department or division name, e.g., '`Purchasing`'.

  *org*
  : large organization name, e.g., '`ABCSystems\, Inc.`' (notice the '`\`' used to protect the '`,`').

  *city*
  : city name, e.g., '`Palo Alto`'.

  *state*
  : state or province name, e.g., '`California`'.

  *countryCode*
  : two-letter country code, e.g., '`CH`'.

  Each field must appear in the above order but it is not necessary to specify all fields.

  Default: CN=login name of the person running `deploywebstart`.

  *Using this option is absolutely not needed to ``self-sign'' jars. It just allows to create a better looking self-signed certificate.*

-online
: Keep configuration files and associated resources (DTD or schema, CSS, XSLT, icons, etc) on the deployment server. This forces the XXE user to work online in order to be able to access the deployment server.

  Default: allow the XXE user to work offline.

-gui *XXE_GUI_spec*
: Specifies which *base* GUI specification to use. Must be a "`xxe-gui:`" location or a `.xxe_gui` file found in the *XXE_install_dir*`/addon/` directory.

  Default: `xxe-gui:app/Professional.xxe_gui`

  One or more `customize.xxe_gui` files (dynamically discovered by `deploywebstart` in the *XXE_install_dir*`/addon/`) may be used to customize this base GUI specification.

-indexjars
: Index all `.jar` files in order to lazily download them.

-packjars
: Compress all `.jar` files using Pack200 compression.

-quiet
: Turns verbosity off.

The `deploywebstart` command line tool generates deployment files in subdirectory `webstart/` of the XXE installation directory.

For example, if XXE is installed in `/opt/xxe/`, `/opt/xxe/bin/deploywebstart` will recursively scan the installation directory and generates its deployment files in `/opt/xxe/webstart/`.

### Caution

Note that `deploywebstart` only scans the `addon/` subdirectory of the installation directory of XXE. It completely ignores the add-ons found in the `addon/` subdirectory of the user preferences directory [37].

`Deploywebstart` creates in `webstart/`:

- `xxe.jnlp`.
- `index.html`, if the `-index` option has been used.
- A copy of all the `.jar` files (Java™ code and resources) and the `.dar` files (spell-checker dictionaries) found in `*XXE_install_dir*/addon/` after signing them.
- `xxe_addon.jar`, a jar file created and signed by `deploywebstart` containing everything found in the `*XXE_install_dir*/addon/` directory (expect `.jar` files and `.dar` files), unless the `-online` option has been used.

By default, `deploywebstart` signs the jars with a self-signed certificate issued by the person running this command-line utility.

Note that because of the default values of these options, if you need to sign the jars with a true certificate, you will have to specify *all the four* `-storepass`, `-keystore`, `-keypass`, `-alias` options.

## 3.2. Deploying XXE using Java™ Web Start, a step by step description

1. Install a Java™ 1.6+ JDK (a JRE is not sufficient) on the deployment server. Example: let's call this server `rapido`.

   ### Important

   Make sure that the `*$JAVA_HOME*/bin/` directory is referenced in `$PATH` because `deploywebstart` needs to run command line tools such as `keytool` and `jarsigner`.

2. Install XXE on the server. Example: `/opt/xxe/`.

3. Customize XXE if needed to. Example:

   - Create directory `/opt/xxe/addon/custom/`. This directory will contain all the add-ons you want to deploy.
   - Unzip one or more add-on distributions in `/opt/xxe/addon/custom/`.

     You'll find the add-on distributions packaged by XMLmind in http://www.xmlmind.com/xmleditor/addons.shtml.

### Tip

This customization of the XXE distribution can be done by hand by unpacking add-on distributions anywhere inside `/opt/xxe/addon/`, or more simply by using the integrated add-on manager (Options → Install Add-ons).

In the latter case, just make sure to check "Install add-ons in XXE installation directory" in the Preferences dialog box (Options → Preferences, Install add-ons section) before using the add-on manager.

4. Test your customized distribution by running `/opt/xxe/bin/xxe` on the server.

   ### Note

   If you use the RenderX XEP plug-in, make sure that you have finished its installation by converting at least a document to PDF or PostScript®.

5. Run the `deploywebstart` command-line tool:

   ```
   /opt/xxe/bin$ ./deploywebstart -index
   ```

   - The above command generates in `/opt/xxe/webstart/` a `xxe.jnlp` file describing the deployment of XXE using Java™ Web Start.
   - `-index` is used to generate a simple `index.html` file in `/opt/xxe/webstart/`.
   - The default codebase `http://rapido.my_company.com/xxe` should work fine for this example. If this is not the case, you'll have to use the `-codebase` option.
   - Jars are signed using a self-signed certificate issued by the person who ran `deploywebstart`. Let's call him `john` (its login name is `john`).

     The first time a user will start XXE, Java™ Web Start will display a dialog box telling him that XMLmind XML Editor code has been signed by `john` (a coworker name known by the user) and that it is strongly not recommended to run such application.

     In our opinion, this is not a problem for applications deployed on a intranet. In this happens to be a problem, first add a true certificate (that is, purchased from VeriSign for example) using the `keytool` command line supplied by Sun in its JDK, then use all the four `-storepass`, `-keystore`, `-keypass`, `-alias` options to specify who is signing the jars.

6. Make sure that the files generated in `/opt/xxe/webstart/` may be read by everyone:

   ```
   /opt/xxe/webstart$ chmod a+rx .
   /opt/xxe/webstart$ chmod a+r *.*
   ```

7. Publish your customized distribution on your intranet using a HTTP server. Apache example:

   a. Add the following MIME type to `/etc/httpd/mime.types`:

   ```
   application/x-java-jnlp-file    jnlp
   ```

   b. Add a similar snippet to `/etc/httpd.conf`:

   ```
   <Directory /opt/xxe/webstart/>
     AllowOverride None
     Order Deny,Allow
     Deny from All
     Allow from my_company.com

     Options Indexes Includes
   ```

```
</Directory>
Alias /xxe /opt/xxe/webstart/
```

c. Restart apache:

```
# cd /etc/rc.d
# ./apache restart
```

8. Tell all your future XXE users to download and install Java™ Runtime 1.6+ on their PCs. This will also automatically install Java™ Web Start.

   You can use this technology to deploy not only XXE, but also any other application written in the Java™ language.

9. Tell all your future XXE users to visit `http://rapido.my_company.com/xxe` (this will display the generated index.html) and to launch XXE from there, at least the first time.

## 3.3. Comparison between deployment using Java Web Start and just centralizing the add-ons on a HTTP server

| **Deploying XXE using Java™ Web Start** | **Centralizing add-ons on a HTTP server** |
|---|---|
| XXE code is downloaded and cached on the PC of the XXE user. | XXE code is installed by the XXE installer on the PC of the XXE user. |
| Add-ons of all sorts are downloaded and cached on the PC of the XXE user.<br><br>(Use the `-online` option if you prefer to keep the add-ons on the deployment server.) | Add-ons of all sorts stay on the server and therefore are not cached on the PC of the XXE user. |
| User can work offline (unless the `-online` option has been used). | User *cannot work offline*. |
| The user cannot install extra add-ons using Options → Install Add-ons. | The user can install extra add-ons using Options → Install Add-ons. |
| Add-ons found in the user preferences directory of user (that is, `%APPDATA%\XMLmind\XMLEditor5\addon\` on Windows and `$HOME/.xxe5/addon/` on Linux) *are ignored*. | Add-ons found in the user preferences directory of user are loaded. |
| Upgrading, for example, from XXE v4.5 to XXE v4.6 is automated for the user. | For example, user will have to manually uninstall XXE v4.5 and then to manually download and install XXE v4.6. |

# 4. Deploying XXE as an applet

## 4.1. Requirements

Sun's ``next generation Java plug-in" can be used to run XMLmind XML Editor as an applet. The requirements for running XMLmind XML Editor as an applet are:

- A Java™ runtime version 1.6.0_10 and above.
- A Web browser supporting the next generation Java™ plug-in. Currently such browsers are: Firefox 3, Google Chrome and Internet Explorer 6, 7 and 8.
- The applet must be signed using a digital certificate.

## 4.2. Testing the XXE applet, a step by step description

The procedure below is identical to the one used to deploy XXE using Java™ Web Start [45]. The only difference is that you need to use different options for the `deploywebstart` command-line utility [43].

1. Install a Java™ 1.6+ JDK (a JRE is not sufficient) on the deployment server. Example: let's call this server `rapido`.

   ### Important

   Make sure that the `$JAVA_HOME/bin/` directory is referenced in `$PATH` because `deploywebstart` needs to run command line tools such as `keytool` and `jarsigner`.

2. Install XXE on the server. Example: `/opt/xxe/`.

3. Customize XXE if needed to. Example:

   - Create directory `/opt/xxe/addon/custom/`. This directory will contain all the add-ons you want to deploy.

     ### Caution

     Note that `deploywebstart` only scans the `addon/` subdirectory of the installation directory of XXE. It completely ignores the add-ons found in the `addon/` subdirectory of the user preferences directory [37].

   - Unzip one or more add-on distributions in `/opt/xxe/addon/custom/`.

     You'll find the add-on distributions packaged by XMLmind in http://www.xmlmind.com/xmleditor/addons.shtml.

     ### Tip

     This customization of the XXE distribution can be done by hand by unpacking add-on distributions anywhere inside `/opt/xxe/addon/`, or more simply by using the integrated add-on manager (Options → Install Add-ons).

     In the latter case, just make sure to check "Install add-ons in XXE installation directory" in the Preferences dialog box (Options → Preferences, Install add-ons section) before using the add-on manager.

4. Test your customized distribution by running `/opt/xxe/bin/xxe` on the server.

   ### Note

   If you use the RenderX XEP plug-in, make sure that you have finished its installation by converting at least a document to PDF or PostScript®.

5. Run the `deploywebstart` command-line tool:

   ```
   /opt/xxe/bin$ ./deploywebstart -applet xxe -index
   ```

   - `-applet xxe` is used to generate in `/opt/xxe/webstart/` a `xxe.jnlp` file describing the deployment of XXE as an applet rather than using Java™ Web Start.

   - `-index` is used to generate a simple `index.html` file in `/opt/xxe/webstart/`. This file contains the applet element.

   - Jars are signed using a self-signed certificate issued by the person who ran `deploywebstart`. If you want to sign the jars with an actual certificate, you need to use all the four `-storepass`, `-keystore`, `-keypass`, `-alias` options.

6. Make sure that the files generated in `/opt/xxe/webstart/` may be read by everyone:

   ```
   /opt/xxe/webstart$ chmod a+rx .
   /opt/xxe/webstart$ chmod a+r *.*
   ```

7. Publish your customized distribution on your intranet using a HTTP server. Apache example:

   a. Add a similar snippet to `/etc/httpd.conf`:

      ```
      <Directory /opt/xxe/webstart/>
        AllowOverride None
        Order Deny,Allow
        Deny from All
        Allow from my_company.com

        Options Indexes Includes
      </Directory>
      Alias /xxe /opt/xxe/webstart/
      ```

   b. Restart apache:

      ```
      # cd /etc/rc.d
      # ./apache restart
      ```

8. Point your Web browser to `http://rapido.my_company.com/xxe`.

## 4.3. Integrating the applet with your web application

### 4.3.1. Dynamically generating an HTML page referencing the applet

A Web application typically consists of a front end communicating with a Servlet, PHP, ASP, CGI, etc, back end. The front end which runs in the Web browser generally consists in a mix of HTML and JavaScript. The code of the front end is dynamically generated by the back end in response to the preceding interactions with the user. Therefore, integrating the applet with a Web application generally means: teach the back end to dynamically generate a page referencing the applet.

In the above example, running:

```
./deploywebstart -applet xxe -index
```

- copies and signs the `.jar` files comprising the code of the applet;
- generates `xxe.jnlp`, an applet deployment descriptor;
- generates a *sample* `index.html` file, which can be used to quickly test that the applet works.

The signed `.jar` files and the `xxe.jnlp` files are expected to be used as is by your web application. Just moves these files to the actual deployment directory (which is the directory from which the Web browser will download the applet code). These files needs to be regenerated only when you'll upgrade XMLmind XML Editor.

The situation is different for the `index.html` file, which is not meant for production use. Instead, use the applet element contained in this file as a template for the applet element which will be dynamically generated by your back end:

```
<applet name="XXE" id="XXE" mayscript
  code="com.xmlmind.xmleditapp.applet.Applet"
  width="100%" height="600">
  <param name="jnlp_href" value="xxe.jnlp">
  <param name="appClass" value="com.xmlmind.xmleditapp.app.Application">
  <param name="separate_jvm" value="true">
  <param name="classloader_cache" value="false">
  Sorry but XMLmind XML Editor requires the Java<sup>TM</sup> Plug-In
  version 1.6.0_10 and above in order to run.
</applet>
```

For production use, you'll prefer to use a more elaborate way to deploy XXE as an applet:

```
./deploywebstart -indexjars -packjars -jsapplet xxe -index
```

Refer to Section 3.1, “The deploywebstart command-line tool” [43] to learn about the `-indexjars`, `-packjars` and `-jsapplet` options.

In production, instead of generating an applet element, your back end can generate some code similar to the one generated by `deploywebstart` using the `-jsapplet` option:

```
    <script type="text/javascript"
            src="http://java.com/js/deployJava.js"></script>
    <script type="text/javascript">
//<![CDATA[

      var attributes = {name:'XXE', id:'XXE', mayscript:true,
                        code:'com.xmlmind.xmleditapp.applet.Applet',
                        width:'100%', height:'600'};
      var parameters = {jnlp_href:'xxe.jnlp',
                        appClass:'com.xmlmind.xmleditapp.app.Application',
                        separate_jvm:true, classloader_cache:false};
      deployJava.runApplet(attributes, parameters, '1.6.0_10');

//]]>
    </script>
```

### 4.3.2. The four different kinds of applet

In the above code snippets, `jnlp_href`, `separate_jvm`, `classloader_cache` are system parameters documented in *Development and Deployment Of Java™ Web Apps (Applets and Java Web Start Applications) for JavaSE 6u10*.

Parameter *`appClass`* is specific to the XMLmind XML Editor applet. The value of this parameter is the fully qualified class name of the application deployed as an applet.

The value of this parameter is generally specified using the `-applet` or `-jsapplet` option of the `deploywebstart` command-line utility:

| Option Value | Class Name |
|---|---|
| `viewer` | `com.xmlmind.xmleditapp.applet.ViewerApp` |
| `editor1` | `com.xmlmind.xmleditapp.applet.Editor1App` |
| `editor2` | `com.xmlmind.xmleditapp.applet.Editor2App` |
| `xxe` | `com.xmlmind.xmleditapp.app.Application` |

There are four different applications which can be deployed as an applet. Each application is designed to solve a different problem. Each application has a different GUI:

viewer
: A document viewer having no user interface at all, except a right-click popup menu.

  In order to open a document in `viewer`, the integrator needs to wrap the corresponding command-line option in some applet parameters [51] or to script the applet [52].

editor1
: A single document, single view, document editor designed to be part of an HTML form.

  This editor has no menu bar and has no New, Open, Save tool bar buttons.

  In order to create a new document using `editor1`, the integrator needs to wrap the `-new` command-line option in some applet parameters [51] or to script the applet [52].

In order to open a document in `editor1`, the integrator needs to wrap the corresponding command-line option in some applet parameters or to script the applet.

In order to save the document edited in `editor1`, the integrator needs to make the applet part of an HTML form and to script it as follows:

1. Query the applet for the XML source of the document being edited.
2. Dynamically add to the form a `hidden` field containing this XML source.
3. Submit the form.

editor2
: A single document, single view, document editor designed to be integrated with a server supporting WebDAV, FTP, FTPS or SFTP (in fact, any kind of storage for which a virtual drive plug-in is available).

  Same user interface as `editor1`, expect that `editor2` has a Save tool bar button and fully supports Save preferences.

xxe
: Full XMLmind XML Editor.

  Same user interface as the desktop application, except that it has no File → Quit menu item.

  This is an alternative to deploying XMLmind XML Editor using Java™ Web Start.

Note that despite the fact the above applications have vastly different user interfaces, it is still XMLmind XML Editor. That is, *all the above applications support the same user preferences and the same add-ons as the desktop application*.

### 4.3.3. Applet parameters

In addition to the `appClass` parameter described in Section 4.3.2, “The four different kinds of applet” [50], the applet has up to 100 `argument`*`N`* parameters: `argument0`, `argument1`, `argument2`, ..., `argument99`. These parameters are used to wrap the command-line arguments supported by the desktop application.

The command-line arguments of XMLmind XML Editor are documented in Appendix A, *Command line options* in *XMLmind XML Editor - Online Help*.

The user preference keys and values of XMLmind XML Editor are documented in Section 5, “The "Preferences" dialog box” in *XMLmind XML Editor - Online Help*.

**Example 6.1. `-applet viewer` example: open a document stored on an HTTP server**

```
<applet ...>
  <param name="appClass" value="com.xmlmind.xmleditapp.applet.ViewerApp">
  <param name="argument0"
         value="http://www.xmlmind.com/xmleditor/_distrib/doc/user/userguide.xml">
  ...
</applet>
```

**Example 6.2. Same example as above, but force the value of the `defaultFontSize` user preference in order to use a larger font to display the document**

```
<applet ...>
  <param name="appClass" value="com.xmlmind.xmleditapp.applet.ViewerApp">
  <param name="argument0" value="-putpref">
  <param name="argument1" value="defaultFontSize">
  <param name="argument2" value="14">
  <param name="argument3"
         value="http://www.xmlmind.com/xmleditor/_distrib/doc/user/userguide.xml">
  ...
</applet>
```

**Example 6.3. `-applet editor1` example: create a new DocBook article**

```
<applet ...>
  <param name="appClass" value="com.xmlmind.xmleditapp.applet.Editor1App">
  <param name="argument0" value="-new">
  <param name="argument1" value="docbook">
  <param name="argument2" value="article">
  <param name="argument3" value="-">
  ...
</applet>
```

**Example 6.4. `-jsapplet editor2` example: open a document stored on a WebDAV server requiring the user to authenticate himself**

```
<script type="text/javascript">
  ...
  var parameters = {
    appClass:'com.xmlmind.xmleditapp.applet.Editor2App',
    argument0:'-auth',
    argument1:'CgpEb2N1bWVudCBTdG9yZQoKanZpY3RvcmlhCnNlY3JldA==',
    argument2:'http://www.acme.com/dav/report.xml',
    ...
  };
  ...
</script>
```

### 4.3.4. Applet scripting

Applet scripting means to use JavaScript™ code to invoke some methods of the applet.

