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>

Published June 21, 2007

Abstract

This document describes how to customize and deploy XXE.

Part I. Guide

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 an XML 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 XML 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 and DocBook[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] DITA, Simplified DocBook, Slides, etc, are available as add-ons.

Chapter 2. Writing a configuration file for XXE

A configuration file is an XML file[1] that customizes XXE for a specific XML application. XXE is bundled with configurations for a few XML applications: DocBook, XHTML, W3C XML Schema, etc. More configurations (e.g. DITA) are available but they need the user to download and install the corresponding add-on[2].

This section describes how to write a configuration for a custom DTD, for a custom W3C XML Schema and for a custom RELAX NG schema. What is described is the recommended way of doing things. This being said, it is also possible to use XXE without writing any configuration file for a custom XML application: see side bar below.

The configurations used as examples are *minimal* configurations. The following configuration items are not described in this section:

- Named element templates. See elementTemplate [46].
- Custom commands implemented in the Java™ language. See command [36].
- Macro commands. See command [36].
- Menu bar menu. See menu [53].
- Tool bar buttons. See toolBar [66].
- Popup menus. See binding [32].
- Mouse and/or keyboard bindings. See binding [32].

Please read Configuration elements [32] if you need to use any of these customization items.

The configurations used as examples are found in *`XXE_install_dir`*`/doc/configure/samples/example1/`, `example2/`, `example3/`.

Another configuration, using W3C XML schemas like example2 but much more comprehensive, is found in *`XXE_install_dir`*`/doc/configure/samples/imagedemo/`. This configuration has been created to explain how to cope with XML documents containing *embedded* binary (i.e. TIFF, PNG, etc) or XML (i.e. SVG) images. However, it is also useful as an example of an XXE configuration.

[1] Conforming to W3C XML Schema `configuration.xsd` available after downloading and installing add-on called "*A configuration for specifying XMLmind XML Editor configurations*".
[2] Simply use Options|Install Add-ons for that.

Q: How to use XXE without writing a configuration file for my XML application?

A: Use File|Open As Template and select an existing document each time you need to create a new document of the same type. See the section called “File menu” in *XMLmind XML Editor - Online Help* for a description of this command.

If you want to use the styled view, the document selected for use by File|Open As Template must contain one or several `<?xml-stylesheet?>` processing instructions.

This processing instruction is specified in the W3C recommendation Associating Style Sheets with XML Documents.

Example of document intended to be ``opened as template'':

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/css" alternate="yes" title="Big fonts"
href="http://www.xmlmind.com/css/example1b.css" ?>
<?xml-stylesheet type="text/css" alternate="yes" title="Important things in red"
href="http://www.xmlmind.com/css/example1r.css" ?>
<?xml-stylesheet type="text/css"
href="http://www.xmlmind.com/css/example1.css" ?>
<!DOCTYPE doc PUBLIC "-//XMLmind//DTD Example1//EN"
"http://www.xmlmind.com/dtd/example1.dtd">
<doc>
  <para>Paragraph 1.</para>
  <para>Paragraph 2.</para>
  <para>Paragraph 3.</para>
</doc>
```

Using XXE this way works fine but really requires you to specify absolute URLs for the DTD and CSS in the ``template''.

1. DTD example

1. Create a subdirectory named `example1` in the `addon/` subdirectory of XXE user preferences directory.

   XXE user preferences directory is:

   - *`$HOME`*`/.xxe/` on Linux, Mac, and more generally, on Unix.
   - *`%APPDATA%`*`\XMLmind\XMLeditor\` on Windows 2000, XP, Vista.

     Example: `C:\Documents and Settings\john\Application Data\XMLmind\XMLeditor\` on Windows 2000 and XP. `C:\Users\john\AppData\Roaming\XMLmind\XMLeditor\` on Windows Vista.

   Next chapter [21] explains how to create a configuration which can be shared with other users. For now suffice to know that this personal `addon/` directory is recursively scanned by XXE during its startup in order to load all files ending with "`.xxe`". (This also means that you are free to organize this subdirectory like you want.)

2. Copy `example1.dtd` to directory `addon/example1/`.

   ```
   <!ELEMENT doc (para+)>
   <!ELEMENT para (#PCDATA)>
   <!ATTLIST para align (left|center|right) "left">
   ```

3. Copy `example1.css` to directory `addon/example1/`.

   ```
   doc,
   para {
       display: block;
   }
   para {
   ```

```
    margin: 1ex 0;
}
para[align] {
    text-align: concatenate(attr(align));
}
```

4. Create a document template for DTD "`-//XMLmind//DTD Example1//EN`" using a text editor. Save it as `addon/example1/example1.xml`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE doc PUBLIC "-//XMLmind//DTD Example1//EN"
  "http://www.xmlmind.com/public/dtd/example1.dtd">
<doc>
  <para></para>
</doc>
```

   It is highly recommended to use a public, absolute, URL such as "`http://www.xmlmind.com/public/dtd/example1.dtd`" rather than relative URL "`example1.dtd`".

5. Using a text editor, create a XML catalog where public ID "`-//XMLmind//DTD Example1//EN`" is associated to local file `example1.dtd`. Save it as `addon/example1/example1_catalog.xml`.

```
<?xml version="1.0" ?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
         prefer="public">
  <public publicId="-//XMLmind//DTD Example1//EN"
          uri="example1.dtd"/>
</catalog>
```

   This catalog will spare XXE the effort of downloading DTD `example1.dtd` from `http://www.xmlmind.com/public/dtd/example1.dtd`.

6. Create a configuration file for XXE using XXE itself. Save it as `addon/example1/example1.xxe`.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<configuration name="Example1"
  xmlns="http://www.xmlmind.com/xmleditor/schema/configuration"
  xmlns:cfg="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>
```

   If you create a configuration file with a text editor, do not forget to check its validity before deploying it because, for performance reasons, XXE does not thoroughly validates its configuration files at start-up time. The simplest way to do that is to open the configuration file in XXE.

7. Restart XXE.

   Now you can use File|New and select Example1 > Template to create a new document.

   ## Important

   Do not forget to temporarily disable the Schema cache (using Options|Preferences, Schema tab, Enable cache toggle) if you intend to develop your own DTD and test it using XXE.

8. Make sure that the template document is valid: the red icon must *not* be displayed at the bottom/left of XXE window.

If the template document, `example1.xml`, is invalid, please use a text editor and fix it because XXE is not designed to be comfortable to use with invalid documents.

Short description of `addon/example1/example1.xxe`. See Configuration elements [32] to have more details.

- configuration [37]: The configuration file must have a name that ends with "`.xxe`" and the configuration element must have a name attribute and must contain a detect [39] element in order to be loaded by XXE.

  Configuration files without a name and/or without a detect element are typically included by other configuration files, see include [51]. To speed up the start up of XXE, it is recommended to use another suffix such as "`.incl`" to name these files.

- detect [39]: Simplest possible detection condition for a DTD based document: if a document opened by XXE has a `<!DOCTYPE>` with public ID equals to `-//XMLmind//DTD Example1//EN`, then XXE will automatically use configuration `addon/example1/example1.xxe`.

- css [38]: If a document detected by Example1 configuration has no `<?xml-stylesheet?>` processing instruction specifying a CSS style sheet, XXE will automatically use `addon/example1/example1.css`.

- template [65]: Entry Example1 > Template is listed in the File|New dialog box. Selecting this entry allows you to create a new document with the `-//XMLmind//DTD Example1//EN` document type.

2. W3C XML Schema example

The W3C XML Schema example is similar to the DTD example.

1. Create a subdirectory named `example2` in the `addon/` subdirectory of XXE user preferences directory [4]:

2. Copy `example2.xsd` to directory `addon/example2/`.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<xs:schema elementFormDefault="qualified"
targetNamespace="http://www.xmlmind.com/xmleditor/schema/example2"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:e2="http://www.xmlmind.com/xmleditor/schema/example2">
  <xs:element name="doc">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="e2:Para" maxOccurs="unbounded" name="para"
        minOccurs="1"></xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="Para" mixed="true">
    <xs:attribute default="left" name="align" type="e2:Align"></xs:attribute>
  </xs:complexType>

  <xs:simpleType name="Align">
    <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="left"></xs:enumeration>
      <xs:enumeration value="center"></xs:enumeration>
      <xs:enumeration value="right"></xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

3. Copy `example2.css` to directory `addon/example2/`.

```
@namespace url(http://www.xmlmind.com/xmleditor/schema/example2);

doc,
para {
    display: block;
```

```
}
para {
    margin: 1ex 0;
}
para[align] {
    text-align: concatenate(attr(align));
}
```

This style sheet would work fine without default namespace declaration at the top of it but rule matching is faster when `@namespace` is used.

4. Create a document template for XML Schema "`http://www.xmlmind.com/xmleditor/schema/example2`" using a text editor. Save it as `addon/example2/example2.xml`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<doc xmlns="http://www.xmlmind.com/xmleditor/schema/example2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xmlmind.com/xmleditor/schema/example2
                      http://www.xmlmind.com/public/schema/example2.xsd">
  <para></para>
</doc>
```

It is highly recommended to use a public, absolute, URL such as "`http://www.xmlmind.com/public/schema/example2.xsd`" rather than relative URL "`example2.xsd`".