Note that you cannot safely invoke any method of the applet. You can only invoke the applet methods which have been specially designed in order to be scripted in JavaScript. Such methods are:

void addAuthorization(String *`info`*), void addAuthorization(String *`host`*, int *`port`*, String *`prompt`*, String *`scheme`*, String *`username`*, String *`password`*)
: Add specified non-interactive authentication credentials.

boolean newDocument(String *`configName`*, String *`templateName`*, String *`saveURI`*, boolean *`createSaveFile`*), boolean newDocument(String *`templateURI`*, String *`saveURI`*, boolean *`createSaveFile`*), boolean newDocument(String *`template`*, String *`saveURI`*), boolean newDocumentFromTemplate(String *`template`*, String *`saveURI`*, boolean *`createSaveState`*)
: Create a new document.

boolean openDocument(String *`docURI`*, boolean *`readOnly`*), boolean openDocument(String doc, String *`docURI`*, boolean *`readOnly`*), boolean openDocument(String doc, String *`docURI`*, boolean *`readOnly`*, boolean *`createSaveState`*)
: Opens specified document.

boolean isSaveNeeded(String *`docURI`*)
: Tests whether specified document needs to be saved. Example of use:

```
    window.onbeforeunload = function() {
        var xxe = document.XXE;
        if (xxe.isSaveNeeded(null)) {
            return "You have not clicked the \"Submit\" button.";
        }
    }
```

boolean unsetSaveNeeded(String *`docURI`*)
: Resets the state of specified document to "Save Not Needed". Except for applets such as Editor1 [50], explicitly invoking this method is almost never needed.

boolean canCompareRevisions(String `originalDocURI`, String `revisedDocURI`)
: Tests whether specified documents can be compared using the Compare tool. Two documents can be compared if:

  - These documents are two revisions of the same initial document for which Tools → Changes → Activate Change Detection has been turned on.
  - Either a document has been opened in read-only mode or it does not need to be saved to disk (or Save As).

boolean compareRevisions(String `originalDocURI`, String `revisedDocURI`, boolean `verticalSplit`)
: Compare specified documents using the Compare tool.

  Note that the Compare tool is available only in the "`xxe`" applet because it's the only applet which can open multiple documents at the same time.

String checkValidity(String `docURI`)
: Checks the validity of specified document.

boolean closeDocument(String `docURI`, boolean `discardChanges`)
: Closes specified opened document.

String listDocuments()
: Returns the list of the URIs of currently opened documents.

String getActiveDocument()
: Returns the URI of the current ``active" document.

boolean setActiveDocument(String `docURI`)
: Changes the current ``active" document to specified opened document.

String getDocumentContents(String `docURI`, String `attachmentDir`)
: Returns the XML contents of specified opened document.

String listDocumentResources(String `docURI`, boolean `attachmentsOnly`)
: Returns the list of the URIs of the resources (typically graphic files) referenced in specified opened document.

String getDocumentAttachment(String `docURI`, String `attachmentURI`)
: Returns the contents of specified attachment encoded in base-64.

The reference manual of these methods is found in *Class Applet*.

### Example 6.5. The online demo of `editor1`.

This instance of `editor1` is made part of an HTML form called `DemoForm`:

```
<form method="POST" enctype="application/x-www-form-urlencoded"
      action="http://127.0.0.1:8080/applet_demo/echo"1
      name="DemoForm" onsubmit="return onSubmitForm();" target="_blank">
  <input type="hidden" name="documentURI" value="" />2
  <input type="hidden" name="xmlSource" value="" />
  <input type="hidden" name="attachmentCount" value="0" />

  ...

  <script type="text/javascript">
    var attributes = {name:'XXE', id:'XXE', mayscript:true,
                      code:'com.xmlmind.xmleditapp.applet.Applet',
                      width:'100%', height:'600'};
    var parameters = {jnlp_href:'xxe.jnlp',
                     appClass:'com.xmlmind.xmleditapp.applet.Editor1App',
                     separate_jvm:true, classloader_cache:false,
                     argument0:'-new',
                     argument1:'DocBook v5+/5.0',
                     argument2:'article',
                     argument3:'Untitled.xml'};3
    deployJava.runApplet(attributes, parameters, '1.6.0_10');
  </script>
</form>
```

1 The content of the form is posted to a Servlet called `applet_demo`.

In this demo, we use `enctype="application/x-www-form-urlencoded"`. In production, we would use `enctype="multipart/form-data"` (even if the form does not contain any file upload control) because this would be more efficient.

2 The `applet_demo` servlet expects the following query arguments:

`documentURI`
: Optional: the URI of the document.

`xmlSource`
: Required: contains the XML source of the document. The value of this field is a string which starts with `<?xml version="1.0" encoding="UTF-8"?>`.

`attachmentCount`
: Required: positive integer which specifies the number of `attachmentName`*`I`*/`attachmentDataI` field pairs. See below.

`attachmentName0-`*`N`*, `attachmentData0-`*`N`*
: Optional: `attachmentName`*`I`* contains the absolute "`file:`" URI of the attachment. `attachmentData`*`I`* contains the base-64 encoded data of the attachment.

3 Initially, the instance of `editor1` contains a new DocBook 5 article.

The URI of this new article is `Untitled.xml` resolved against the document base of the demo, which gives: `http://www.xmlmind.com/xmleditor/_applet/Untitled.xml`.

The JavaScript function `onSubmitForm()` is implemented as follows:

```
function onSubmitForm() {
    ...
    var demoForm = document.DemoForm;
    var xxe = document.XXE;

    demoForm.documentURI.value = xxe.getActiveDocument();1
    ...

    demoForm.xmlSource.value = xxe.getDocumentContents(null, null);2
    ...

    var attachments = xxe.listDocumentResources(null, true);3
    if (attachments != null) {
        var attachmentList = attachments.split("\n");
        var count = attachmentList.length;
        if (count > 0) {
            ...
            var j = 0;

            for (var i = 0; i < count; i += 2) {
                var name = attachmentList[i];

                var data = xxe.getDocumentAttachment(null, attachmentList[i]);4
                ...

                var jj = j.toString();

                var attachmentName = document.createElement("input");
                attachmentName.type = "hidden";
                attachmentName.name = "attachmentName" + jj;
                attachmentName.id = "attachmentName" + jj;
                attachmentName.value = name;
                var attachmentData = document.createElement("input");
                attachmentData.type = "hidden";
                attachmentData.name = "attachmentData" + jj;
                attachmentData.id = "attachmentData" + jj;
                attachmentData.value = data;

                demoForm.appendChild(attachmentName);
                demoForm.appendChild(attachmentData);

                ++j;
            }

            demoForm.attachmentCount.value = j;5
        }
    }

    return true;
}
```

1 Invoke `getActiveDocument()` to give a value to the `documentURI` `hidden` form field.

2 Invoke `getDocumentContents()` to give a value to the `xmlSource` `hidden` form field.

3 Invoke `listDocumentResources()` to list *attachments*.

What is called an attachment here is a document resource having an absolute "`file:`" URI.

When an author uses the applet ``normally" to reference a local resource, the URI of this resource is necessarily specified as an absolute "`file:`" URI. The reason for this fact is that a "`file:`" URI cannot be made relative to the URI of the edited document, which is an "`http:`" URI. The applet uses this specificity to automatically detect which resources files are to be attached to the posted XML source.

4 For each attachment detected by the applet, attempt to load it using `getDocumentAttachment()` and if this succeeds, add two `hidden` fields to the form. First field contains the absolute "`file:`" URI of the attachment. Second field contains the base-64 encoded contents of the attachment.

5 Give a value to the `attachmentCount` `hidden` form field.

# Part II. Reference

# Table of Contents

# Chapter 7. Configuration elements

Configuration elements are directives which are executed by XXE

- during its start-up (help [80], include [83], translation [112], template [104]);
- or when loading a document (detect [71] elements of all configurations are tried in turn in an attempt to recognize the type of the document);
- or just after loading a document which has been associated to a configuration because the `detect` element of this configuration has recognized it (all other elements: binding [61], css [70], etc).

## 1. attributeEditor

```
<attributeEditor
  attribute = Name
  elementMatches = XPath pattern
>
  Content: [ class [ property ]* ]? |
           [ list ]?
</attributeEditor>

<class>
  Content: Java class name
</class>

<property
  name = NMTOKEN matching [_a-zA-Z][_a-zA-Z0-9]*
  type = (boolean|byte|char|short|int|long|float|double|
          String)
  value = string
/>

<list
  allowAnyValue = boolean : false
  allowWhitespace = boolean : dynamic
  allowMultipleValues = boolean : false
  selectItems = XPath expression
  itemValue = XPath expression
  itemDescription = XPath expression
>
  Content: [ item ]*
</list>

<item
  description = Non empty token
>
  Content: Non empty string
</item>
```

The `attributeEditor` configuration element allows to extend the Attributes tool. There are two kinds of such extensions:

1. An extension which returns the list of all possible values for a given attribute. Example:

```
<attributeEditor attribute="f:remove" elementMatches="f:filter"
  xmlns:f="urn:namespace:filter">
  <list>
    <item>red</item>
    <item>green</item>
    <item>blue</item>
  </list>
</attributeEditor>
```

2. An extension which creates a modal dialog box allowing to edit the value of a given attribute. This dialog box is passed the initial attribute value (or the empty string if the attribute has not yet been specified). The dialog box is then expected to return a possibly modified value for this attribute. XHTML example:

   ```
   <attributeEditor attribute="bgcolor"
     elementMatches="html:table|html:tr|html:th|html:td|html:body"
     xmlns:html="http://www.w3.org/1999/xhtml">
     <class>HexColorChooser</class>
   </attributeEditor>
   ```

These extensions are used by the Attributes tool as follows:

1. The Value field which supports auto-completion will display the items of the list.

2. When you click the Edit button or right-click on an attribute, this displays a popup menu. The first entry of this menu is also called Edit and displays a dialog box allowing to edit the attribute more comfortably than with the Value field. The dialog box displayed in this case comes from the `attributeEditor` configuration element.

Note that when an extension returns a list, a specialized dialog box may be automatically wrapped around this list. That is, when an extension returns a list, not only the Value field will provide auto-completion for the attributes values, but also the Edit popup menu item will display a specialized dialog box.

The attributes of the `attributeEditor` configuration element are used to detect attributes for which a custom editor is to be created:

`attribute`
: The XML qualified name of the attribute.

`elementMatches`
: An XPath pattern matching the elements possibly having the attribute whose name is specified by above attribute `attribute`.

Note that an `attributeEditor` is uniquely identified by its `attribute` and `elementMatches` attributes and also by the name of the configuration containing it. For example, the following `attributeEditors` do not conflict provided that they are defined in different configurations:

```
<attributeEditor attribute="ref" elementMatches="*">
  <list selectItems="//part/@number" />
</attributeEditor>
```

```
<attributeEditor attribute="ref" elementMatches="*">
  <list>
    <item>internal</items>
    <item>external</items>
  </list>
</attributeEditor>
```

The child elements the `attributeEditor` configuration element are used to specify how the custom editor is to be implemented by the Attributes tool:

`class`
: This element contains the fully qualified name of a class which implements one or both of the following interfaces: `com.xmlmind.xmledit.cmd.attribute.SetAttribute.ChoicesFactory`, `com.xmlmind.xmledit.cmd.attribute.SetAttribute.EditorFactory`.

  The `property` child elements of the class element allow to parameterize the newly created instance of this class. See bean properties [76].

  DocBook example:

  ```
  <attributeEditor attribute="linkend" elementMatches="xref|link">
    <class>com.xmlmind.xmleditapp.linktype.RefChoicesFactory</class>
  ```

```
  <property name="listIfMemberOfDocSet" type="boolean" value="true" />
</attributeEditor>
```

`list`

This element specifies all possible values for a given attribute. The items of this list may be statically described by the means of the `item` child element or dynamically computed by the means of the `selectItems`, `itemValue` and `itemDescription` XPath expressions.

## Static lists

A static list comprises only the items specified by its `item` child elements. The string contained in the `item` element specifies the value of the item. The optional `description` attribute provides a description of this value.

Items are automatically sorted by their values. Duplicate items are automatically removed.

DITA example:

```
<attributeEditor attribute="audience" elementMatches="*">
  <list allowMultipleValues="true">
    <item description="A user of the product">user</item>
    <item description="A product purchaser">purchaser</item>
    <item description="A product administrator">administrator</item>
    <item description="A programmer">programmer</item>
    <item description="An executive">executive</item>
    <item description="Someone who provides services
                       related to the product">services</item>
  </list>
</attributeEditor>
```

## Dynamic lists

Unless a list has `item` child elements, specifying at least attribute `selectItems` is mandatory.

`selectItems`

Returns a node set enumerating all list items. This XPath expression is evaluated in the context of the element having the attribute being edited by the Attributes tool.

`itemValue`

This XPath expression is evaluated in the context of each node returned by `selectItems`. It returns a string which is the value of the item. Items having an empty value are discarded.

When this attribute is missing, the value of an item is the string value of the node selected by `selectItems`.

`itemDescription`

This XPath expression is evaluated in the context of each node returned by `selectItems`. It returns a string which is the description of the item. Empty descriptions are ignored.

When this attribute is missing, an item has no description.

Items are automatically sorted by their values. Duplicate items are automatically removed.

XHTML example:

```
<attributeEditor attribute="for" elementMatches="html:label">
  <list selectItems="//html:input|//html:select" itemValue="@id"
        itemDescription="concat(local-name(.), ' ', @type)"
        allowWhitespace="false" />
</attributeEditor>
```

### Tip

A convenient way to describe an element is to use XPath extension function `sa:getElementDescription(nodeset_returning_an_element)`, where prefix "sa" is bound to namespace "`java:com.xmlmind.xmledit.cmd.attribute.SetAttribute`".

For example, the above XHTML example could be rewritten as:

```
<attributeEditor attribute="for" elementMatches="html:label">
  <list selectItems="//html:input|//html:select" itemValue="@id"
        itemDescription="sa:getElementDescription(.)"
        xmlns:sa="java:com.xmlmind.xmledit.cmd.attribute.SetAttribute"
        allowWhitespace="false" />
</attributeEditor>
```

### Other `list` attributes

allowAnyValue
: Allow the user to specify values other than the ones coming from the list.

allowWhitespace
: List items may have values containing whitespace. When the list is static, the default value of this attribute is determined by examining all the items of the list. When the list is dynamic, the default value of this attribute is `true`.

allowMultipleValues
: The value of the attribute may contain one or more tokens (coming from the values of the list items) separated by whitespace.

Remember that a custom attribute editor specified using `attributeEditor` is just here to help the user specify an attribute value. It's not really designed to *validate* what the user specifies. It's up to the underlying DTD or schema to perform this validation task.

An `attributeEditor` element without any child element may be used to remove from a configuration a previously defined `attributeEditor` having the same `attribute` and `elementMatches` attributes.

## 2. binding

```
<binding>
  Content: [ mousePressed | mouseDragged | mouseReleased |
             mouseClicked | mouseClicked2 | mouseClicked3 |
             [ keyPressed | charTyped ]{1,3} |
             appEvent ]
           [ command | menu ]?
</binding>

<mousePressed
  button = (1|2|3|popupTrigger[1]) : 1
  modifiers = possibly empty list of (ctrl|shift|alt|meta|altGr|mod[2])
/>

<mouseDragged
  button = (1|2|3|popupTrigger) : 1
  modifiers = possibly empty list of (ctrl|shift|alt|meta|altGr|mod)
/>

<mouseReleased
  button = (1|2|3|popupTrigger) : 1
  modifiers = possibly empty list of (ctrl|shift|alt|meta|altGr|mod)
/>

<mouseClicked
  button = (1|2|3|popupTrigger) : 1
  modifiers = possibly empty list of (ctrl|shift|alt|meta|altGr|mod)
```

```
/>

<mouseClicked2
  button = (1|2|3|popupTrigger) : 1
  modifiers = possibly empty list of (ctrl|shift|alt|meta|altGr|mod)
/>

<mouseClicked3
  button = (1|2|3|popupTrigger) : 1
  modifiers = possibly empty list of (ctrl|shift|alt|meta|altGr|mod)
/>
```

Note that:

1. `popupTrigger` is a shorthand for mouse-pressed-3, no matter the modifiers or the number of clicks. On the Mac, it is additionally a shorthand for **Ctrl**+mouse-pressed-1.
2. `mod` is the Command key on Mac and the Control key on other platforms.

```
<keyPressed
  code = key code
  modifiers = possibly empty list of (ctrl|shift|alt|meta|altGr|mod)
/>

<charTyped
  char = single character
/>

<appEvent
  name = name of application event
/>

<command
  name = NMTOKEN (optionally preceded by a command namespace [68])
  parameter = string
/>

<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 [68])
  parameter = string
/>

key code = (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
            9 | A | ACCEPT | ADD | AGAIN |
            ALL_CANDIDATES | ALPHANUMERIC | AMPERSAND |
            ASTERISK | AT | B | BACK_QUOTE | BACK_SLASH |
            BACK_SPACE | BEGIN | BRACELEFT | BRACERIGHT | C |
            CANCEL | CAPS_LOCK | CIRCUMFLEX | CLEAR |
            CLOSE_BRACKET | CODE_INPUT | COLON | COMMA | COMPOSE |
            CONTEXT_MENU | CONVERT | COPY | CUT | D | DEAD_ABOVEDOT |
            DEAD_ABOVERING | DEAD_ACUTE | DEAD_BREVE |
            DEAD_CARON | DEAD_CEDILLA | DEAD_CIRCUMFLEX |
            DEAD_DIAERESIS | DEAD_DOUBLEACUTE | DEAD_GRAVE |
            DEAD_IOTA | DEAD_MACRON | DEAD_OGONEK |
            DEAD_SEMIVOICED_SOUND | DEAD_TILDE |
            DEAD_VOICED_SOUND | DECIMAL | DELETE |
            DIVIDE | DOLLAR | DOWN | E | END | ENTER |
            EQUALS | ESCAPE | EURO_SIGN | EXCLAMATION_MARK |
            F | F1 | F10 | F11 | F12 | F13 | F14 | F15 | F16 | F17 |
            F18 | F19 | F2 | F20 | F21 | F22 | F23 | F24 | F3 | F4 |
            F5 | F6 | F7 | F8 | F9 | FINAL | FIND | FULL_WIDTH |
            G | GREATER | H | HALF_WIDTH | HELP | HIRAGANA |
            HOME | I | INPUT_METHOD_ON_OFF | INSERT |
            INVERTED_EXCLAMATION_MARK | J | JAPANESE_HIRAGANA |
            JAPANESE_KATAKANA | JAPANESE_ROMAN | K | KANA |
            KANA_LOCK | KANJI | KATAKANA | KP_DOWN | KP_LEFT |
            KP_RIGHT | KP_UP | L | LEFT | LEFT_PARENTHESIS |
            LESS | M | MINUS | MODECHANGE | MULTIPLY | N |
            NONCONVERT | NUMBER_SIGN | NUMPAD0 | NUMPAD1 |
            NUMPAD2 | NUMPAD3 | NUMPAD4 | NUMPAD5 | NUMPAD6 |
            NUMPAD7 | NUMPAD8 | NUMPAD9 | NUM_LOCK | O |
            OPEN_BRACKET | P | PAGE_DOWN | PAGE_UP | PASTE |
            PAUSE | PERIOD | PLUS | PREVIOUS_CANDIDATE |
            PRINTSCREEN | PROPS | Q | QUOTE | QUOTEDBL | R |
            RIGHT | RIGHT_PARENTHESIS | ROMAN_CHARACTERS |
            S | SCROLL_LOCK | SEMICOLON | SEPARATOR | SLASH |
            SPACE | STOP | SUBTRACT | T | TAB | U | UNDERSCORE |
            UNDO | UP | V | W | WINDOWS | X | Y | Z)
```

Bind a key stroke to a command or bind a mouse click to a command or a popup menu or bind an application event [64] to a command.