5. Using a text editor, create a XML catalog where URL "`http://www.xmlmind.com/public/schema/example2.xsd`" is associated to local file `example2.xsd`. Save it as `addon/example2/example2_catalog.xml`.

```
<?xml version="1.0" ?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
         prefer="public">
  <uri name="http://www.xmlmind.com/public/schema/example2.xsd"
       uri="example2.xsd"/>
</catalog>
```

This catalog will spare XXE the effort of downloading W3C XML Schema `example2.xsd` from `http://www.xmlmind.com/public/schema/example2.xsd`.

6. Create a configuration file for XXE using XXE itself. Save it as `addon/example2/example2.xxe`.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<configuration name="Example2"
  xmlns="http://www.xmlmind.com/xmleditor/schema/configuration"
  xmlns:cfg="http://www.xmlmind.com/xmleditor/schema/configuration">
  <detect>
    <rootElementNamespace
      >http://www.xmlmind.com/xmleditor/schema/example2</rootElementNamespace>
  </detect>

  <css name="Style sheet" location="example2.css" />

  <template name="Template" location="example2.xml" />
</configuration>
```

If you create a configuration file with a text editor, do not forget to check its validity before deploying it because, for performance reasons, XXE does not thoroughly validates its configuration files at start-up time. The simplest way to do that is to open the configuration file in XXE.

7. Restart XXE.

Now you can use File|New and select Example2 > Template to create a new document.

Important

Do not forget to temporarily disable the Schema cache (using Options|Preferences, Schema tab, Enable cache toggle) if you intend to develop your own schema and test it using XXE.

8. Make sure that the template document is valid: the red icon must *not* be displayed at the bottom/left of XXE window.

   If the template document, `example2.xml`, is invalid, please use a text editor and fix it because XXE is not designed to be comfortable to use with invalid documents.

About `addon/example2/example2.xxe`:

- detect [39]: Simplest possible detection condition for a XML Schema based document: if a document opened by XXE has a root element in namespace "`http://www.xmlmind.com/xmleditor/schema/example2`" then XXE will automatically use configuration `addon/example2/example2.xxe`.

3. RELAX NG example

The RELAX NG example is similar to the other examples.

1. Create a subdirectory named `example3` in the `addon/` subdirectory of XXE user preferences directory [4]:

2. Copy `example3.rnc`[3] to directory `addon/example3/`.

```
default namespace = "http://www.xmlmind.com/xmleditor/schema/example3"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

start = doc-element

doc-element = element doc {
    para-element+
}
para-element = element para {
    mixed {
        [ a:defaultValue = "left" ]
        attribute align { "left" | "center" | "right" }?
    }
}
```

3. Copy `example3.css` to directory `addon/example3/`.

```
@namespace url(http://www.xmlmind.com/xmleditor/schema/example3);

doc,
para {
    display: block;
}
para {
    margin: 1ex 0;
}
para[align] {
    text-align: concatenate(attr(align));
}
```

   This style sheet would work fine without default namespace declaration at the top of it but rule matching is faster when `@namespace` is used.

4. Create a document template for RELAX NG schema "`http://www.xmlmind.com/xmleditor/schema/example3`" using a text editor. Save it as `addon/example3/example3.xml`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<doc xmlns="http://www.xmlmind.com/xmleditor/schema/example3">
  <para></para>
</doc>
```

[3] `Example3.rng` is also available in *`XXE_install_dir`*`/doc/configure/samples/example3/`, in case you prefer the XML syntax to the compact syntax.

Note that, unlike with DTDs and with W3C XML Schemas, there is no standard way to associate a RELAX NG schema to an instance[4].

5. Create a configuration file for XXE using XXE itself. Save it as `addon/example3/example3.xxe`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration name="Example3"
  xmlns="http://www.xmlmind.com/xmleditor/schema/configuration"
  xmlns:cfg="http://www.xmlmind.com/xmleditor/schema/configuration">
  <detect>
    <rootElementNamespace
      >http://www.xmlmind.com/xmleditor/schema/example3</rootElementNamespace>
  </detect>

  <relaxng compactSyntax="true" encoding="ISO-8859-1" location="example3.rnc"
           name="http://www.xmlmind.com/xmleditor/schema/example3"/>

  <css location="example3.css" name="Style sheet"/>

  <template location="example3.xml" name="Template"/>
</configuration>
```

The `relaxng` configuration element is essential because there is no standard way to associate a RELAX NG schema to an instance.

6. Restart XXE.

Now you can use File|New and select Example3 > Template to create a new document.

Important

Do not forget to temporarily disable the Schema cache (using Options|Preferences, Schema tab, Enable cache toggle) if you intend to develop your own schema and test it using XXE.

7. Make sure that the template document is valid: the red icon must *not* be displayed at the bottom/left of XXE window.

If the template document, `example3.xml`, is invalid, please use a text editor and fix it because XXE is not designed to be comfortable to use with invalid documents.

[4]There is a non standard, proprietary, way to do that: the <?xxe-relaxng-schema location="..."?> processing instruction [44]. However, its use should be restricted to testing and other quick and dirty experiments.

Chapter 3. Customizing mouse and key bindings used by XXE

The bindings used as examples in this chapter are found in *XXE_install_dir*/doc/configure/samples2/.

1. XML application specific bindings

A configuration file such as docbook.xxe can contain binding [32] 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/common.incl.

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

In next chapter [16], we will learn how to customize an existing configuration as a whole. We will use the DocBook configuration as an example.

Important

XXE *does not allow bindings defined in XML application 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 XML application specific. 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/`*`.xxe/` on Linux, Mac, and more generally, on Unix.
- *`%APPDATA%`*`\XMLmind\XMLeditor\` on Windows 2000, XP, Vista.

  Example: `C:\Documents and Settings\john\Application Data\XMLmind\XMLeditor\` on Windows 2000 and XP. `C:\Users\john\AppData\Roaming\XMLmind\XMLeditor\` on Windows Vista.

For more information about how XXE finds its configuration files, please read Section 1, “Dynamic discovery of add-ons” [21].

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 [57], imageToolkit [48], spreadsheetFunctions [65], property [57], which are not XML application specific.

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 4. 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, parametrizable, table editor documented in the section called “A generic, parametrizable, 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

Procedure 4.1. Procedure

1. The corresponding support code is contained in *`XXE_install_dir`*`/doc/configure/jars/xhtml_table.jar`. *In theory*, you need to copy this file to the directory containing your custom configuration.

   Now, XXE is bundled with a configuration for XHTML and this configuration includes `xhtml.jar`. File `xhtml.jar` already contains all the code needed to support HTML4 tables in XXE. Therefore, unless you have deleted the standard XHTML configuration for XXE, you *don't need* to copy `xhtml_table.jar` to the directory containing your custom configuration.

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

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

3. 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.xmleditapp.xhtml.table.HTMLTableEdit</class>
   </command>
   ```

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

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/ColumnInsertBefore16.gif"
        command="my.tableEdit" parameter="insertColumnBefore"/>
  ...
```

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.
A row must be explicitly selected. 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. HTML4 form elements

What applies to HTML4 tables, also applies to HTML4 form elements (`input`, `textarea`, etc).

Procedure 4.2. Procedure

1. The corresponding support code is contained in *`XXE_install_dir`*`/doc/configure/jars/xhtml_form.jar`. *In theory*, you need to copy this file to the directory containing your custom configuration.

   Now, XXE is bundled with a configuration for XHTML and this configuration includes `xhtml.jar`. File `xhtml.jar` already contains all the code needed to style HTML4 form elements. Therefore, unless you have

deleted the standard XHTML configuration for XXE, you *don't need* to copy `xhtml_form.jar` to the directory containing your custom configuration.

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

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

If you use a namespace (e.g. `http://acme.com/ns`) for all the elements defined in your schema, including for form 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:xhtml/css/xhtml_form.imp);
```

3. CALS tables

Procedure 4.3. Procedure

1. The corresponding support code is contained in *`XXE_install_dir`*`/doc/configure/jars/cals_table.jar`. *In theory*, you need to copy this file to the directory containing your custom configuration.

Now, XXE is bundled with a configuration for DocBook and this configuration includes `docbook.jar`. File `docbook.jar` already contains all the code needed to support CALS tables in XXE. Therefore, unless you have deleted the standard DocBook configuration for XXE, you *don't need* to copy `cals_table.jar` to the directory containing your custom configuration.