Note that a key stroke or an application event cannot be used to display a popup menu.

A `binding` element not containing a `command` or `menu` child element may be used to remove the corresponding keyboard shortcut or mouse click.

XXE does not allow to replace any of its default bindings, just to add more bindings, unless these bindings are specified in a special purpose configuration file called `customize.xxe`. For more information about `customize.xxe`, see Generic bindings [25].

Examples: bind **F4** to command "`insert into tt`":

```
<binding>
  <keyPressed code="F4" />
  <command name="insert" parameter="into tt" />
</binding>
```

Bind **Esc @** to command "`insert into a`":

```
<binding>
  <keyPressed code="ESCAPE" />
  <charTyped char="@" />
  <command name="insert" parameter="into a" />
</binding>
```

Unbind the command bound to **Ctrl**+**A** (Command+**A** on the Mac):

```
<binding>
  <keyPressed code="A" modifiers="mod" />
</binding>
```

Bind **Ctrl**+Shift+mouse-pressed-3 to a “convert case” popup menu:

```
<binding>
  <mousePressed button="3" modifiers="shift ctrl" />
  <menu>
    <item label="Lower-case" command="convertCase" parameter="lower"/>
    <item label="Upper-case" command="convertCase" parameter="upper"/>
    <item label="Capital-case" command="convertCase" parameter="capital"/>
  </menu>
</binding>
```

**About application events**

An *application event*, like a mouse click or a keystroke, is used to trigger an action. But unlike user inputs, application events are not generated by the graphics system (i.e. Java™ AWT). Application events are directly created and dispatched to the document view by XXE.

Application events have been created to be able to use the very useful binding mechanism for events other than mouse clicks or keystrokes. For example: drag and drop, changes of the editing context, document events, etc.

Currently XXE generates the following application events:

drag

Generated when the user drags something other than an `drag-source` (see Section 13, “drag-source” in *XMLmind XML Editor - Support of Cascading Style Sheets (W3C CSS)*) in the document view.

### Important

Dragging an object in the document view means: dragging the mouse over the object while keeping the left button *and the **Alt** key* pressed.

The command bound to this application event must return a *string*. This string will be passed as is to the drop target.

By default, XXE uses the following binding:

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

DITA example: a example of a contextual drag command:

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

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

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

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

drop

Generated when the user drops a string in the document view.

If the object dropped from an external application is not a string, this object will be automatically converted to a string. For example, a file is converted to a string by using its absolute filename.

In addition to `%{value}`, which is substituted with the dropped string, the following convenience variables are also supported:

%{url}

If `%{value}` contains an URL or the absolute filename of a file or a directory, this variable contains the corresponding URL.

%{file}

If `%{value}` contains a "`file:`" URL or the absolute filename of a file or a directory, this variable contains the corresponding filename.

By default, XXE uses the following binding: (notice how the string is passed to the `drop` command):

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

DocBook example: a contextual drop command:

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

<command name="docb.drop">
  <macro>
    <choice>
      <sequence>
        <match context="$clickedElement" pattern="ulink|ulink//*" />
        <set variable="selectedElement" context="$clickedElement"
             expression="(ancestor-or-self::ulink)[last()]" />

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

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

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

Interactively resize an image by dragging one of the “handles” displayed around it

The following 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>
```

Notice that both the above examples use the same, generic, command resizeImage in *XMLmind XML Editor - Commands*.

Interactively resize a table column by dragging its column separator

The name of the corresponding application event is `resize-table-column`. Binding this application event to a command allows to have one or more of the following variables substituted in the parameter of the bound command:

`%{resizedColumn}`
: The index of the leftmost resized column. The index of the first column of a table is 0.

`%{columnCount}`
: The number of columns of the table containing the column being resized.

`%{oldColumnWidths}`
: The widths of the columns of the table before the column has been resized. A column width is expressed in pixels. Column widths are separated by space characters.

`%{newColumnWidths}`
: The widths of the columns of the table after the column has been resized.

XHTML example:

```
<binding>
  <appEvent name="resize-table-column" />
  <command name="xhtml.resizeTableColumn"
           parameter="%{resizedColumn} %{columnCount}
                      %{oldColumnWidths} %{newColumnWidths}" />
</binding>

<command name="xhtml.resizeTableColumn">
  <class>com.xmlmind.xmleditext.xhtml.table.ResizeTableColumn</class>
</command>
```

DocBook 5 example:

```
<binding>
  <appEvent name="resize-table-column" />
  <command name="db5.resizeTableColumn"
           parameter="%{resizedColumn} %{columnCount}
                      %{oldColumnWidths} %{newColumnWidths}" />
</binding>

<command name="db5.resizeTableColumn">
  <macro>
    <choice>
      <!-- tgroup is selected -->
      <command name="db5.resizeCALSTableColumn" parameter="%*" />

      <!-- table or informaltable is selected -->
      <command name="db5.resizeHTMLTableColumn" parameter="%*" />
    </choice>
  </macro>
</command>

<command name="db5.resizeCALSTableColumn">
  <class>com.xmlmind.xmleditext.docbook.table.ResizeTableColumn</class>
</command>

<command name="db5.resizeHTMLTableColumn">
  <class>com.xmlmind.xmleditext.xhtml.table.ResizeTableColumn</class>
</command>
```

There is no generic command which, after a proper parameterization, would allow to resize the columns of all kinds of tables. However, as shown in the above examples, you can apply `com.xmlmind.xmleditext.xhtml.table.ResizeTableColumn` to HTML tables and `com.xmlmind.xmleditext.docbook.table.ResizeTableColumn` to DocBook (CALS) tables.

## 3. command

```
<command
  name = NMTOKEN (optionally preceded by a command namespace [68])
>
  Content: class | menu | macro | process
</command>

<class>
  Content: Java class name
</class>
```

Register command specified by `class`, `macro` or `process` with XXE. The newly registered command can be referenced in binding [61] `command` or `menu`, menu [90] `item`, toolBar [105] `item` and command [68] `macro` using name *`name`*.

Example:

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

In the above example, custom command `com.xmlmind.xmleditext.xhtml.Preview` written in Java™ is registered by XXE under the name `xhtml.preview`.

Child elements of `command`:

class
: Register command implemented in the Java™ language by class *`class`* (implements interface `com.xmlmind.xmledit.gadget.Command` -- See Chapter 4, *Writing a command* in *XMLmind XML Editor - Developer's Guide*).

menu
: Define a popup menu of commands. This special type of command, typically invoked from contextual macro-commands, is intended to be used to specify contextual popup menus, redefining or extending the standard right-click popup menu. See Chapter 3, *Menu commands* in *XMLmind XML Editor - Commands*.

macro
: Define a macro-command which is, to make it simple, a sequence of native commands, menu commands, process commands or other macro-commands. See Chapter 4, *Macro commands* in *XMLmind XML Editor - Commands*.

process
: Define a process command, which is an arbitrarily complex transformation of part or all of the document being edited. See Chapter 5, *Process commands* in *XMLmind XML Editor - Commands*.

### 3.1. About command names

The name of a command is basically an `NMTOKEN`. To make it simple, this means that a command name may contain letters, digits, a few punctuation characters such as `'_'`, `'-'` and `'.'`, but no space characters.

All the commands are registered by their names in the same global registry. In practice, this means that if configuration *`A`* defines a command called `doIt`, then configuration *`B`* has access to this command. This also means that if configuration *`C`* also defines a command called `doIt`, then this command may overwrite the one defined in configuration *`A`*[1].

A simple way to prevent this kind of name conflict is to use a *prefix* (not related to XML namespace prefixes) for the name of your commands. Example of commands written by XMLmind: `docb.promote`, `docb.demote`, `xhtml.preview`. (We always use *`short_lower_case_prefix.camelCaseCommandName`*.)

[1]Or the opposite way around: `doIt` defined in configuration *`A`* overwrites `doIt` defined in configuration *`C`*. This depends on the order of configuration loading by XXE.

However, in some cases, the commands you are writing are really private to your configuration. Example: a helper macro-command invoked by another macro-command. In such cases, you'll not want anyone to be able to access these commands. In particular, you don't want the end-user to execute these ancillary commands by using Tools → Execute Command or Options → Customize configuration → Add Keyboard Shortcut.

When you'll want your command to be really hidden from the end-users, you may consider giving it a name having a namespace (not related to XML namespaces). Example: in "`{My Config}doit`", the namespace is "`My Config`" and the local name is "`doIt`".

A command namespace may contain any character except `'}'`. A command local name is an `NMTOKEN`. The command namespace is of course optional.

In many cases, you'll want your command namespace to contain the name of your configuration. When this is the case, you may reference the "`$c`" pseudo-variable anywhere in your command namespace. Examples: "`{$c}doIt`", "`{$c helper}selectItFirst`"

Command namespaces also play a fundamental role in defining or extending the right-click, contextual, menu in *XMLmind XML Editor - Commands*.

## 4. configuration

```
<configuration
  name = non empty token
  mimeType = non empty token
  icon = anyURI
  extensions = a non empty list of filename extensions
>
  Content: [ attributeEditor|binding|command|css|detect|documentResources|
             dtd|elementTemplate|help|imageToolkit|include|inclusionScheme|
             linkType|menu|newElementContent|parameterGroup|parameterSet|
             preserveSpace|property|relaxng|saveOptions|schema|schematron|
             spellCheckOptions|spreadsheetFunctions|template|toolBar|translation|
             validate|validateHook|windowLayout ]*
</configuration>
```

This root element of a XXE configuration is just a container for all the other configuration elements. See Writing a configuration file for XXE [5].

Attributes:

name
: This attribute uniquely identifies the configuration. This attribute is required in top-level configurations (e.g. `docbook.xxe`). On the other hand, it must not be specified in configuration modules (e.g. `common.incl`).

mimeType
: The value of this attribute is used to specify the content type of XML documents saved on WebDAV servers. When this attribute is not specified, the content type passed to the WebDAV server is always `application/xml`. This attribute allows to be more specific: `application/xhtml+xml`, `application/docbook+xml`, etc.

icon
: Specifies an icon which may be used to differentiate this kind of document. The format of this icon is expected to be GIF, PNG or JPEG and its size is expected to be 16x16.

extensions
: Specifies one or more filename extensions commonly used for this kind of document. Multiple filename extensions must be separated by whitespace. It is not useful to include "`xml`" in this list.

  A filename extension must match the "`[_a-zA-Z0-9]+`" regular expression pattern. Notice that a filename extension cannot have a leading dot. That is, specify "`foo`" and not "`.foo`".

Example:

```
<configuration name="Example1"
  xmlns="http://www.xmlmind.com/xmleditor/schema/configuration">

  <detect>
    <dtdPublicId>-//XMLmind//DTD Example1//EN</dtdPublicId>
  </detect>

  <css name="Style sheet" location="example1.css" />

  <template name="Template" location="example1.xml" />

</configuration>
```

The structure of the configuration element is loose: you can add any number of any of its child elements in any order.

This loose structure is very convenient when you need to create a new configuration which just adds or replaces a few elements to an existing configuration.

Example: The following configuration called `DocBook` overrides bundled configuration also called `DocBook`.

```
<configuration name="DocBook" mimeType="application/docbook+xml"
  icon="../common/mime_types/docbook.png" extensions="dbk"
  xmlns="http://www.xmlmind.com/xmleditor/schema/configuration">

  <include location="docbook-config:docbook.xxe" />

  <css name="DocBook" location="MyDocBook.css" />
  <css name="Big Fonts" location="MyDocBook_BigFonts.css" />

  <template name="Chapter" />
  <template name="Section" />

  <binding>
    <keyPressed code="F5" modifiers="mod shift" />
    <command name="insert" parameter="into literal" />
  </binding>

</configuration>
```

The configuration in previous example can be described as follows:

- It includes the stock DocBook configuration from `docbook-config:docbook.xxe` to reuse its `detect`, `elementTemplate`, `toolBar`, etc, elements. ("`docbook-config:`" resolves to the directory containing the "`DocBook`" configuration, by default it's *`XXE_install_dir`*`/addon/config/docbook/`.)
- It replaces bundled style sheet named `DocBook` by another one contained in `MyDocBook.css`. It adds another style sheet called `Big Fonts`.
- It discards document templates named "`Chapter`" and "`Section`" (template [104] with no `location` attribute).
- Its binds key stroke **Shift**+**Ctrl**+**F5** command "`insert into literal`". (`mod` is the Command key on Mac and the Control key on other platforms).

## 5. css

```
<css
  name = non empty token
  location = anyURI
  alternate = boolean : false
/>
```

Add CSS style sheet named *`name`*, contained in file *`location`*, to the Style menu.

Any style sheet with `alternate="false"` is used preferably to a style sheet with `alternate="true"` to render a newly opened document.

Note that if a document contains `<?xml-stylesheet type="text/css"?>` processing instructions, by default (there is an XXE option to specify this) the style sheets specified this way are used and the style sheets specified in the configuration file are ignored.

Specifying a `css` element without a location may be used to remove `css` element with the same name from the configuration.

Example:

```
<css name="XHTML" location="css/xhtml-form.css" />
<css name="XHTML (form elements not styled)"
     location="css/xhtml.css" alternate="true" />
```

Special attribute value `name="-"` may be used to instruct XXE to initially display the opened document as a tree view. Example of the configuration allowing to edit W3C XML Schemas in XXE:

```
<css location="" name="-" />
<css location="wxs.css" name="W3C XML Schema"
     alternate="true" />
```

Notice that, when `name="-"`, the value of the `location` attribute is ignored, therefore suffice to specify `location=""`.

## 6. DTD

```
<dtd
  systemId = anyURI
  publicId = non empty token
/>
```

Use the DTD specified by this element to constrain the document.

Note that

- if a document contains a document type declaration (<!DOCTYPE>) which defines elements,
- or if the root element of a document has `xsi:schemaLocation`/`xsi:noNamespaceSchemaLocation` attributes,
- of if a document contains a `<?xml-model href="..."?>`,

the grammar specified this way is used and the DTD specified in the configuration file is ignored.

Example:

```
  <dtd publicId="-//W3C//DTD XHTML 1.0 Strict//EN"
       systemId="dtd/xhtml1-strict.dtd" />
```

### Caution

When using this configuration, also specify `<saveOptions [97] saveCharsAsEntityRefs="false">`. Otherwise, if the added DTD specifies character entities as it is often the case, you may end up creating documents which cannot be interchanged with other applications. The other applications would see such DTD-less documents containing references to named character entities as being non-well formed.

## 7. detect

```
<detect>
  Content: and|dtdPublicId|dtdSystemId|fileNameExtension|mimeType|
           not|or|rootElementLocalName|rootElementNamespace|
           rootElementAttribute|schemaType
</detect>
```

```
<and>
  Content: [ and|dtdPublicId|dtdSystemId|fileNameExtension|mimeType|
             not|or|rootElementLocalName|rootElementNamespace|
             rootElementAttribute|schemaType ]+
</and>

<dtdPublicId
  substring = boolean : false
>
  Content: non empty token
</dtdPublicId>

<dtdSystemId>
  Content: anyURI
</dtdSystemId>

<fileNameExtension>
  Content: file name extension
</fileNameExtension>

<mimeType>
  Content: non empty token
</mimeType>

<not>
  Content: and|dtdPublicId|dtdSystemId|fileNameExtension|mimeType|
           not|or|rootElementLocalName|rootElementNamespace|
           rootElementAttribute|schemaType
</not>

<or>
  Content: [ and|dtdPublicId|dtdSystemId|fileNameExtension|mimeType|
             not|or|rootElementLocalName|rootElementNamespace|
             rootElementAttribute|schemaType ]+
</or>

<rootElementLocalName>
  Content: Name
</rootElementLocalName>

<rootElementNamespace>
  Content: anyURI
</rootElementNamespace>

<rootElementAttribute
  localName = Name
  namespace = anyURI
  value = string
  substring = boolean : false
/>

<schemaType>
  Content: 'dtd' | 'schema' | 'relaxng'
</schemaType>
```

Register with XXE a condition which can be used to detect the type of a document.

During its start-up, XXE loads all the configuration files it can find, because it needs to keep a list of all `detect` elements.

The order of a `detect` element in this list depend on the location of its configuration file: configurations loaded from the `config` subdirectory of user preferences directory precede configurations loaded from the value of environment variable `XXE_ADDON_PATH` which in turn precede configurations loaded from the `addon` subdirectory of XXE distribution directory.

When a document is opened, XXE tries each `detect` element in turn. If the condition expressed in the `detect` element evaluates to true, the detection phase stops and the configuration containing the `detect` element is associated to the newly opened document.

Child elements of `detect`:

and
: Evaluates to true if all its children evaluate to true.

dtdPublicId
: Evaluates to true if the document has a document type declaration (`<!DOCTYPE>`) with a public ID equals to the content of this element.

  If `substring="true"`, evaluates to true if public ID contains the specified string.

dtdSystemId
: Evaluates to true if the document has a document type declaration (`<!DOCTYPE>`) with a system ID equals to the content of this element.

fileNameExtension
: Evaluates to true if the file containing the document has a name which ends with `'.'` followed by the content of this element.

mimeType
: Evaluates to true if the file containing the document has a MIME type equals to the content of this element.

not
: Evaluates to true if its child evaluates to false.

or
: Evaluates to true if any of its children evaluates to true.

rootElementLocalName
: Evaluates to true if the document has a root element with a local name (name without the namespace part) equals to the content of this element.

rootElementNamespace
: Evaluates to true if the document has a root element with a name which belongs to the namespace equals to the content of this element.

  Use "`<rootElementNamespace xsi:nil='true' />`" to specify that the name of root element has no namespace.

rootElementAttribute
: Evaluates to true if the document has a root element which has at least one attribute where *all* of the following is true:

  - The local part of the name of the attribute is equal to the value of `localName`. When `localName` is not specified, any local part will do.
  - The namespace URI of the name of the attribute is equal to the value of `namespace`. When `namespace` is not specified, any namespace URI or no namespace URI at all will do.

    Use the empty string (e.g. `namespace=""`) to specify that the name of the attribute should have no namespace at all.
  - The value of the attribute must be equal to the value of `value`. When `value` is not specified, any `value` will do.

    If `substring` is specified with value `true`, suffice for the value of the attribute to contain the value of `value`.

DocBook 5 example: use a specific configuration for documents conforming to version 1.0 of Acme Corporation's extension of DocBook 5. As explained in the DocBook 5 documentation, the root element of such document should have a `version` attribute with value `5.0-extension acme-1.0`.