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

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

3. 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.xmleditapp.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/ColumnInsertBefore16.gif"
        command="my.tableEdit" parameter="insertColumnBefore"/>
  ...
```

4. File `cals_table.jar` (and also `docbook.jar`) also contains a *document 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, saved to disk or converted to another format.

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

```
<documentHook name="cols_checker">
  <class>com.xmlmind.xmleditapp.docbook.table.DocumentHookImpl</class>
</documentHook>
```

3.1. CALS table editor command

The parameters supported by this table editor command are identical to those of the HTML4 table editor command [13].

Chapter 5. Customizing an existing configuration

A sample customization of the stock DocBook may be downloaded and installed using XXE add-on manager (Options|Install Add-ons). This configuration contains the following files and directories: `mydocbook.xxe`, `mydocbook.css`, `mydocbook.xsl`, `icons/`.

Caution

If you choose to install the above configuration in *`XXE_install_dir`*`/addon/`, then you'll have to manually rename `mydocbook.xxe` to `0mydocbook.xxe`.

If you choose to install the above configuration in *`XXE_user_preferences_dir`*`/addon/` (add-on manager's default), then there is nothing special to do.

Why? When XXE finds several configurations having the same name and when these configurations have the same priority, XXE loads the configuration having *a file basename which lexicographically precedes the others*. More information about this in next chapter [21].

- Stock `docbook.xxe` is named "`DocBook`" and custom `mydocbook.xxe` is also named "`DocBook`".
- Stock `docbook.xxe` and custom `mydocbook.xxe` being located in a subdirectory of *`XXE_install_dir`*`/addon/` are both system configurations. They have the same priority.
- `docbook.xxe` precedes `mydocbook.xxe` but `0mydocbook.xxe` precedes `docbook.xxe`.
- Renaming `mydocbook.xxe` to `0mydocbook.xxe` when `mydocbook.xxe` is installed in *`XXE_user_preferences_dir`*`/addon/` is *not* needed because user configurations have priority over system configurations.

1. Structure of a configuration file customizing an existing configuration

Important

If you create a configuration file with a text editor, do not forget to check its validity before deploying it because, for performance reasons, XXE does not thoroughly validates its configuration files at start-up time. The simplest way to do that is to open the configuration file in XXE.

Excerpt of the sample customization, `mydocbook.xxe`, of the stock DocBook configuration:

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

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

     .
    customization items
     .
     .

</configuration>
```

- The configuration file must have a "`.xxe`" extension.

- The `configuration` element must have a `name` attribute and the value of this `name` attribute must be the same as the value of the `name` attribute of the overridden configuration.
- The customized configuration must include the overridden configuration using configuration element include [51].
- If you need to refer to a file found in *`XXE_install_dir`*`/addon/config/`, it is recommended to use a relative URI which begins with string "`xxe-config:`".

  Note

  The following rule has been added to the XML catalog bundled with XXE, *`XXE_install_dir`*`/addon/config/catalog.xml`:

  ```
  <rewriteURI uriStartString="xxe-config:" rewritePrefix="." />
  ```

  This means that any URI which starts with string "`xxe-config:`" is understood as being relative to *`XXE_install_dir`*`/addon/config/`.

2. Customization items

We will describe in this section the customization items found in the `mydocbook.xxe` sample. Many more customization items not described here are available too: document templates [65], element templates [46], custom menu entries [53], etc. See Configuration elements [32].

2.1. Custom CSS style sheet

The first configuration element css [38] removes CSS style sheet named "`Show info about included elements`" from the Style menu.

The second one replaces the default CSS style sheet for DocBook (named "`DocBook`", see *`XXE_install_dir`*`/addon/config/docbook/docbook.xxe`) by a customized one.

```
<!-- Discard this CSS -->
<css name="Show info about included elements"/>

<!-- Customize the normal CSS -->
<css name="DocBook" location="mydocbook.css" />
```

The customized CSS style sheet imports the normal DocBook CSS style sheet using standard construct `@import` (but with special URI starting with "`xxe-config:`") and customizes the look and feel of DocBook element `link`.

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

link:after {
    content: set-attribute-button(attribute, linkend,
                                  icon, icon(right));
    color: rgb(128,128,196);
}
```

2.2. Custom bindings

Among other things, this customization defines a macro-command called `docb.openInDefaultViewer` and adds a custom binding for it. See configuration element binding [32].

```
<command name="docb.openInDefaultViewer">
  <macro>
    <sequence>
      <get context="$implicitElement/@url" expression="resolve-uri(.)" />
      <command name="start" parameter="helper(defaultViewer) '%_'" />
    </sequence>
  </macro>
</command>
```

```
<binding>
  <keyPressed code="F2" />
  <charTyped char="o" />
  <command name="docb.openInDefaultViewer" />
</binding>

    .
more macro-commands and bindings
    .
    .
```

A tutorial about macro-commands and the reference of all built-in commands (such as the `start` command used to define the above macro) are found in another document: XMLmind XML Editor - Commands.

2.3. Custom tool bar

This customization adds two buttons to the stock DocBook tool bar. See configuration element toolBar [66].

```
    .
definition of macro-command docb.openInDefaultViewer
    .
    .

<command name="docb.editDocument">
  <macro trace="false">
    <sequence>
      <get context="$implicitElement/@url" expression="resolve-uri(.)" />
      <command name="XXE.edit" parameter="%_" />
    </sequence>
  </macro>
</command>

<toolBar>
  <insert />
  <separator />
   <button toolTip="Open in Default Viewer [F2 o]"
          icon="icons/openInDefaultViewer.gif">
    <command name="docb.openInDefaultViewer" />
  </button>
  <button toolTip="Edit Document [F2 e]"
          icon="icons/editDocument.gif">
    <command name="docb.editDocument" />
  </button>
</toolBar>
```

Note the `insert` attribute of the `toolBar` element which is used to insert all the tool bar buttons specified in the stock DocBook configuration before the new Open in Default Viewer button.

2.4. Custom menu

This customization adds two new items, equivalent to the above tool bar buttons, to the stock DocBook menu. See configuration element menu [53].

```
<menu label="_DocBook" insert="Move _Up">
  <item label="Open in Default _Viewer"
        icon="icons/openInDefaultViewer.gif"
        command="docb.openInDefaultViewer" />
  <item label="Edi_t Document"
        icon="icons/editDocument.gif"
        command="docb.editDocument" />
  <separator />
</menu>
```

Note the `insert` attribute in the `menu` element which is used to insert the two new items and a separator before the "Move _Up" item found in the stock DocBook menu.

The `insert` child element and the `insert` attribute are the two ways to extend existing tool bars and menus.

2.5. Custom parameters for the XSLT style sheet used to convert DocBook documents to RTF, PostScript and PDF

Named parameterGroup [57]s are used here to customize the HTML files generated using sub menu Convert of the DocBook menu (see *`XXE_install_dir`*`/addon/config/docbook/xslMenu.incl`)

Which `parameterGroup`s to use for DocBook is described in another document: Customizing the XSL style sheets used by the above Convert commands in *XMLmind XML Editor - DocBook Support*.

The reference manual of Norman Walsh's DocBook XSLT style sheets (which is needed to know, for instance, what means "`chunker.output.encoding`") is found in another document: DocBook XSL Stylesheet Documentation.

```
  <!-- Use UTF-8 encoding for generated 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>
```

2.5.1. Extensively customizing the conversion process

It is also possible to *extensively* customize the **Convert** commands by specifying alternate XSLT style sheets for them.

Example: defining the following property in any XXE configuration file (see configuration element property [57]) allows to use customized XSLT style sheet `mydocbook.xsl` instead of the one normally used by the `docb.toHTML1` process command:

```
<property name="docb.toHTML1.transform" url="true">mydocbook.xsl</property>
```

The customized XSLT style sheet always imports the stock style sheet and generally redefines a few custom templates.

```
<?xml version='1.0' encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:saxon="http://icl.com/saxon"
                extension-element-prefixes="saxon">