```
<rootElementAttribute localName="version" value="acme" substring="true" />
```

What follows is even more precise, though not strictly needed:

```
<rootElementAttribute localName="version" namespace="" value="acme" substring="true" />
```

schemaType
: Evaluates to true

- if the document is explicitly constrained by a DTD (that is, has a `<!DOCTYPE>`) and the content of this element is `DTD`,
- OR if he document is explicitly constrained by an W3C XML Schema (that is, has a `xsi:schemaLocation` or a `xsi:noNamespaceSchemaLocation` attribute on its root element) and the content of this element is `schema`.
- OR if he document is explicitly constrained by RELAX NG schema (that is, contains a `<?xml-model href="..."?>` pointing to a RELAX NG schema) and the content of this element is `relaxng`.

Use "`<schemaType xsi:nil='true' />`" to specify that document is not explicitly constrained by a DTD, a W3C XML Schema or a RELAX NG schema.

Example:

```
<detect>
  <and>
    <or>
      <rootElementLocalName>book</rootElementLocalName>
      <rootElementLocalName>article</rootElementLocalName>
      <rootElementLocalName>chapter</rootElementLocalName>
      <rootElementLocalName>section</rootElementLocalName>
      <rootElementLocalName>sect1</rootElementLocalName>
      <rootElementLocalName>sect2</rootElementLocalName>
      <rootElementLocalName>sect3</rootElementLocalName>
      <dtdPublicId substring="true">DTD DocBook XML</dtdPublicId>
    </or>
    <rootElementNamespace xsi:nil="true" />
    <not>
      <dtdPublicId substring="true">Simplified</dtdPublicId>
    </not>
  </and>
</detect>
```

The `detect` element in this example can be described as follows: opened document is a DocBook document if

- The local name of the root element is one of `book`, `article`, `chapter`, `section`, `sect1`, sect2, `sect3`.

  OR the public ID of its DTD contains string "`DTD DocBook XML`".
- AND the name of its root element does not belong to any namespace.
- AND the public ID of its DTD does not contain string "`Simplified`".

## 8. documentResources

```
<documentResources>
  Content: [ resource|selector ]+
</documentResources>

<resource>
  path = Absolute XPath (subset [75])
  kind = NMTOKEN
/>

<selector>
  <class>Content: Java class name</class>
</selector>
```

Specifies which resources are logically part of the document being edited. Generally these resources are external image files.

Attributes of child element `resource`:

`path`
: XPath expression used to find the URIs of the resources within the document content. These URIs are generally attribute values but could also be element values.

`kind`
: Specifies the kind of resources (`image`, `video`, `audio`, etc) selected by the XPath expression.

  The value of this attribute may be referenced in the `include` or `exclude` attributes of element `process/copyDocument/resources` in *XMLmind XML Editor - Commands*. This allows to specify whether a document resource should be ignored, copied or converted, depending on the kind of this resource.

In complex cases, specifying document resources using simple XPath expressions (see XPath subset [75] below) is not sufficient. In such case, use `selector` child elements instead of `resources`. The `class` element contains the name of a Java™ class which implements `com.xmlmind.xml.save.ResourceSelector`.

DITA example:

```
<cfg:documentResources xmlns="">
  <cfg:resource path="//image/@href"/>
  <cfg:resource path="//coderef/@href"/>
</cfg:documentResources>
```

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>
```

**XPath 1.0 subset supported by configuration elements**

The XPath 1.0 subset supported by configuration elements is the one defined in "XML Schema Part 1: Structures, Identity-constraint Definitions", except that absolute XPaths (/foo/bar, //bar, etc) are also supported.

```
XPath      ::=    Path ( '|' Path )*
Path       ::=    ('/'|'//')? ( Step ('/'|'//') )* ( Step | '@' NameTest )
Step       ::=    '.' | NameTest
NameTest   ::=    QName | '*' | NCName ':' '*'
```

Both abbreviated syntax and non-abbreviated syntax are supported.

## 9. documentSetFactory

```
<documentSetFactory>
  Content: [ class [ property ]* ]?
</documentSetFactory>

<class>
```

```
  Content: Java class name
</class>

<property
  name = NMTOKEN matching [_a-zA-Z][_a-zA-Z0-9]*
  type = (boolean|byte|char|short|int|long|float|double|
          String|URL)
  value = string
/>
```

Creates a *document set* factory and registers it with XMLmind XML Editor. More information about document sets in Section 8.1, “What is a document set?” in *XMLmind XML Editor - Online Help*.

Child elements of `documentSetFactory`:

class
: The fully qualified name of a Java™ class implementing interface `com.xmlmind.xmleditapp.docset.DocumentSetFactory`.

property
: `Property` child elements may be used to parametrize the newly created factory See bean properties [76].

DocBook v5+ example:

```
<documentSetFactory>
  <class>com.xmlmind.xmleditapp.docset.modulardoc.ModularDocumentFactory</class>
  <property name="styleSheetURL" type="URL" value="css/toc.css" />
</documentSetFactory>
```

Note that class `com.xmlmind.xmleditapp.docset.modulardoc.ModularDocumentFactory` is not specific to DocBook v5+. It may be used for any kind of modular document which makes use of inclusion schemes [84] supported by XMLmind XML Editor.

However the CSS stylesheet (in the above example, it's "`css/toc.css`") used to render the XML representation of the document set[2] is specific to each document type.

## 9.1. Bean properties

Some of the class instances created by the means of the `class` element may be parameterized using `property` child elements. A property child element specifies a *Bean* (that is, a Java™ Object) property.

Example:

```
<property name="columns" type="int" value="40" />
```

implies that the bean to be parametrized has a public method which resembles:

```
setColumns(int number)
```

Such properties are completely specific to the bean they parametrize and therefore, cannot be described in this section.

| **type** | Corresponding Java™ type | Syntax of `value` | Example |
|---|---|---|---|
| boolean | `boolean` | true, false | `true` |
| byte | `byte` | integer: -128 to 127 inclusive | `100` |
| char | `char` | a single character | `a` |

[2]In the case of a modular document, the XML representation of the document set is the modular document itself.

| `type` | Corresponding Java™ type | Syntax of `value` | Example |
|---|---|---|---|
| short | `short` | integer: -32768 to 32767 inclusive | `1000` |
| int | `int` | integer: -2147483648 to 2147483647 inclusive | `-1` |
| long | `long` | integer: -9223372036854775808 to 9223372036854775807, inclusive | `255` |
| float | `float` | single-precision 32-bit format IEEE 754 | `-0.5` |
| double | `double` | double-precision 64-bit format IEEE 754 | `1.0` |
| String | `java.lang.String` | A string | `Hello, world!` |
| URL | `java.net.URL` | An absolute or relative URI.<br><br>A relative URI is relative to the URI of the file containing the configuration element. | `css/toc.css` |

Example actually used in the XHTML configuration:

```
<validateHook name="checkLinks">
  <class>com.xmlmind.xmleditapp.linktype.LinkChecker</class>
  <property name="checkAnchors" type="boolean" value="false" />
  <property name="checkRefs" type="boolean" value="false" />
</validateHook>
```

## 10. elementTemplate

```
<elementTemplate
  name = NMTOKEN
  parent = XPath (subset [75])
  selectable = (false|true|override) : true
>
  Content: [ any element ]?
</elementTemplate>
```

Register with XXE the element template specified in this element.

An element template can include another element template. This is specified by `<`*included_element_name* `cfg:template="`*included_template_name*`"/>` inside the body of the template. See DocBook example below.

Note that the validity of the element contained in the `elementTemplate` is not checked by XXE when the configuration file is parsed.

Specifying a `elementTemplate` containing no element may be used to remove all `elementTemplates` with the same name from the configuration.

name
: ``Title'' of the element template.

  Different element templates may have the same name provided that they contain different elements.

parent

With grammars such as W3C XML Schema and RELAX NG, different element types may have save the same element name.

Examples:

1. Element `title` with enumerated values `Doctor` and `Professor` can be inserted inside element `author`.
2. Element `title` containing plain text, `strong` or `emphasis` children can be used as the title of a `figure` or a `table`.

In such situation, the XPath attribute `parent` must be used to specify to XXE in which context (that is, for which parent element) the element template can be used.

Examples:

1. Specify `parent="author"`.
2. Specify `parent="figure|table"`.

selectable

Value `true` specifies that this element template is to be listed as *`element_name`*(*`element_template_name`*) in the Edit tool.

Value `false` or `override` prevents XXE to list the element template in the Edit tool.

Value `false` is useful for an element template which is just referenced in a macro-command or in another template and which is not for general use.

Value `override` specifies that this element template is to be used everywhere the automatically generated element would otherwise have been used. See DocBook 4 example below.

**Example 7.1. DocBook 4 example**

By default, XXE creates a `listitem` containing a `para`. The following template forces XXE to create a `listitem` containing a `simpara`.

```
<cfg:elementTemplate xmlns="" name="simpara" selectable="override">
  <listitem>
    <simpara></simpara>
  </listitem>
</cfg:elementTemplate>
```

The `listitem` specified above will also be automatically used inside newly created `itemizedlist`, `orderedlist` and `variablelist`.

By default, XXE creates an `itemizedlist` containing a single `listitem`. The following template forces XXE to create an `itemizedlist` with two `listitems`.

Note that this template includes the `listitem` template specified above by using attribute `cfg:template`.

```
<cfg:elementTemplate xmlns="" name="simpara" selectable="override">
  <itemizedlist>
    <listitem cfg:template="simpara" />
    <listitem cfg:template="simpara" />
  </itemizedlist>
</cfg:elementTemplate>
```

## 10.1. Adding empty text nodes to your element templates

In some cases, you want the element template to contain an empty text node because, when a new element corresponding to this template is inserted in the document, the empty text node acts as a placeholder.

XHTML example:

```
<cfg:elementTemplate name="name_field">
  <p xmlns="http://www.w3.org/1999/xhtml"
     class="name_field"><b>Name: </b>   </p>
</cfg:elementTemplate>
```

The above element template should work fine. However all the whitespace following the `b` element will be automatically trimmed and no empty text node will inserted after it.

If you rewrite the above template as:

```
<cfg:elementTemplate name="name_field">
  <p xmlns="http://www.w3.org/1999/xhtml"
     class="name_field"><b>Name: </b><?text?></p>
</cfg:elementTemplate>
```

the element template will work as expected.

Note that the `<?text?>` processing instruction must be completely empty, otherwise it is inserted in the document as is. Also note that the `<?text?>` processing instruction must not follow or precede a text node (empty or not), otherwise it is simply discarded.

## 10.2. Specificities of `selectable="override"`

- The validity of the contents of an element template having `selectable="override"` is checked before the editing operation is performed. If this contents is found to be *structurally* invalid, then the element template is ignored and an automatically generated element is used instead.

  Example of a structurally invalid element template (the `linkend` attribute of DocBook 4 element `xref` is missing):

  ```
  <elementTemplate name="simple" selectable="override">
    <xref xmlns="" role="LINK" />
  </elementTemplate>
  ```

  Note that the above element can be made usable by slightly modifying it:

  ```
  <elementTemplate name="simple" selectable="override">
    <xref xmlns="" linkend="???" role="LINK" />
  </elementTemplate>
  ```

  The above element template is data-type invalid (`"???"` is not a valid ID), but structurally valid.

- Unlike W3C XML Schema, with RELAX NG, different element types may have save the same element name *regardless of the element type of the parent*. DocBook 5 example: there are 3 different `indexterm` element types that may be inserted into almost any parent element.

  In the case of the above example, XXE lists these 3 different `indexterm` element types in its Edit tool as: `indexterm`, `indexterm-2`, `indexterm-3`. These automatically generated names are hard to understand. Here comes `selectable="override"`. This facility may also be used to give user-friendly names to the competing element types listed by XXE.

  DocBook 5 example:

  ```
  <elementTemplate name="singular" selectable="override">
    <indexterm
      xmlns="http://docbook.org/ns/docbook"><primary></primary></indexterm>
  </elementTemplate>

  <elementTemplate name="startofrange" selectable="override">
    <indexterm xmlns="http://docbook.org/ns/docbook" xml:id="???"
               class="startofrange"><primary></primary></indexterm>
  </elementTemplate>

  <elementTemplate name="endofrange" selectable="override">
    <indexterm xmlns="http://docbook.org/ns/docbook"
  ```

```
             class="endofrange" startref="???" />
</elementTemplate>
```

In the case of the above example, the Edit tool will not list `indexterm`, `indexterm-2`, `indexterm-3`. Instead it will list `indexterm(singular)`, `indexterm(startofrange)`, `indexterm(endofrange)`.

## 11. help

```
<help
  location = anyURI
/>
```

Adds specified JavaHelp™ `.jar` file to the online help displayed using Help → Help.

Example:

```
<help location="docbook_help.jar" />
```

If the `.jar` file is called *`foo`*`.jar`, it must contain a help set file called *`foo`*`/jhelpset.hs`. In the case of the above example, `docbook_help.jar` must contain `docbook_help/jhelpset.hs`.

An online help may be available in several languages. Example: let's suppose that `docbook_help.jar` is to be available in English, French and German. In such case, you must provide three JavaHelp™ `.jar` files:

- `docbook_help.jar` containing `docbook_help/jhelpset.hs`.
- `docbook_help_fr.jar` containing `docbook_help_fr/jhelpset.hs`.
- `docbook_help_de.jar` containing `docbook_help_de/jhelpset.hs`.

Which JavaHelp™ `.jar` file is actually used depends on the locale of the machine running XMLmind XML Editor.

In the above example, we assume that English is the default, fallback, language.

Use two-letter, lowercase, language codes such as `fr` and `de` to specify languages. Do not use language variants like `fr_CA`.

Declare just `docbook_help.jar` using using a `help` configuration element (that is, `<help location="docbook_help.jar"/>`). Do not declare `docbook_help_fr.jar` and `docbook_help_de.jar`.

## 12. imageToolkit

```
<imageToolkit
  name = non empty token
>
  Content: [ description ]? [ converter ]+
</imageToolkit>

<description>
  Content: string
</description>

<converter>
  Content: input output [ shell ]+
</converter>

<input
  extensions = non empty list of file name extensions
  magicStrings = non empty list of strings
  magicNumbers = non empty list of hexBinaries
  rootNames = non empty list of QNames
/>

<output
```

```
  extensions = non empty list of file name extensions
/>

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

The `imageToolkit` configuration element allows 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. Without this mechanism, image toolkit plug-ins such as the Batik plug-in need to be written in the Java™ programming language.

The add-on called "*A sample customize.xxe* " (download and install it using Options → Install Add-ons) contains three useful `imageToolkits` from which the examples used here are taken.

An `imageToolkit` has a required `name` attribute which is used to register the plug-in and an optional `description` child element which is displayed in the dialog box opened by menu entry Help → Plug-ins.

An `imageToolkit` contains one or more `converter` child elements. A converter mainly contains a command template (`shell` child element) which can be used to convert from one or more input formats (`input` child element) to one or more output formats (`output` child element).

The following example is not useful because PNM support is provided by the plug-in called "JAI Image I/O Tools". However, this example shows how to declare a simple `imageToolkit`.

```
<imageToolkit name="netpbm">
  <description>Converts PBM, PGM, PPM images to PNG.</description>

  <converter>
    <input extensions="pnm pbm pgm ppm" magicStrings="P4 P5 P6 P1 P2 P3"/>
    <output extensions="png"/>

    <shell command='pnmtopng %A "%I" &gt; "%O"' />
  </converter>
</imageToolkit>
```

In the `input` and `output` elements, attribute `extensions` is required and specifies the file name extensions of the supported image formats. For the `output` elements, extensions other than `png`, `gif`, `jpg` and `jpeg` (case-insensitive) are currently ignored.

The `input` elements have means other than file name extensions to detect the format of images *embedded* in the XML document:

Binary images
: Attribute `magicNumbers` contains a list of numbers in hexadecimal format. These numbers are possible values for the first bytes found in the image file.

  These first bytes are often ASCII characters (even for binary images such as PNG or TIFF), that's why it is often more convenient to use attribute `magicStrings` rather than attribute `magicNumbers`.

  Example: `magicNumbers`="5034 5035" is equivalent to `magicStrings`="P4 P5".

XML images (typically SVG images)
: The format of an XML image embedded in an XML document can be detected by examining the name of its root element. Attribute `rootNames` contains a list of such `QNames` (qualified names: data type which is part of the W3C XML Schema standard).

  The following example is not useful because Batik is available as a plug-in written in Java™. However, this example shows how to declare an `imageToolkit` which handles XML images.