  <xsl:import href="xxe-config:docbook/xsl/html/docbook.xsl"/>1

  <xsl:output method="html"
              encoding="UTF-8"2
              indent="no"
              saxon:character-representation="native;decimal"/>
</xsl:stylesheet>
```

In the above example (`mydocbook.xsl`), the goal is to generate single-page HTML files using the UTF-8 encoding instead of ISO-8859-1.

1 Imports the stock XSLT style sheet used to create single-page HTML files. Note the "`xxe-config:`"-style URL.

2 The above example does not redefine templates. It redefines the `xsl:output` element of the style sheet.

Another example is found in the documentation of the `process`/`transform` configuration element. See the section called “Element transform” in *XMLmind XML Editor - Commands*.

Tip

In our opinion, it is almost impossible to cope with the complexity of customizing Norman Walsh's DocBook XSLT style sheets without reading this excellent book: DocBook XSL: The Complete Guide by Bob Stayton.

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, non-XML format and virtual drive plug-ins,
- customizations of 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`*`/.xxe/` on Linux, Mac, and more generally, on Unix.
   - *`%APPDATA%`*`\XMLmind\XMLeditor\` on Windows 2000, XP, Vista.

     Example: `C:\Documents and Settings\john\Application Data\XMLmind\XMLeditor\` on Windows 2000 and XP. `C:\Users\john\AppData\Roaming\XMLmind\XMLeditor\` on Windows Vista.

     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-std-210\doc\configure\samples\example1;\
file:///C:/xxe-std-210/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.

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 [51], elementTemplate [46], binding [32]). 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`.
  - 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).
  - RELAX NG schema locations in <?xxe-relaxng-schema location="..."?> [44].

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.xmleditapp.process.FOProcessor.

  The exact structure of a plug-in jar (manifest, service providers, etc) is described in Chapter 11, 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.

Non-XML format plug-ins
: Non-XML format plug-ins are contained in Java™ jars:

  - with a file name ending with "`_format.jar`",
  - implementing service com.xmlmind.xmleditapp.structformat.StructuredFormat.

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`).

   This feature is available only in XMLmind XML Editor Professional Edition.

2. Centralizing add-ons on a HTTP server

1. Install XXE on the server. Example: `/opt/xxe/` on a server called `rapido`.

2. Customize the distribution, if needed to. Example:

   - Create directory `/opt/xxe/addon/custom/`. This directory will contain all the custom add-ons you want to deploy.
   - Copy (XSL-FO processor plug-in) `xfc_foprocessor.jar` and `xfc.jar` to `/opt/xxe/addon/custom/`.
   - Recursively copy directory `my_configs/` containing `my_dtd1.xxe` and `my_dtd2.xxe` and all associated resources (DTD, CSS, etc) to `/opt/xxe/addon/custom/`.
   - Directory `my_configs/` also contains `my_catalog.xml`, the following XML catalog file:

     ```
     <?xml version="1.0" ?>
     <catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
              prefer="public">
       <public publicId="-//My Company//DTD DTD1 V1.0//EN"
               uri="dtd1.dtd"/>

       <public publicId="-//My Company//DTD DTD2 V1.0//EN"
               uri="dtd2.dtd"/>
     </catalog>
     ```

     This file has been copied to `/opt/xxe/addon/custom/my_configs/` along with all the other files.

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

4. In `/opt/xxe/addon/`, run command "`find custom > custom.list`" to create text file `custom.list`:

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

   /opt/xxe/addon$ cat custom.list

   custom
   custom/xfc.jar
   custom/xfc_foprocessor.jar
   custom/my_configs
   custom/my_configs/dtd1.dtd
   custom/my_configs/dtd2.dtd
   custom/my_configs/my_catalog.xml
   custom/my_configs/my_css1.css
   custom/my_configs/my_css2.css
   custom/my_configs/my_dtd1.xxe
   custom/my_configs/my_dtd2.xxe
   custom/my_configs/my_template1.xml
   custom/my_configs/my_template2.xml
   ```

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-std-`*`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 feature is available only in XMLmind XML Editor Professional Edition.

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

Important

XXE requires `<security><all-permissions/></security>` in order to run.

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: `http:`*`//machine name on which deploywebstart was run`*`/xxe`

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

  Default: `teststorepass`

-keystore *`url`*
: Keystore location.

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

-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`.

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.

-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/`.

`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` `deploywebstart` options.

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

1. Install XXE on the server. Example: `/opt/xxe/` on a server called `rapido`.
2. Install a Java™ 1.4.1+ JDK on `rapido` (a JRE is not sufficient).

   ### 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`.
3. Customize the XXE distribution, if needed to. Example:
   - Create directory `/opt/xxe/addon/custom/`. This directory will contain all the extra add-ons you want to deploy.
   - Copy (XSL-FO processor plug-in) `xfc_foprocessor.jar` and `xfc.jar` to `/opt/xxe/addon/custom/`.
   - Recursively copy directory `my_configs/` containing `my_dtd1.xxe` and `my_dtd2.xxe` and all associated resources (DTD, CSS, etc) to `/opt/xxe/addon/custom/`.
   - Directory `my_configs/` also contains `my_catalog.xml`, the following XML catalog file:

     ```
     <?xml version="1.0" ?>
     <catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
              prefer="public">
       <public publicId="-//My Company//DTD DTD1 V1.0//EN"
               uri="dtd1.dtd"/>

       <public publicId="-//My Company//DTD DTD2 V1.0//EN"
               uri="dtd2.dtd"/>
     </catalog>
     ```

     This file has been copied to `/opt/xxe/addon/custom/my_configs/` along with all the other files.
   - Add supplemental dictionaries to `/opt/xxe/addon/custom/`.