```
<imageToolkit name="Batik as an external SVG toolkit">
  <description>Converts SVG to PNG.</description>

  <converter>
```

```
    <input extensions="svg svgz"
           magicStrings="&lt;?xml"
           rootNames="svg:svg" xmlns:svg="http://www.w3.org/2000/svg" />
    <output extensions="png"/>
    <shell
      command='java -jar /opt/batik/batik-rasterizer.jar %A "%I" -d "%O"' />
  </converter>
</imageToolkit>
```

A `converter` element contains one or more `shell` elements. Each `shell` element contains a command template usable on a given platform. That is, a *single* shell command is executed when the `imageToolkit` is used to convert between image formats.

After substituting the variables contained in the template (see below), the command is executed the using the native shell of the machine running XXE: **cmd.exe** on Windows and **/bin/sh** on Unix (Mac OS X is considered to be a Unix platform).

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.

The `command` template must contain at least the `%I` and `%O` variables but may also contain the following variables:

| Variable | Description |
|---|---|
| `%I` | Input image file to be converted by the imageToolkit. **Warning** The file names contained in `%I` and `%O` often contain whitespaces. Do not forget to properly quote these variables in the command template. |
| `%O` | Output image file. |
| `%A` | Extra command line arguments taken from the `convertImage/parameter` elements of a `process` command (see Chapter 5, *Process commands* in *XMLmind XML Editor - Commands*). See example below. |
| `%S` | `%S` is the native path component separator of the platform. Example: `'\'` on Windows. |
| `%C`, `%c` | `%C` is the name of the directory containing the XXE configuration file from which the `imageToolkit` element has been loaded. Example: `C:\Documents and Settings\john\Application Data\XMLmind\XMLEditor5\addon`. `%c` is the URL of the above directory. Example: `file:///C:/Documents%20and%20Settings/john/Application%20 Data/XMLmind/XMLEditor5/addon`. Note that this URL does not end with a `'/'`. |

Example:

```
<imageToolkit name="Ghostscript">
  <description>Converts EPS and PDF graphics to PNG.
Important: requires Ghostscript 8+.</description>

  <converter>
    <input extensions="eps epsf ps pdf" magicStrings="%!PS %PDF"/>
    <output extensions="png"/>

    <shell command='gs -q -dBATCH -dNOPAUSE -sDEVICE=png16m
                    -r96 -dTextAlphaBits=4 -dGraphicsAlphaBits=4 -dEPSCrop
                    %A "-sOutputFile=%O" "%I"'
           platform="Unix"/>

    <shell command='gswin32c -q -dBATCH -dNOPAUSE -sDEVICE=png16m
                    -r96 -dTextAlphaBits=4 -dGraphicsAlphaBits=4 -dEPSCrop
                    %A "-sOutputFile=%O" "%I"'
           platform="Windows"/>
  </converter>
</imageToolkit>
```

**About the `%A` variable.** Let's suppose a `process` command contains the following `convertImage` element:

```
<convertImage from="raw/*.eps" to="resources" format="png">
  <parameter name="-r">120</parameter>
  <parameter name="-dDOINTERPOLATE" />
</convertImage>
```

When the above `convertImage` is executed, the command template is equivalent to:

```
gs -q -dBATCH -dNOPAUSE -sDEVICE=png16m \
    -r96 -dTextAlphaBits=4 -dGraphicsAlphaBits=4 -dEPSCrop \
    -r "120" -dDOINTERPOLATE "-sOutputFile=%O" "%I"
```

## 13. include

```
<include
  location = anyURI
/>
```

Include all elements contained in specified configuration file in current configuration file.

The URI found in the `location` attribute may be resolved using XML catalogs.

Example 1:

```
<include location="toolBar.incl" />
```

If the file containing the above snippet is `/home/john/.xxe5/addon/mydocbook.xxe`, the included file is then `/home/john/.xxe5/addon/toolBar.incl`.

Example 2:

```
<include location="docbook-config:toolBar.incl"/>
```

If XXE has been installed in `/opt/xxe/`, the included file is by default `/opt/xxe/addon/config/docbook/toolBar.incl`, because the XML catalog found in the "`DocBook`" configuration contains this rule:

```
<rewriteURI uriStartString="docbook-config:" rewritePrefix="." />
```

The "`---`" prefix before an URL instructs XXE to silently skip the inclusion when the URL cannot be successfully resolved. Example:

```
<include location="---mathml-config:docbook5/mathml_support.incl" />
```

## 14. inclusionScheme

```
<inclusionScheme
  name = non empty token
>
  Content: [ class ]?
</inclusionScheme>

<class>
  Content: Java class name
</class>
```

Register `inclusionScheme` specified by *`class`* with XXE.

An `inclusionScheme` is associated to a type of document.

To make it simple:

- Each time a document for which an inclusion scheme has been declared is opened, XXE invokes this scheme in order to ``evaluate'' the inclusion directives it contains. Evaluating the inclusion directives means replacing these directives by up-to-date included nodes.
- Each time a document for which an inclusion scheme has been declared is saved, XXE invokes this scheme in order to convert included nodes back to inclusion directives.

`Xi:include` (XInclude) elements are inclusion directives handled by the "`XInclude`" inclusion scheme. DITA elements having a `conref` attribute are inclusion directives handled by the "`Conref`" inclusion scheme.

By default, no inclusion schemes at all, not even XInclude, are associated to a document type.

Several `inclusionSchemes` can be associated to the same document type. In such case, they are invoked in the order of their registration.

Child elements of `inclusionScheme`:

class
: Register `inclusionScheme` implemented in the Java™ language by class *`class`* (implements interface `com.xmlmind.xml.load.InclusionScheme`).

Attributes of `inclusionScheme`:

name
: This name is useful to remove or replace a previously registered `inclusionScheme`. Anonymous `inclusion-Schemes` cannot be removed or replaced.

  When a `inclusionScheme` element is used to remove a registered `inclusionScheme`, a `name` attribute must be specified and there must be no `class` child element.

DITA Example:

```
  <inclusionScheme name="Conref">
    <class>com.xmlmind.xmleditext.dita.ConrefScheme</class>
  </inclusionScheme>
```

## 15. linkType

```
<linkType
  name = NMTOKEN : "default"
>
  Content: [ class ]? |
           [ link | anchor ]*
</linkType>
```

```
<class>
  Content: Java class name
</class>

<link
    match = XPath pattern
    ref = XPath expression
    refs = XPath expression
    href = XPath expression
    excludePath = Regular expression
    includePath = Regular expression
/>

<anchor
    match = XPath pattern
    name = XPath expression
/>
```

A `linkType` configuration element allows to define a generalization of the `ID`/`IDREF`/`IDREFS` mechanism for use in modular documents. Some modular documents examples are:

- A DocBook `book` including (by the means of XInclude) `chapter` documents, each `chapter` element being found in its own file.
- A DITA map referencing a number of topic files.
- A set of XHTML pages (e.g. `.html` and `.shtml` files) found in the same directory.

Defining a linkType allows to follow a link even when this link points inside another file. See Section 15.1, “Using `linkType` to implement link navigation” [88].

Defining a linkType allows to check links even when some of the links point outside the file containing these links. See Section 15.2, “Using `linkType` to implement link validation” [88].

A `linkType` configuration element has a single optional attribute: `name`. The default value of attribute name is `default`. Giving a meaningful name to a linkType is useful in 3 cases:

1. When your configuration file contains several `linkType` elements. This is rarely needed, but it may happen. See example below [86].
2. The name of the `linkType` is `xml:id`. This special name means that the anchors defined by this link type are true XML IDs and thus, there is no need to create proprietary XPointers to implement link navigation. These proprietary XPointers are described in Section 15.1, “Using `linkType` to implement link navigation” [88].
3. When you want to allow removing this `linkType` from the configuration. This is possible because a `linkType` configuration element without any child element may be used to remove from a configuration a previously defined `linkType` having the same `name`.

A `linkType` configuration element contains either a single `class` child element or an number of `anchor` and `link` child elements:

`class`

This element contains the fully qualified name of a class which implements interface `com.xmlmind.xmleditapp.linktype.LinkType`.

An example of such class is `com.xmlmind.xmleditapp.IDLinkType`, which leverages the standard `ID`/`IDREF`/`IDREFS` mechanism to implement a link type. When no `linkType` element is found in a configuration file, XXE will behave as if the following one was defined:

```
<linkType>
  <class>com.xmlmind.xmleditapp.linktype.IDLinkType</class>
</linkType>
```

`anchor`

The `anchor` element is used to detect in a document all the elements which may be used as link targets. This kind of elements is called *link target*, or more simply *anchor*. XHTML 1.0 example:

```
<anchor match="*[@id]" name="@id" />
<anchor match="html:a[@name]" name="@name" />
```

For a given link type and in a given document, an anchor is uniquely identified by its name. That is, for a given link type and in a given document, it is an error to find several anchors having the same name.

`link`
: The `link` element is used to detect in a document all elements acting as links pointing to targets. This kind of elements is called *link source*, or more simply *link*. XHTML example:

```
<link match="html:a[@href]" href="@href" />
```

A link must reference the name of an existing anchor and optionally, depending on the link type, the file containing this existing anchor. That is, it is an error to find a link pointing to an unknown anchor and/or pointing to a non-existent file.

For a given link type, the same element may be at the same time an anchor and also one or more links.

It is also possible to define several link types in the same configuration. Fictitious example:

```
<linkType>1
  <link match="*[@ref]" ref="@ref" />
  <anchor match="*[@id]" name="@id" />
</linkType>

<linkType name="bug">2
  <link match="supportTicket[@bug]" href="@bug" />
  <anchor match="bugReport[@number]" name="@number" />
</linkType>
```

1 The first `linkType` corresponds to general purpose ID/IDREF cross-references.
2 The second `linkType` corresponds to specialized cross-references.

Attributes of the `anchor` child element:

`match`
: Specifies an XPath pattern which is used to detect an element which may be used as a link target.

`name`
: The XPath expression is evaluated in the context of the element matched by attribute `match`. It returns a string which is the name of the detected anchor.

The name of a anchor is any non-empty string which does not contain whitespace. Special value `"-"` may be used to ignore the element matched by attribute `match`.

Attributes of the `link` child element:

`match`
: Specifies an XPath pattern which is used to detect an element which acts as a link source.

`ref`, `refs`, `href`
: One of the `ref`, `refs`, `href` attributes must be specified. The XPath expression is evaluated in the context of the element matched by attribute `match`. It returns a string which specifies the target of the link. This string cannot be empty. Special value `"-"` may be used to ignore the element matched by attribute `match`.

The string returned by the `ref` XPath expression is expected to be an anchor name.

The string returned by the `refs` XPath expression is expected to be one or more anchor names separated by whitespace.

The string returned by the `href` XPath expression is expected to conform to the following syntax:

```
[ absolute_or_relative_URI ]? [ '#' anchor_name ]?
```

A relative URI is relative to the base URI of the element matched by attribute `match`.

`excludePath`, `includePath`
: These attributes are ignored unless the `href` attribute has been specified. They allow to make a difference between a link to an external resource and a link to some content found in a peer XML document. Of course, a `linkType` is about links pointing inside the local document or inside peer XML documents, and not about links to external resources.

The value of these attributes is a regular expression which must match the *`absolute_or_relative_URI`* part of the string returned by the `href` XPath expression.

An `href` returning an *`absolute_or_relative_URI`* which matches the `excludePath` regular expression is considered to be a link to an external resource and as such, is ignored. Example: ignore all absolute URIs:

```
excludePath="^[a-zA-Z][a-zA-Z0-9.+-]*:"
```

An `href` returning an *`absolute_or_relative_URI`* which does not match the `includePath` regular expression is considered to be a link to an external resource and as such, is ignored. Example: ignore URIs not ending with the `.xml`, `.dbk` and `.db5` suffixes.

```
includePath="\.(xml|dbk|db5)$"
```

The `excludePath` and `includePath` attributes are rarely used because they are implicitly defined by XXE as:

- Ignore absolute URIs which cannot be made relative to the URI of the document containing the element matched by `match`. For example, an "`http://`" URI cannot be made relative to a "`file:/`" URI, so ignore it.
- Ignore URIs which do not have the same file extension as the document containing the element matched by `match`. For example, a link contained in `packaging.dita` and pointing to `samples/manifest.xml` is ignored.

An actual, commented, `linkType` for DocBook 5:

```
<linkType>
  <!-- link, xref, biblioref -->
  <link match="*[@linkend]" ref="@linkend" />
  <link match="*[@endterm]" ref="@endterm" />

  <!-- area, co -->
  <link match="*[@linkends]" refs="@linkends" />

  <!-- callout -->
  <link match="*[@arearefs]" refs="@arearefs" />

  <!-- glosssee, glossseealso -->
  <link match="*[@otherterm]" ref="@otherterm" />

  <!-- indexterm -->
  <link match="*[@startref]" ref="@startref" />
  <link match="*[@zone]" refs="@zone" />

  <!-- th, td -->
  <link match="*[@headers]" refs="@headers" />

  <!-- We'll assume that "*[@xlink:href]" is a replacement for
       ulink and not an alternative to link and xref. -->

  <!-- Allows to use xi:include as a link when the transclusion
       has been turned off. -->
  <link match="xi:include" href="xincl:toHref(.)"
        xmlns:xincl="java:com.xmlmind.xml.xinclude.XInclude" />

  <anchor match="*[@xml:id]" name="@xml:id" />
</linkType>
```

## 15.1. Using `linkType` to implement link navigation

The user can navigate from a link source to the link target and the other way round by using the items of menu Select → Link and also the equivalent toolbar buttons.

These menu items and these toolbar buttons are always operative whether a `linkType` confirguration element has been defined or not. The reason is, when no `linkType` element is found in a configuration file, XXE will behave as if the following one was defined:

```
<linkType>
  <class>com.xmlmind.xmleditapp.linktype.IDLinkType</class>
</linkType>
```

Note that during the link navigation, XXE may have to open the document containing the destination. When this is the case, the user is prompted to confirm that she/he really wants to open this document and whether she/he wants to open it in read-only mode or in normal, read-write, mode.

This works because command `XXE.edit` in *XMLmind XML Editor - Commands* also allows to select a link target or a link source. In order to do that, `XXE.edit` is passed an URL ending with a proprietary XPointer leveraging the link type of the destination. Examples of such proprietary XPointers:

- Open file `book.xml` and select the element having "`introduction`" as its anchor name. The searched anchor "`introduction`" belongs to the link type called "`default`".

  ```
  book.xml#xmlns(l=urn:xxe:link)l:anchor(introduction)
  ```

- Same but the searched anchor "`978-3-16-148410-0`" belongs to the link type called "`ISBN`".

  ```
  book.xml#xmlns(l=urn:xxe:link)l:anchor(978-3-16-148410-0%20ISBN)
  ```

- Open file `introduction.dita` and select the element acting as a link pointing to "`concepts.dita#concepts/collection`". The searched anchor "`concepts/collection`" belongs to the link type called "`default`".

  ```
  introduction.dita#xmlns(l=urn:xxe:link)l:link(concepts.dita%20concepts/collection)
  ```

- Open file `chapter2.xml` and select the element acting as a link pointing to "`reference`". The searched anchor "`reference`" belongs to the link type called "`default`".

  ```
  chapter2.xml#xmlns(l=urn:xxe:link)l:link(%20reference)
  ```

## 15.2. Using `linkType` to implement link validation

In order to use one or more `linkType` elements defined in a configuration file to perform link validation in addition to link navigation, you'll have to declare the following validateHook [114]:

```
<validateHook>
  <class>com.xmlmind.xmleditapp.linktype.LinkChecker</class>
</validateHook>
```

By default, this `validateHook` checks all anchors and all links, and this for all the link types defined the configuration file. By default, the diagnostics reported by this `validateHook` are *added* to those reported by the validation performed by the DTD or schema. All these default behaviors may be changed by using the bean properties [76] described below:

| Name | Type | Default | Description |
|---|---|---|---|
| excludedLinkTypes | `String` (zero or more link type names separated by whitespace) | `""` (empty string) | Do not check the anchor and links belonging to specified link types. |

| Name | Type | Default | Description |
|---|---|---|---|
| checkRefs | boolean | `true` | If `true`, check links which are direct references to anchors (that is, the equivalent of `IDREF` and `IDREFS`). |
| checkHrefs | boolean | `true` | If `true`, check links which are hrefs (that is, *`URI#anchor_name`*). |
| checkAnchors | boolean | `true` | If true, check anchors (that is, the equivalent of `ID`). |
| replaceDiagnostics | boolean | `false` | If `true`, replace some of the diagnostics issued by the DTD or schema. For example, if checkAnchors is `true`, replace the duplicate ID error messages reported by the DTD or schema by the errors detected by this link checker. |
| checkIfMemberOf-DocSet | boolean | `false` | If `true`, check anchors and links, but only when the document being checked is part of a document set. This implies that when the document being checked is *not* part of the document set, the job done by the DTD or schema is sufficient. |

An actual, commented, `validateHook` for DocBook 5:

```
<validateHook name="checkLinks">
  <class>com.xmlmind.xmleditapp.linktype.LinkChecker</class>

  <!-- Let the schema check IDs. We'll only check refs. -->
  <property name="checkAnchors" type="boolean" value="false" />

  <property name="replaceDiagnostics" type="boolean" value="true" />

  <!-- When the document is not a member of a set, the DTD is just fine
       to check IDREFs. -->
  <property name="checkIfMemberOfDocSet" type="boolean" value="true" />

  <property name="excludedLinkTypes" type="String" value="xml:id" />
</validateHook>
```

## 15.3. Using `linkType` to define custom, specialized, attribute editors

The `attributeEditor` configuration element [58] allows to extend the Attributes tools by implementing custom, specialized, attribute editors. The link type feature comes with two implementations of interface `com.xmlmind.xmledit.cmd.attribute.SetAttribute.ChoicesFactory`. These two classes allow to quickly and easily specify the value of attributes which are anchor names and/or the value of attributes which are pointers to anchors:

`com.xmlmind.xmleditapp.linktype.RefChoicesFactory`

This class lists all the anchor names found in the document being edited. Each anchor name is followed by a short description of the element acting as an anchor.

If the document being edited is part of a document set, this class also lists all the anchor names found in the peer documents.

DocBook example:

```
<attributeEditor attribute="linkend" elementMatches="xref|link">
  <class>com.xmlmind.xmleditapp.linktype.RefChoicesFactory</class>
  <property name="listIfMemberOfDocSet" type="boolean" value="true" />
</attributeEditor>

<attributeEditor attribute="id" elementMatches="*">
```

```
  <class>com.xmlmind.xmleditapp.linktype.RefChoicesFactory</class>
  <property name="listIfMemberOfDocSet" type="boolean" value="true" />
</attributeEditor>
```

`com.xmlmind.xmleditapp.linktype.HrefChoicesFactory`

This class lists all the anchor names found in the document being edited. An anchor name is prefixed with '`#`'. Each anchor name is followed by a short description of the element acting as an anchor.

If the document being edited is part of a document set, this class also lists the relative URIs of all peer documents. If the user types *`relative_URI#`* in the Value field of the Attribute tool or in the specialized dialog box displayed by the Edit button, this class will auto-complete *`relative_URI#`* by all the anchor names found in peer document *`relative_URI`*.

XHTML example:

```
<attributeEditor attribute="href" elementMatches="html:a">
  <class>com.xmlmind.xmleditapp.linktype.HrefChoicesFactory</class>
</attributeEditor>
```

DITA topic example:

```
<attributeEditor attribute="href" elementMatches="xref|link">
  <class>com.xmlmind.xmleditapp.linktype.HrefChoicesFactory</class>
</attributeEditor>
```

## 16. menu

```
<menu
  label = non empty token
  name = NMTOKEN
  insert = non empty token
>
  Content: [ menu | separator | item | insert ]*
</menu>

<separator />

<insert />

<item
  label = non empty token
  icon = anyURI
  command = NMTOKEN (optionally preceded by a command namespace [68])
  parameter = string
>
  Content: [ accelerator ]?
</item>

<accelerator
  code = key code
  modifiers = possibly empty list of (ctrl|shift|alt|meta|altGr|mod)
/>
```

Specifies the label and content of the XML (placeholder) menu.

Note that the mnemonic of a menu or of a menu item is specified by adding an underscore ('_') before the character used as a mnemonic. Currently, only `a-zA-Z0-1` characters can be used as mnemonics. Moreover, Java™ does not make a difference between an uppercase letter and a lowercase letter.

Example:

```
  <menu label="_XHTML">
    ...
    <menu label="C_ell">
      <item label="_Increment Column Span"
            icon="../common/icons2/incrColumnSpan.gif"
            command="xhtml.tableEdit" parameter="incrColumnSpan"/>
      <item label="_Decrement Column Span"
            icon="../common/icons2/decrColumnSpan.gif"
            command="xhtml.tableEdit" parameter="decrColumnSpan"/>
      ...
    </menu>
    <separator />
    <item label="_Go to Opposite Link End"
          command="followLink" parameter="swap"/>
    ...
    <separator />
    <item label="Pre_view" icon="../common/icons/Refresh16.gif"
          command="xhtml.preview">
      <accelerator code="F5" />
    </item>
  </menu>
```

There are two ways to extend previously defined menu:

1. By using the `insert` child element.
2. By using the `insert` attribute.

Both methods cannot be used in the same `menu` element. Method 1 is faster and simpler to use than method 2.

1. Using the `insert` child element. Example:

   ```
   <include location="../common/common.incl" />
   <!-- =============================================
   Let's suppose this menu is defined in common.incl:

   <cfg:menu label="Insert">
     <cfg:item label="Insert..." command="insert" parameter="into" />
   </cfg:menu>
   ============================================== -->

   <cfg:menu label="Insert">
     <cfg:item label="Insert Before..." command="insert"
        parameter="before[implicitElement]" />
     <cfg:insert />
     <cfg:item label="Insert After..." command="insert"
       parameter="after[implicitElement]" />
   </cfg:menu>
   ```

   The `insert` child element is a directive which means: insert all the items of the previous definition of the same menu here.

   ### About the `label` attribute

   When you extend a previously defined menu, the `label` attribute specifies the title of the extended menu. This means that you can *change* the title of a menu at the same time you extend it.

   If you don't want to do that, which is often the case, simply specify `label="-"` in the menu extension. This is simpler and safer than repeating the original label of the menu. In such case, the above example becomes:

   ```
   <cfg:menu label="-">
     <cfg:item label="Insert Before..." command="insert"
        parameter="before[implicitElement]" />
     <cfg:insert />
     <cfg:item label="Insert After..." command="insert"
       parameter="after[implicitElement]" />
   </cfg:menu>
   ```

2. Using the `insert` attribute. Example:

```
<include location="../common/common.incl" />
<!-- ============================================
Let's suppose this menu is defined in common.incl:

<cfg:menu label="Insert">
  <cfg:item label="Insert Before..." command="insert"
     parameter="before[implicitElement]" />
  <cfg:item label="Insert After..." command="insert"
    parameter="after[implicitElement]" />
</cfg:menu>
=============================================== -->

<cfg:menu label="Insert" insert="Insert After...">
  <cfg:item label="Insert..." command="insert" parameter="into" />
</cfg:menu>
```

The `insert` attribute is a directive which means: insert all the items found in this menu into the previous definition of the same menu, and this, at specified position.

The value of the `insert` attribute is the `label` of an `item` found in the previous definition of the same menu. This label may be preceded by modifier "`before `" or by modifier "`after `". Modifier "`before `" is the implicit one.

In the above example, extending menu "`Insert`" could have also been achieved by using:

```
<cfg:menu label="Insert" insert="before Insert After...">
  <cfg:item label="Insert..." command="insert" parameter="into" />
</cfg:menu>
```

or by using:

```
<cfg:menu label="Insert" insert="after Insert Before...">
  <cfg:item label="Insert..." command="insert" parameter="into" />
</cfg:menu>
```

Alternatively, the value of the `insert` attribute may be `##first` or `##last`. Value `##first` specifies the first item of the previous definition of the same menu. Value `##last` specifies the last item of the previous definition of the same menu. Example:

```
<cfg:menu label="Insert" insert="before ##last">
  <cfg:item label="Insert..." command="insert" parameter="into" />
</cfg:menu>
```

The value of the `insert` attribute may start with `ifDefined(`*`system_property_name`*`)`. In such case, the previously defined menu is extended if and only if a system property called *`system_property_name`* has been defined (no matter its value). Example:

```
<menu label="_XHTML"
      insert="ifDefined(HAS_CONVERT_SUBMENU)after ##last">
  <separator />
  <menu label="_Convert Document">
  ...
  </menu>
</menu>
```

## 16.1. Multiple menus

Specifying a `name` attribute for the `menu` element allows to create a GUI having several XML application specific menus.

Example:

1. In *`XXE_user_preferences_dir`*`/addon/xhtml.xxe`, add something like this:

```
<menu name="menu2" label="My XHTML Menu">
  ...
</menu>
```

2. In `XXE_user_preferences_dir/addon/docbook.xxe`, add something like this:

   ```
   <menu name="menu2" label="My DocBook Menu">
     ...
   </menu>
   ```

   Notice that the *same* name `menu2` is used in all XML application specific configuration files.

3. In `XXE_user_preferences_dir/addon/customize.xxe_gui` (see XMLmind XML Editor - Customizing the User Interface), add something like this:

   ```
   <menuItems name="configSpecificMenuItems2">
     <class>com.xmlmind.xmleditapp.kit.part.ConfigSpecificMenuItems</class>
     <property name="specificationName" type="String" value="menu2" />
   </menuItems>

   <menu name="configSpecificMenu2" label="_My Menu">
     <menuItems name="configSpecificMenuItems2" />
   </menu>

   <menu name="fileMenu">
     <menu name="configSpecificMenu2" />
     <insert />
   </menu>
   ```

## 17. newElementContent

```
<newElementContent
  addRequiredAttributes = boolean : true
  emptyAttributes = boolean : false
  generateIds = boolean : false
  addChildElements = (noChoice|
                      firstChoice|
                      simplestChoice|
                      elementOnlyContentNotEmpty) : simplestChoice
/>
```

Parametrizes the content of a newly inserted element automatically generated by XXE (has no effect on element templates):

addRequiredAttributes, emptyAttributes, generateIds
: Example:

  ```
  <!ELEMENT anchor EMPTY>
  <!ATTLIST anchor id ID #REQUIRED>
  ```

  `addRequiredAttributes="false"` creates `<anchor/>` (`emptyAttributes` and `generateIds` are ignored in such case) .

  `addRequiredAttributes="true"`, `emptyAttributes="false"`, `generateIds="false"` creates `<anchor id="???"/>`.

  `addRequiredAttributes="true"`, `emptyAttributes="true"`, `generateIds="false"` creates `<anchor id=""/>`.

  `addRequiredAttributes="true"`, `generateIds="true"`, creates `<anchor id="__f34a62b09.b"/>` (whatever is the value of `emptyAttributes`).

addChildElements
: Example:

```
<!ELEMENT section (title,(table|para)+)>
<!ELEMENT para #PCDATA>
<!ELEMENT table (header?,row*)>
```

`addChildElements="noChoice"` creates `<section><title></title></section>` (which is invalid) because it will not choose between a `para` and a `table`.

`addChildElements="firstChoice"` creates `<section><title></title><table></table></section>`. This option is useful for authors who write small schemas for use in XXE and don't want to worry about elementTemplates [77].

`addChildElements="simplestChoice"` creates `<section><title></title><para></para></section>` because the content of a `para` is simpler than the content of a `table`.

`addChildElements="elementOnlyContentNotEmpty"` is a variant of `simplestChoice` for elements having an element-only content. In the case of this kind of elements, this variant will not create empty elements, even if this is allowed by the schema. For example, using this option creates this table: `<table><row><cell></cell></row></table>`, where using `simplestChoice` would have created an empty table: `<table></table>`.

Example:

```
  <newElementContent generateIds="true" addChildElements="firstChoice" />
```

## 18. property

```
<property
  name = non empty token
  url = boolean : false
  xml:space = preserve
>text</property>
```

Define Java™ system property (that is, java.lang.System.setProperty()) called *name*. The value of this property is specified by *text*.

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

This element is mainly intended to be used to configure some custom commands.

Examples:

```
<property name="color">red</property>

<property name="icon.3" url="true">resources/icon.gif</property>
```

The "$c" pseudo-variable may be referenced in the name of the property. This pseudo-variable is substituted with the name of the configuration being loaded. Example: the following property is used to parameterize the behavior of commands `insertNewlineOrSplitBlock` in *XMLmind XML Editor - Commands*, `insertSameBlock` in *XMLmind XML Editor - Commands* and `deleteSelectionOrJoinBlockOrDeleteChar` in *XMLmind XML Editor - Commands* when invoked from a DITA topic.

```
<property name="$c blockList">
  p
  dt
  li
  dlentry
  step
  substep
  choice
</property>
```

## 19. parameterGroup

```
<parameterGroup
  name = non empty token
>
  Content: [ parameter | parameterGroup ]*
</parameterGroup>

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

Define a named group of XSLT style sheet parameters for use inside element `transform` of a process command [68].

If the `url` attribute of a `parameter` 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.

Parameter groups make it easier to customize the XSLT style sheet used to convert a document to other formats such as HTML or PDF.

For example, instead of redefining the whole process command `docb.toPS`, suffice to redefine in `%APPDATA%\XMLmind\XMLEditor5\addon\customize.xxe` (`$HOME/.xxe5/addon/customize.xxe` on Linux) its *placeholder parameterGroup* named "`docb.toPS.transformParameters`".

Examples:

```
<parameterGroup name="docb.toPS.transformParameters">
  <parameter name="variablelist.as.blocks">1</parameter>
</parameterGroup>

<parameterGroup name="docb.toRTF.transformParameters">
  <parameterGroup name="docb.toPS.transformParameters"/>
</parameterGroup>

<parameterGroup name="docb.toPS.FOPParameters">
  <parameter name="configuration" url="true">fop.xconf</parameter>
</parameterGroup>
```

## 20. parameterSet

```
<parameterSet
  name = non empty token
>
  Content: [ parameterGroup ]*
</parameterGroup>
```

A `parameterSet` is a named set of XSL style sheet parameters normally created using the Parameter Set → Save menu item of the Parameter Set Chooser dialog box in *XMLmind XML Editor - Online Help*. However a consultant may create such parameter sets manually and add them to a (manually written) configuration file. This allows to make available to a group of authors a number of predefined parameter sets.

A parameter set has a name which uniquely identifies it. It contains a list of `parameterGroups` [95], each `parameterGroup` corresponding to an existing process command. Example:

```
<parameterSet name="Experts">
  <parameterGroup name="docb.toPS.transformParameters">
    <parameter name="profile.condition">expert</parameter>
    <parameter name="variablelist.as.blocks">1</parameter>
  </parameterGroup>
  <parameterGroup name="docb.toRTF.transformParameters">
    <parameter name="profile.condition">expert</parameter>
  </parameterGroup>
</parameterSet>
```

## 21. preserveSpace

```
<preserveSpace
  elements = list of XPath (subset [75])
/>
```

Specifies which elements are whitespace-preserving.

Using standard attribute `xml:space` with default value `preserve` is still the preferred way of specifying this. However, this is not always possible, for example in the case of DTDs/ W3C XML schemas that you don't control or in the case of RELAX NG schemas which do not really support the concept of attribute default value.

DocBook example:

```
<cfg:preserveSpace xmlns=""
    elements="address funcsynopsisinfo classsynopsisinfo
              literallayout programlisting screen synopsis" />
```

## 22. relaxng

```
<relaxng
  location = anyURI
  compactSyntax = boolean
  encoding = any encoding supported by Java™
/>
```

Use the RELAX NG schema specified by this element to constrain the document.

location
: Required. Specifies the URL of the RELAX NG schema.

compactSyntax
: Specifies that the RELAX NG schema is written using the compact syntax. Without this attribute, if `location` has a "`rnc`" extension, the schema is assumed to use the compact syntax, otherwise it is assumed to use the XML syntax.

encoding
: Specifies the character encoding used for a RELAX NG schema written using the compact syntax. Ignored if the XML syntax is used. Without this attribute, the schema is assumed to use the native encoding of the platform.

Note that

- if a document contains a document type declaration (<!DOCTYPE>) which defines elements,
- or if the root element of a document has `xsi:schemaLocation`/`xsi:noNamespaceSchemaLocation` attributes,
- of if a document contains a `<?xml-model href="..."?>`,

the grammar specified this way is used and the RELAX NG schema specified in the configuration file is ignored.

Example:

```
<relaxng location="rng/xhtml-strict.rng" />
```

Compact syntax example:

```
<relaxng compactSyntax="true" encoding="ISO-8859-1"
         location="example3.rnc" />
```

### Validating RELAX NG schemas or ISO Schematrons

If your configuration allows to edit RELAX NG schemas, you'll typically specify:

```
<relaxng location="rng/relaxng.rng" />
```

where `rng/relaxng.rng` is a copy of the RELAX NG schema for RELAX NG.

However, it is also possible to validate the document being edited using XXE's built-in schema validator. This kind of validation is more comprehensive than the one performed by the RELAX NG schema for RELAX NG alone. This is achieved by referencing the "`xxe-schema-for-relaxng`" dummy URI:

```
<relaxng location="xxe-schema-for-relaxng" />
```

The same trick also applies to ISO Schematrons, which are normally validated against the RELAX NG schema for ISO Schematron (`iso-schematron.rnc`). In this case, you'll have to reference the "`xxe-schema-for-schematron`" dummy URI.

## 23. saveOptions

```
<saveOptions
  encoding = NMTOKEN
  indent = none | (int >= 0)
  maxLineLength = unbounded | (int > 0)
  addOpenLines = boolean
  cdataSectionElements = list of XPath (subset [75])
  saveCharsAsEntityRefs = boolean
  charsSavedAsEntityRefs = list of character ranges
  favorInteroperability = boolean
  omitXMLDeclaration = false | true | auto : false
/>
```

Force XXE to use the specified save options for this type of document, unless Options → Preferences, Save tab, Override settings specified in config. files checkbox has been checked by the user, in which case, it is the save options specified in the dialog box which are used.

`encoding`
: Specifies the encoding used for XML files saved by XXE.

`indent`
: If this value is different from `none`, XML files saved by XXE are indented .

  Note that XXE cannot indent XML files not constrained by a grammar.

`indentation`
: Specifies the number of space characters used to indent a child element relatively to its parent element.

`maxLineLength`
: Specifies the maximum line length for elements containing text interspersed with child elements.

  This value is only used as a hint: XML files created by XXE may contain lines much longer than the specified length.

`addOpenLines`
: If value is `true`, an open line is added between the child elements of a parent element (if the content model of the parent only allows child elements).

`cdataSectionElements`
: List of XPaths specifying elements. These elements are expected to only contain text and to have an `xml:space="preserve"` attribute.

Text contained in elements matching any of the XPaths specified by this attribute is saved as a CDATA section. Text inside a CDATA section is not escaped which makes it more readable using a text editor. Example:

```
<script type="text/javascript"><![CDATA[function min(x, y) {
    return (x < y)? x : y;
}]]></script>
```

If an element matching any of the XPaths specified by this attribute contains anything other than text (even a comment), it is saved normally.

Note that, in most configuration elements, XXE only supports the XPath subset [75] needed to implement XML-Schemas (but not only relative paths, also absolute paths). Moreover, for efficiency reasons, an XPath whose last step does not test an element name is ignored. For example, "`foo//*`" is ignored.

`saveCharsAsEntityRefs`
Specifies whether characters not supported by the encoding are saved as entity references (example: "`&euro;`") or as numeric character references (example: "`&#8364;`").

Of course, for a character to be saved as an entity reference, the corresponding entity must have been defined in the DTD.

`charsSavedAsEntityRefs`
Specifies which characters, even if they are supported by the encoding, are always saved as entity references.

For example, the Copyright sign is supported by the ISO-8859-1 encoding but you may prefer to see it saved as "`&copy;`". In such case, specify `charsSavedAsEntityRefs="169"`.

Ignored if `saveCharsAsEntityRefs` is false.

This attribute contains a list of character ranges. A character range is either a single character or an actual range *`char1`*`:`*`char2`*.

A character may be specified using its Unicode character number, in decimal (example: `233` for e acute), in hexadecimal (example: `0xE9`) or in octal (example: `0351`).

Because names are easier to remember than numbers, a character may also be specified using its entity name as defined in the DocBook 4.2 DTD (example: `eacute`). Note that is possible whatever is the DTD or Schema targeted by the configuration file.

### Note

There is no need to specify the non-breaking space character (`nbsp` = `160` = `0xa0` = `0240`) as it is always implicitly added to this list.

`favorInteroperability`
If value is `true`, favor interoperability with HTML and SGML.

- Empty elements having a non empty content are saved as "`<tag></tag>`".
- Empty elements having an empty content are saved as "`<tag />`" (with a space after the tag).

`omitXMLDeclaration`
Specifies whether the XML declaration (that is, `<?xml version="1.`*`x`*`"...?>`) is to be omitted from the save file.

### Note

Omitting the XML declaration is useful when an XHTML document is delivered by the Web server to the Web browser as if it were an HTML document. That is, for the Web browser, the media type of the document is `text/html` and not `application/xhtml+xml`.

This is useful because both the XML declaration and the `<!DOCTYPE>` declaration have an effect on the behavior of Web browsers. See *Activating Browser Modes with Doctype*.

`false`
: Default value. Do not omit the XML declaration from the save file.

`true`
: Omit the XML declaration and force the encoding of the save file to be UTF-8.

`auto`
: Determine whether the XML declaration is to be omitted by examining the content of the document to be saved.

  If the document is an XHTML document and contains `<meta http-equiv="Content-Type" content="`*`MEDIA`*`;charset=`*`CHARSET`*`"/>`, then:
  - If the media type is "`text/html`" and the charset is "`UTF-8`", then the XML declaration is omitted and the encoding of the save file is forced to be UTF-8.
  - Otherwise the XML declaration is *not* omitted but, if a valid charset has been successfully parsed, the encoding of the save file is forced to be this charset.

  If the document is not an XHTML document or does not contain `<meta http-equiv="Content-Type".../>`, then the XML declaration is *not* omitted.

  Element `<meta charset="`*`CHARSET`*`"/>` is considered to be equivalent to `<meta http-equiv="Content-Type" content="text/html;charset=`*`CHARSET`*`"/>`.

  The values of all the aforementioned attributes are parsed in a case-insensitive manner.

Examples:

```
<saveOptions addOpenLines="false" />

<saveOptions xmlns:htm="http://www.w3.org/1999/xhtml"
  cdataSectionElements="htm:head/htm:script"
  omitXMLDeclaration="auto" />

<saveOptions saveCharsAsEntityRefs="true"
             charsSavedAsEntityRefs="copy reg 023400:024000"/>
```

Note that a `saveOptions` element does not replace the `saveOptions` element previously found in a configuration file. When a configuration file contains several `saveOptions` elements, these `saveOptions` elements are merged.

Example:

```
<cfg:saveOptions xmlns="" cdataSectionElements="script pre"
                 addOpenLines="false"/>
        .
        .
        .
<cfg:saveOptions addOpenLines="true" encoding="ISO-8859-1"/>
```

is equivalent to:

```
<cfg:saveOptions xmlns="" cdataSectionElements="script pre"
                 addOpenLines="true" encoding="ISO-8859-1" />
```

## 24. schema

```
<schema>
  Content: location | noNamespaceLocation | (location noNamespaceLocation)
</schema>

<location>
  Content: list of anyURI pairs
</location>

<noNamespaceLocation>
```

```
  Content: anyURI
</noNamespaceLocation>
```

Use the W3C XML Schema specified by this element to constrain the document.

The content of child element `location` is identical to the one of standard attribute `xsi:schemaLocation`. The content of child element `noNamespaceLocation` is identical to the one of standard attribute `xsi:noNamespaceSchemaLocation`.

Note that

- if a document contains a document type declaration (<!DOCTYPE>) which defines elements,
- or if the root element of a document has `xsi:schemaLocation`/`xsi:noNamespaceSchemaLocation` attributes,
- of if a document contains a `<?xml-model href="..."?>`,

the grammar specified this way is used and the W3C XML Schema specified in the configuration file is ignored.

Example:

```
<schema>
  <location>http://www.w3.org/1999/xhtml
            xsd/5.0/xhtml5.xsd</location>
</schema>
```

### Validating W3C XML Schemas

If your configuration allows to edit W3C XML Schemas, you'll typically specify:

```
<schema>
  <location>http://www.w3.org/2001/XMLSchema
            xsd/XMLSchema.xsd</location>
</schema>
```

where `xsd/XMLSchema.xsd` is a copy of the XML Schema for XML Schemas.

However, it is also possible to validate the document being edited using XXE's built-in schema validator. This kind of validation is more comprehensive than the one performed by the XML Schema for XML Schema alone. This is achieved by referencing the "`xxe-schema-for-schema`" dummy URI:

```
<schema>
  <location>http://www.w3.org/2001/XMLSchema
            xxe-schema-for-schema</location>
</schema>
```

## 25. schematron

```
<schematron
  location = anyURI
  phase = non empty token
  evaluatePhase = boolean : false
>
```

Specifies which Schematron schema to use to validate the document being edited.

Note that a Schematron schema is by no mean a replacement for *grammars*: DTD, W3C XML Schema or RELAX NG schema. A Schematron schema is mainly useful to enforce *business rules*. Example: the authors in your organization must write articles conforming to the DocBook grammar but they also need to follow this business rule: first section must have a title called "`Introduction`" and last section must have a title called "`Conclusion`".

XXE built-in Schematron implementation supports both ISO Schematron or Schematron 1.5 schemas. The only supported query language binding is XSLT 1 (`queryBinding="xslt"`).

Attributes:

location
: URL of the Schematron schema.

  Note that `location` may point to a schema other than a schematron, but where some Schematron elements have been embedded (typically RELAX NG, but not with the compact syntax).

phase
: The ID of the phase to use for validation. By default, `#DEFAULT` if a default phase has been declared in the schematron, `#ALL` otherwise.

  The value of this attribute may also be an XPath expression which is used to compute the ID of the phase based on the contents of the document being edited. See `evaluatePhase` below.

evaluatePhase
: If this attribute is specified with value `true`, then attribute `phase` is understood as being an XPath expression rather than a literal phase ID. Each time a Schematron validation is to be performed, this XPath expression is evaluated in the context of the document and is expected to return the ID of the phase which is to be used for the validation.

DocBook 5 (RELAX NG) example:

```
<schematron location="docbook.sch" />
```

DocBook 4.4 (DTD) example:

```
<schematron location="docbook.sch"
            phase="if(/*/@status='draft','empty','#ALL')"
            evaluatePhase="true" />
```

The meaning of the `phase` attribute is: if we are working on a draft document, no real schematron validation (phase ID = `empty`) should be performed. (The schematron `docbook.sch` actually contains an empty phase having `empty` as its ID, that is, `<sch:phase id="empty"/>`.)

## 25.1. Relationship between `schematron` and `validateHook`

This `schematron` configuration element is a validateHook [114] configuration element in disguise. A `schematron` element is equivalent to:

```
<validateHook name="Schematron">
  <class>com.xmlmind.xmleditapp.config.SchematronHook</class>
</validateHook>
```

However the above syntax cannot be used for `SchematronHook` which requires a number of arguments (e.g. the URL of the schematron).

This information is worth mentioning for two reasons:

1. Document hooks are *ordered*. They are invoked in the order of their declarations. This is also true for `schematron`. In the example below, schematron validation is guaranteed to be invoked *after* the DocBook document hook:

   ```
   <!-- Fixes the cols attribute of tgroup and entrytbl if needed to. -->
   <validateHook>
     <class>com.xmlmind.xmleditext.docbook.table.ValidateHookImpl</class>
   </validateHook>

   <schematron location="docbook.sch" />
   ```

2. The snippet below may be used to *remove* previously declared `schematron`.

   ```
   <validateHook name="Schematron" />
   ```

## 26. spellCheckOptions

```
<spellCheckOptions
  useAutomaticSpellChecker = boolean
  languageAttribute = list of QNames
  defaultLanguage = language
  checkComments = boolean
  checkedProcessingInstructions = list of Names
  checkedAttributes = list of XPath (subset [75])
  skippedElements = list of XPath (subset [75])
/>
```

Specifies, on a per document type basis, options for the spell checker. Used by both the automatic (AKA on-the-fly) and the ``traditional" spell checkers.

useAutomaticSpellChecker

If `true`, the automatic spell checker must be automatically activated each time a document of that type is opened.

Default: `false`; see language lookup [102].

This setting may be overridden by the user with Options → Preferences, Tools/Spell section, Automatic Spell Checker radio buttons.

languageAttribute

Specifies which attributes specify the language of an element and all its descendants. This is typically `xml:lang` or `lang` (or both in the case of XHTML).

Default: there is no such attribute; see language lookup [102].

defaultLanguage

Specifies the default language of a document of that type. (This option is rarely used.)

Default: no default language; see language lookup [102].

### Note

XMLmind XML Editor determines the language of an element by examining, in that order:

1. the value of any of the attributes specified by option `languageAttribute`. Note that the attribute lookup starts at current element and ends at the root element of the document,
2. the value of option `defaultLanguage`,
3. the value selected in the Default language combobox of the Spell tool.

checkComments

Specifies whether comments must be checked for spelling.

Default: do not check comments.

checkedProcessingInstructions

Specifies the targets of processing instructions which must be checked for spelling. May be an empty list, which means: do not check processing instructions.

Default: do not check processing instructions.

checkedAttributes

Specifies the XPaths (subset [75]) of attributes which must be checked for spelling. May be an empty list, which means: do not check attributes.

For efficiency reasons, an XPath whose last step does not test an attribute name is ignored. For example, "`foo/@*`" is ignored.

Default: do not check attributes.

skippedElements

Specifies the XPaths (subset [75]) of elements which must be automatically skipped by the spell checker. May be an empty list, which means: do not skip any element.

For efficiency reasons, an XPath whose last step does not test an element name is ignored. For example, "`foo//*`" is ignored.

Default: do not skip any element.

Examples (DocBook V4, XHTML):

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

<cfg:spellCheckOptions xmlns:html="http://www.w3.org/1999/xhtml"
  useAutomaticSpellChecker="true"
  languageAttribute="xml:lang lang"
  skippedElements="html:pre html:style html:script" />
```

Note that a `spellCheckOptions` element does not replace the `spellCheckOptions` element previously found in a configuration file. When a configuration file contains several `spellCheckOptions` elements, these `spellCheckOptions` elements are merged.

Example:

```
<cfg:spellCheckOptions xmlns=""
    useAutomaticSpellChecker="true"
    languageAttribute="xml:lang lang"
    skippedElements="html:pre html:script" />
        .
        .
        .
<cfg:spellCheckOptions xmlns=""
    languageAttribute="xml:lang"
    defaultLanguage="en-US"
    checkComments="true"
    checkedProcessingInstructions="annotation remark"
    checkedAttributes="@alt html:table/@summary html:table/@title" />
```

is equivalent to:

```
<cfg:spellCheckOptions xmlns=""
    useAutomaticSpellChecker="true"
    languageAttribute="xml:lang"
    defaultLanguage="en-US"
    checkComments="true"
    checkedProcessingInstructions="annotation remark"
    checkedAttributes="@alt html:table/@summary html:table/@title"
    skippedElements="html:pre html:script">
```

## 27. spreadsheetFunctions

```
<spreadsheetFunctions
  location = anyURI
/>
```

Specifies the location of an XML document containing user-defined spreadsheet functions.

This XML document contains the definitions of the functions (as Java™ class names or directly using the formula language) as well as their documentations (for online use in the Formula Editor).

This XML document must conform to the `http://www.xmlmind.com/xmleditor/schema/spreadsheet/functions` W3C XML Schema. A complete XXE configuration for writing such documents is found in *`XXE_install_dir`*`/doc/configure/functions_config/`.

Specify `spreadsheetFunctions` in customize.xxe to add general purpose spreadsheet functions.

Specify `spreadsheetFunctions` in XML application specific XXE configuration files (example: `invoice.xxe`) if you want make your spreadsheet functions visible only when certain types of documents (example: `Invoices`) of are opened.

Adding user-defined spreadsheet functions to XXE is extensively described in XMLmind XML Editor - Using the Integrated Spreadsheet Engine.

## 28. template

```
<template
  name = non empty token
  location = anyURI
  category = one or more category segments separated by '/' : configuration name
  order = int : 100
/>
```

Add document template named `name`, contained in file `location`, to the dialog box displayed by the File → New dialog box.

Specifying a `template` element without a location may be used to remove `template` element having the same name and the same category from this category.

Optional attributes `category` and `order` allow to better organize the content of the File → New dialog box.

category
: Specifies 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.

order
: Specifies the relative order of the document template within its category. Default value is 100.

Example 1:

```
<template name="Slides"
          location="template/slides.xml" />
```

The above template is specified in the configuration named "`Slides`", hence its category is by default "`Slides`" and its order is by default 100.

Example 2:

```
<template name="Map" location="template/v1.1/dtd/map.ditamap"
          category="DITA/1.1" order="100" />

<template location="template/v1.1/dtd/template.ditaval" name="DITAVAL"
          category="DITA/1.1" order="1000" />
```

The first above template is specified in the configuration named "`DITA Map`" and the second one in the configuration named "`DITAVAL`". Despite the fact that the two above templates are specified in different configurations, the File → New dialog box will display them in the same category "`DITA/1.1`" and template "`DITAVAL`" will follow template "`Map`".

Example 3:

```
<template name="DITAVAL" category="DITA/1.1"/>
```

Remove template "`DITAVAL`" from category "`DITA/1.1`".

## Specifying composite document templates

Composite document templates, that is, modular document templates and/or document templates referencing graphics files, must be packaged in a `.zip` archive. Example: `modular_book.zip`:

```
$ unzip -v modular_book.zip
modular_book.xml
chapter1.xml
chapter2.xml
chapter3.xml
appendix.xml
images/
images/xmlmind.gif
```

The master document, `modular_book.xml` in the above example:

1. Must be directly contained in the archive (that is, not in a subdirectory like `images/`),
2. Must have the same basename, extension not included, as the archive. The basename, less the extension, is "`modular_book`" in the above example.

# 29. toolBar

```
<toolBar
  name = NMTOKEN
  insert = non empty token
>
  Content: [ separator | button | insert ]*
</toolBar>

<separator />

<insert />

<button
  icon = anyURI
  toolTip = non empty token
>
  Content: [ class [ property ]* ]? command | menu
</button>

<class>
  Content: Java class name
</class>

<property
  name = NMTOKEN matching [_a-zA-Z][_a-zA-Z0-9]*
  type = (boolean|byte|char|short|int|long|float|double|
          String|URL)
  value = string
/>

<command
  name = NMTOKEN (optionally preceded by a command namespace [68])
  parameter = string
/>

<menu>
  Content: [ item | separator ]+
</menu>

<item
  label = non empty token
  icon = anyURI
  command = NMTOKEN (optionally preceded by a command namespace [68])
  parameter = string
/>
```

Add buttons specified in this element to the tool bar.

Example:

```
<toolBar>
  <button toolTip="Convert to emphasis"
          icon="../icons2/emphasis_menu.gif">
    <menu>
      <item label="emphasis" command="convert"
            parameter="[implicitElement] emphasis" />
      <separator />
      <item label="literal" command="convert"
            parameter="[implicitElement] literal" />
    </menu>
  </button>

   <button toolTip="Convert to plain text" icon="../icons2/plain.gif">
    <command name="convert" parameter="[implicitElement] #text" />
  </button>

  <separator />

  <button toolTip="Add para" icon="../icons2/para.gif">
    <command name="add" parameter="after[implicitElement] para" />
  </button>
</toolBar>
```

## Extending a previously defined tool bar

There are two ways to extend previously defined tool bar:

1. By using the `insert` child element.

2. By using the `insert` attribute.

Both methods cannot be used in the same `toolBar` element. Method 1 is faster and simpler to use than method 2.

1. Using the `insert` child element. Example:

   ```
   <include location="../common/common.incl" />
   <!-- ====================================================
   Let's suppose this tool bar is defined in common.incl:

   <toolBar>
     <button toolTip="Convert to b" icon="../common/icons2/b.gif">
       <command name="convert" parameter="[implicitElement] b" />
     </button>
   </toolBar>
   ===================================================== -->

   <toolBar>
     <button toolTip="Convert to i" icon="../common/icons2/i.gif">
       <command name="convert" parameter="[implicitElement] i" />
     </button>
     <insert />
     <button toolTip="Convert to tt" icon="../common/icons2/tt.gif">
       <command name="convert" parameter="[implicitElement] tt" />
     </button>
   </toolBar>
   ```

   The `insert` child element is a directive which means: insert all the buttons of the previous definition of the same tool bar here.

2. Using the `insert` attribute. Example:

```
<include location="../common/common.incl" />
<!-- ====================================================
Let's suppose this tool bar is defined in common.incl:

<toolBar>
  <button toolTip="Convert to i" icon="../common/icons2/i.gif">
    <command name="convert" parameter="[implicitElement] i" />
  </button>
  <button toolTip="Convert to tt" icon="../common/icons2/tt.gif">
    <command name="convert" parameter="[implicitElement] tt" />
  </button>
</toolBar>
==================================================== -->

<toolBar insert="Convert to tt">
  <button toolTip="Convert to b" icon="../common/icons2/b.gif">
    <command name="convert" parameter="[implicitElement] b" />
  </button>
</toolBar>
```

The `insert` attribute is a directive which means: insert all the buttons found in this tool bar into the previous definition of the same tool bar, and this, at specified position.

The value of the `insert` attribute is the `toolTip` of a `button` found in the previous definition of the same tool bar. If desired position is a button having no `toolTip` attribute, it is possible to use the basename of its icon (e.g. `"para.gif"` for `icon="../icons2/para.gif"`).

This tool tip (or icon basename) may be preceded by modifier "`before `" or by modifier "`after `". Modifier "`before `" is the implicit one.

In the above example, extending the tool bar could have also been achieved by using:

```
<toolBar insert="before Convert to tt">
  <button toolTip="Convert to b" icon="../common/icons2/b.gif">
    <command name="convert" parameter="[implicitElement] b" />
  </button>
</toolBar>
```

or by using:

```
<toolBar insert="after Convert to i">
  <button toolTip="Convert to b" icon="../common/icons2/b.gif">
    <command name="convert" parameter="[implicitElement] b" />
  </button>
</toolBar>
```

Alternatively, the value of the `insert` attribute may be `##first` or `##last`. Value `##first` specifies the first button of the previous definition of the same tool bar. Value `##last` specifies the last button of the previous definition of the same tool bar. Example:

```
<toolBar insert="before ##last">
  <button toolTip="Convert to b" icon="../common/icons2/b.gif">
    <command name="convert" parameter="[implicitElement] b" />
  </button>
</toolBar>
```

The value of the `insert` attribute may start with `ifDefined(`*`system_property_name`*`)`. In such case, the previously defined tool bar is extended if and only if a system property called *`system_property_name`* has been defined (no matter its value). Example:

```
<toolBar insert="ifDefined(XXE.Feature.Spreadsheet) before ##last">
  ...
</toolBar>
```

## 29.1. Custom controls

Normally the above specification is used to create a normal button, either directly invoking a command or displaying a menu where each item invokes a different command. However, if the first child element of a `button` element is a `class` element, a custom control is created rather than a normal button.

Note that a custom control created this way interprets the attributes (`icon`, `toolTip`, etc) and the other child elements of its `button` parent (`command`, `menu`) in a specific way. These specificities must be documented separately for each type of custom control.

class
: Must be a class which extends `java.awt.Component` and which implements the `com.xmlmind.xmleditapp.kit.part.toolbar.ToolBarTool` interface.

property
: One or more `property` elements may be used to parametrize the newly created custom component. See bean properties [76].

DocBook 4 examples:

```
<button toolTip="emphasis"
        icon="xxe-config:common/icons/italic.png">
  <class>com.xmlmind.xmleditapp.kit.part.toolbar.TextStyleToggle</class>

  <command name="pass" parameter="emphasis" />
</button>

<button toolTip="link"
        icon="xxe-config:common/icons/hyperText_menu.png">
  <class>com.xmlmind.xmleditapp.kit.part.toolbar.TextStyleToggle</class>
  <property name="toggleShowsActiveTextStyle" type="boolean" value="true" />

  <menu>
    <item label="link" icon="xxe-config:common/icons/hyperText.png"
          command="pass" parameter="link[@linkend]" />
    <item label="ulink" icon="xxe-config:common/icons/link.png"
          command="pass" parameter="ulink[@url]" />
  </menu>
</button>
```

### 29.1.1. The `TextStyleMenu` custom control

This custom control consists in a button which displays a menu containing checkboxes. Each checkbox toggles a different *text style*. More information about text style toggles in About “text style” toggles in *XMLmind XML Editor - Online Help*.

Class name: `com.xmlmind.xmleditapp.kit.part.toolbar.TextStyleMenu`

**Table 7.1. `TextStyleMenu` properties**

| Property | Type | Default value | Description |
|---|---|---|---|
| customizationId | String | - | If set, a Customize item is added at the end of the menu. The Customize item displays a dialog box allowing to quickly customize the entries of the menu. More information in Section 14, “Dialog box allowing to edit “text style” menu items” in *XMLmind XML Editor - Online Help*.<br><br>The value of this property is a string which must be chosen in order to be unique within all the customizationId values of all the `toolBar` custom controls of all XXE configurations. |

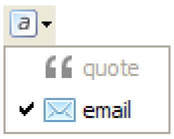
This custom control is specified similarly to a normal `toolBar button` element containing a `menu` child element (see Section 29, “toolBar” [105]), except that:

1. The first child of the `button` element must be a `class` element containing `com.xmlmind.xmleditapp.kit.part.toolbar.TextStyleMenu`.
2. The `item/@command` attributes are completely ignored. For example, you may specify **pass** or **alert**.
3. The `item/@parameter` attributes must contain a specification of a text style. DocBook 4 examples: `emphasis`, `link[@linkend]`, `sgmltag[@class="element"]`. Text style specification is documented in Section 98, “toggleTextStyle” in *XMLmind XML Editor - Commands*.

In the following DocBook 5 examples, the caret is found inside an `email` element. That's why the email checkbox is checked.

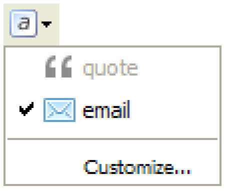
### Example 7.2. Simplest `TextStyleMenu` custom control

```
<button toolTip="Miscellaneous text styles"
        icon="xxe-config:common/icons/emphasisText_menu.png">
  <class>com.xmlmind.xmleditapp.kit.part.toolbar.TextStyleMenu</class>

  <menu>
    <item label="quote"
          icon="xxe-config:common/icons/quote.png"
          command="pass"
          parameter="{http://docbook.org/ns/docbook}quote" />
    <item label="email"
          icon="xxe-config:common/icons/email.png"
          command="pass"
          parameter="{http://docbook.org/ns/docbook}email" />
  </menu>
</button>
```



### Example 7.3. Customizable `TextStyleMenu` custom control

```
<button toolTip="Miscellaneous text styles"
        icon="xxe-config:common/icons/emphasisText_menu.png">
  <class>com.xmlmind.xmleditapp.kit.part.toolbar.TextStyleMenu</class>
  <property name="customizationId" type="String"
            value="db5.miscTextStyles" />

  <menu>
    <item label="quote"
          icon="xxe-config:common/icons/quote.png"
          command="pass"
          parameter="{http://docbook.org/ns/docbook}quote" />
    <item label="email"
          icon="xxe-config:common/icons/email.png"
          command="pass"
          parameter="{http://docbook.org/ns/docbook}email" />
  </menu>
</button>
```



### 29.1.2. The `TextStyleToggle` custom control

This custom control is a variant of the `TextStyleMenu` custom control [108]. This custom control combines a toggle button and a plain button having an arrow icon. The arrow button displays a menu containing checkboxes. Each checkbox toggles a different *text style*. More information about text style toggles in About “text style” toggles in *XMLmind XML Editor - Online Help*. By default, the toggle button toggles the first text style of the menu. Therefore, this toggle button may be considered to be a “quick access” to the first entry of the menu. Note that when the menu contains a single entry, the arrow button —which is not useful in this case— is automatically suppressed.

Class name: `com.xmlmind.xmleditapp.kit.part.toolbar.TextStyleToggle`

**Table 7.2. `TextStyleToggle` properties**

| Property | Type | Default value | Description |
|---|---|---|---|
| customizationId | String | - | If set, an arrow button displaying a menu is always created and a Customize item is added at the end of the menu. The Customize item displays a dialog box allowing to quickly customize the entries of the menu. More information in Section 14, “Dialog box allowing to edit “text style” menu items” in *XMLmind XML Editor - Online Help*.<br><br>The value of this property is a string which must be chosen in order to be unique within all the customizationId values of all the `toolBar` custom controls of all XXE configurations. |
| toggleShowsLabel | boolean | false | If set to true, the toggle button shows a label, possibly in addition to an icon. |
| toggleShowsActive-TextStyle | boolean | false | By default, the toggle button is simply a “quick access” to the first entry of the menu. When this property is set to true, the toggle button becomes a quick access to the entry of the menu which is checked. If there is no checked checkbox in the menu, then the toggle button is a quick access to the first entry of the menu.<br><br>**Important**<br><br>Do not set this property to true unless each menu item has a different icon and/or you also set toggleShowsLabel to true.<br><br>If you don't follow this recommendation, the user of XXE will probably not understand the behavior of `TextStyleToggle`. |

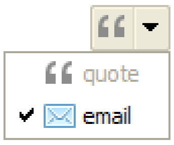
This custom control is specified similarly to a normal `toolBar` `button` element containing a `command` or a `menu` child element (see Section 29, “toolBar” [105]), except that:

1. The first child of the `button` element must be a `class` element containing `com.xmlmind.xmleditapp.kit.part.toolbar.TextStyleToggle`.
2. The `command/@name` and `item/@command` attributes are completely ignored. For example, you may specify **pass** or **alert**.
3. The `command/@parameter` and `item/@parameter` attributes must contain a specification of a text style. DocBook 4 examples: `emphasis`, `link[@linkend]`, `sgmltag[@class="element"]`. Text style specification is documented in Section 98, “toggleTextStyle” in *XMLmind XML Editor - Commands*.

In the following DocBook 5 examples, the caret is found inside an `email` element. That's why the email checkbox is checked and, for some examples, the toggle button is selected and shows an envelope icon.

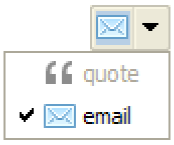
## Example 7.4. Simplest `TextStyleToggle` custom control

```
<button toolTip="Miscellaneous text styles"
        icon="xxe-config:common/icons/emphasisText_menu.png">
  <class>com.xmlmind.xmleditapp.kit.part.toolbar.TextStyleToggle</class>

  <menu>
    <item label="quote"
          icon="xxe-config:common/icons/quote.png"
          command="pass"
          parameter="{http://docbook.org/ns/docbook}quote" />
    <item label="email"
          icon="xxe-config:common/icons/email.png"
          command="pass"
          parameter="{http://docbook.org/ns/docbook}email" />
  </menu>
</button>
```



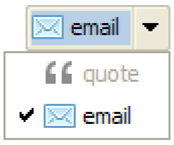
## Example 7.5. “Active” `TextStyleToggle` custom control

```
<button toolTip="Miscellaneous text styles"
        icon="xxe-config:common/icons/emphasisText_menu.png">
  <class>com.xmlmind.xmleditapp.kit.part.toolbar.TextStyleToggle</class>
  <property name="toggleShowsActiveTextStyle" type="boolean" value="true" />

  <menu>
    <item label="quote"
          icon="xxe-config:common/icons/quote.png"
          command="pass"
          parameter="{http://docbook.org/ns/docbook}quote" />
    <item label="email"
          icon="xxe-config:common/icons/email.png"
          command="pass"
          parameter="{http://docbook.org/ns/docbook}email" />
  </menu>
</button>
```



## Example 7.6. “Active” `TextStyleToggle` custom control showing a label in addition to an icon

```
<button toolTip="Miscellaneous text styles"
        icon="xxe-config:common/icons/emphasisText_menu.png">
  <class>com.xmlmind.xmleditapp.kit.part.toolbar.TextStyleToggle</class>
  <property name="toggleShowsActiveTextStyle" type="boolean" value="true" />
  <property name="toggleShowsLabel" type="boolean" value="true" />

  <menu>
    <item label="quote"
          icon="xxe-config:common/icons/quote.png"
          command="pass"
          parameter="{http://docbook.org/ns/docbook}quote" />
    <item label="email"
          icon="xxe-config:common/icons/email.png"
          command="pass"
          parameter="{http://docbook.org/ns/docbook}email" />
  </menu>
</button>
```



## 29.2. Multiple toolBars

Specifying a `name` attribute for the `toolBar` element allows to create a GUI having several XML application specific tool bars.

Example:

1. In *`XXE_user_preferences_dir`*`/addon/xhtml.xxe`, add something like this:

   ```
   <toolBar name="toolBar2">
     ...
   </toolBar>
   ```

2. In *`XXE_user_preferences_dir`*`/addon/docbook.xxe`, add something like this:

   ```
   <toolBar name="toolBar2">
     ...
   </toolBar>
   ```

   Notice that the *same* name `toolBar2` is used in all XML application specific configuration files.

3. In *`XXE_user_preferences_dir`*`/addon/customize.xxe_gui` (see XMLmind XML Editor - Customizing the User Interface), add something like this:

   ```
   <toolBarItems name="configSpecificToolBarItems2">
     <class>com.xmlmind.xmleditapp.kit.part.ConfigSpecificToolBarItems</class>
     <property name="specificationName" type="String" value="toolBar2" />
   </toolBarItems>

   <toolBar name="configSpecificToolBar2">
     <toolBarItems name="configSpecificToolBarItems2" />
   </toolBar>

   <layout>
     <topToolBars>
       <insert />
       <toolBar name="configSpecificToolBar2" />
     </topToolBars>
   </layout>
   ```

# 30. translation

```
<translation
  location = anyURI matching [path/]resourcename_lang.properties
/>
```

Specifies how to translate messages found in menu [90] `item label`, toolBar [105] `button toolTip`, template [104] `name`, elementTemplate [77] `name`, css [70] `name`, binding [61] `menu item label`, etc.

Localizing configuration files works as follows:

1. The `location` attribute points to a Java™ property file. XHTML example:

   ```
   <translation location="xhtml_en.properties" />
       ...
   ```

```
  <item label="Pre_view" icon="../common/icons/Refresh16.gif"
        command="xhtml.preview">
    <accelerator code="F5" />
  </item>
</menu>
    ...
```

Where `xhtml_en.properties` contains:

```
     ...
preview=Pre_view
convertToI=Convert to i
convertToB=Convert to b
     ...
```

The location URL specifies:

- The reference language of the configuration file: a two-letter lower-case ISO code. In the above example: `en`.
- A unique resource name used to find translations to other languages. In the above example: `xhtml`. More on this below.

The reference property file is only used to map messages to message IDs. Example: message "`Convert to i`" has ID "`convertToI`".

2. If, for example, XXE is started using a French locale, a property file called `xhtml_fr.properties`:

   - is searched in the same directory as the reference property file;
   - OR, if this file is not found there, this property file is searched as a resource using the `CLASSPATH`. That is, `xhtml_fr.properties` is supposed to be contained[3] in a `jar` file found in the `CLASSPATH`.

   For performance reasons, language variants such `CA` in `fr-CA` are not supported.

3. For the localization to work, the translated property file must refer to the same IDs as those found in reference property file.

   For example, `xhtml_fr.properties` contains:

```
     ...
preview=Prévisualiser
convertToI=Convertir en i
convertToB=Convertir en b
     ...
```

## 31. validate

```
<validate
  namespace = non empty anyURI
>
  Content: dtd|schema|relaxng
</validate>
```

Dynamically compose the auxiliary schema specified in this element with the main schema specified in the document itself (e.g. `<!DOCTYPE>`) or, in absence of such specification, with the main schema specified using the DTD [71], schema [99] or relaxng [96] configuration element.

[3] *Directly* contained, and not contained in a ``folder''. That is, "`jar tvf foo.jar`" must display `xhtml_fr.properties` and not `foo/bar/xhtml_fr.properties`.

More precisely, this element means: whenever you find an XML subtree having a root element belonging to the namespace specified using the `namespace` attribute, use specified schema rather than the content model specified in the main schema.

This facility is meant to be used to validate ``alien subtrees", for example SVG or MathML subtrees found in XHTML, DocBook or DITA documents. A well-designed main schema generally specifies a very loose content model for such alien elements. Example: `<!ELEMENT mml:math ANY>`.

It is possible to compose schemas of different kinds. For example, it is possible to compose the main DITA DTD with a RELAX NG auxiliary schema.

It is possible to specify several `validate` configuration elements, each element having of course a different `namespace` attribute.

Example: Validate XML subtree having a root element belonging to the "`http://www.w3.org/1998/Math/MathML`" namespace using the "`rng/mathml2.rng`" RELAX NG schema.

```
  <validate namespace="http://www.w3.org/1998/Math/MathML">
    <relaxng location="rng/mathml2.rng" />
  </validate>
```

## 32. validateHook

```
<validateHook
  name = non empty token
>
  Content: [ class [ property ]* ]?
</validateHook>

<class>
  Content: Java class name
</class>

<property
  name = NMTOKEN matching [_a-zA-Z][_a-zA-Z0-9]*
  type = (boolean|byte|char|short|int|long|float|double|
          String|URL)
  value = string
/>
```

Register `validateHook` specified by *`class`* with XXE.

A `validateHook` is some code notified by XXE before and after a document is checked for validity.

This is a very general mechanism which has been created to perform semantic validation beyond what can be done using a DTD or a schema alone.

Child elements of `validateHook`:

class
: Register `validateHook` implemented in the Java™ language by class *`class`* (implements interface `com.xmlmind.xmleditapp.validatehook.ValidateHook` -- SeeChapter 6, *Writing a validateHook* in *XMLmind XML Editor - Developer's Guide*).

property
: `Property` child elements may be used to parametrize a newly created validateHook. See bean properties [76].

Attributes of `validateHook`:

name
: This name is useful to remove or replace a previously registered `validateHook`. Anonymous `validateHooks` cannot be removed or replaced.

When a `validateHook` element is used to remove a registered `validateHook`, a `name` attribute must be specified and there must be no `class` child element.

Example: In this example, a Java™ class named `com.xmlmind.xmleditext.docbook.table.ValidateHookImpl` is contained in `docbook.jar` (among other DocBook commands and extensions).

```
<validateHook>
  <class>com.xmlmind.xmleditext.docbook.table.ValidateHookImpl</class>
</validateHook>
```

A `validateHook` is always specific to a document type.

For example, the DocBook `validateHook` is used to fix the `cols` attribute of `tgroups` and `entrytbls` (if needed to) just before a DocBook document is checked for validity.

These `validateHooks` are specified in the XXE configuration file associated to the document type. For example, the DocBook `validateHook` is specified in `docbook.xxe`.

Several `validateHooks` can be associated to the same document type. In such case, they are notified in the order of their registration.

## 33. windowLayout

```
<windowLayout>
  Content (in any order): center [ top ]? [ bottom ]?
                          [ left ]? [ right ]?
</windowLayout>

<center
  css = non empty token
/>

<top
  css = non empty token
  size = double between 0 and 1 exclusive : 0.25
/>

<bottom
  css = non empty token
  size = double between 0 and 1 exclusive : 0.25
/>

<left
  css = non empty token
  size = double between 0 and 1 exclusive : 0.25
/>

<right
  css = non empty token
  size = double between 0 and 1 exclusive : 0.25
/>
```

By default, XXE creates a single view when a document is opened. This view is the tree view if no CSS style sheets are available for the opened document. This view is a styled view using first non-alternate CSS style sheet if one or more style sheets are available for the opened document.

The windowLayout element allows to force XXE to automatically create several views for the same document when this document is opened. This is similar to using menu item View → Add except that these actions have been automated.

Child elements `center`, `top`, `bottom`, `left`, `right` specify which view to add and where it is added. Note that having a `center` child element is required.

The `css` attribute of these child elements specify which CSS style sheet to use. An absent `css` attribute means that a tree view is to be used.

The `size` attribute of the four ``border views": `top`, `bottom`, `left`, `right`, specify the proportional size of the view. For example: <top.size="0.25"/> means that a tree view will occupy one fourth of the available height and that this tree view will be found above the central, main view.

Two DocBook examples:

```
<windowLayout>
  <center css="DocBook" />
  <bottom css="Document structure" size="0.15" />
</windowLayout>

<windowLayout>
  <left />
  <top css="Document structure" />
  <center css="DocBook" />
</windowLayout>

<css name="DocBook" location="css/docbook.css" />
<css name="Document structure" alternate="true"
     location="css/structure.css" />
<css name="Show info about included elements" alternate="true"
     location="css/visible_inclusions.css" />
```