   This customization of XXE distribution can be done by hand as shown in the above example, or 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.
5. Run the `deploywebstart` command-line tool:

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

   - `-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 power user who has 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` `deploywebstart` options to specify who is signing the jars.

6. 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/>
     AllowOverride None
     Order Deny,Allow
     Deny from All
     Allow from my_company.com

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

   c. Restart apache:

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

7. Tell all your future XXE users to download and install Java™ Runtime 1.4.1+ 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.

8. 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.
Spell-checker dictionaries are downloaded and cached on the PC of the XXE user.	Spell-checker dictionaries are installed by the XXE installer on the PC of the XXE user.
Plug-ins are downloaded and cached on the PC of the XXE user.	Plug-ins stay on the server and therefore are not cached on the PC of the XXE user.

Deploying XXE using Java™ Web Start	Centralizing add-ons on a HTTP server
By default, configuration files and associated resources (DTD, CSS, icons, etc) are downloaded and cached on the PC of the XXE user. (Use the `-online` option if you prefer to keep configuration files on the deployment server.)	Configuration files and associated resources (DTD, CSS, icons, etc) stay on the server and therefore are not cached on the PC of the XXE user.
User can work offline.	User *cannot work offline*.
Personal add-ons installed on the user preferences directory of user (that is, *`%APPDATA%`*`\XMLmind\XMLeditor\addon\` on Windows and *`$HOME`*`/.xxe/addon/` on Unix) *are ignored*.	Personal add-ons installed on the user preferences directory of user are loaded.
Upgrading, for example, from XXE v3.9 to XXE v3.10 is automated for the user.	For example, user will have to manually uninstall XXE v3.9 and then to manually download and install XXE v3.10.

Part II. Reference

Chapter 7. Configuration elements

Configuration elements are directives which are executed by XXE

- during its start-up (help [48], include [51], translation [68], template [65]);
- or when loading a document (detect [39] 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 [32], css [38], etc).

1. binding

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

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

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

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

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

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

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

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

Note that `mod` is the Command key on Mac and the Control key on other platforms.

```
<charTyped
  char = single character
/>

<appEvent
  name = name of application event
/>
```

```
<command
  name = NMTOKEN
  parameter = string
/>

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

<separator
/>

<item
  label = non empty token
  icon = anyURI
  command = NMTOKEN
  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 | BRACELEFT | BRACERIGHT | C |
            CANCEL | CAPS_LOCK | CIRCUMFLEX | CLEAR |
            CLOSE_BRACKET | CODE_INPUT | COLON | COMMA |
            COMPOSE | 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 | 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 [35] to a command.

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

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 [10].

Examples:

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

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

<binding>
  <mousePressed button="2" />
  <menu label="XHTML">
    <menu label="Table">
      <item label="Insert column before" command="xhtml.tableColumn"
            parameter="insertBefore"/>
      <item label="Insert column after" command="xhtml.tableColumn"
            parameter="insertAfter"/>
      <item label="Delete column" command="xhtml.tableColumn"
            parameter="delete"/>
    </menu>
    <separator />
    <item label="Go to opposite link end"
          command="xhtml.crossReference" parameter="swap" />
    <separator />
    <item label="Preview" icon="icons/preview.gif"
          command="xhtml.preview" />
  </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 the section called “drag-source” in *XMLmind XML Editor - Support of Cascading Style Sheets (W3C CSS)*) in the document view.

  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 site.

  DITA example:

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

<command name="dita.drag">
  <macro xmlns:hrefu="java:com.xmlmind.xmleditapp.dita.HrefUtil">
    <sequence>
      <command name="ensureSelectionAt" parameter="selectElement" />
      <test context="$selectedElement"
            expression="hrefu:get-href(.) != ''" />
      <get expression="hrefu:get-href($selectedElement)" />
    </sequence>
  </macro>
</command>
```

drop
: Generated when the user drops a *string* (typically a filename or an URL) in the document view.

  If the object dropped from an external application is not a string (that is, some text), this object will be automatically converted to a string (when possible). 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:

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

Note how the string is passed to the `XXE.open` command.

Mouse click in the left or in the right margin

Generated when the user clicks in the gray margins found at the left and at the right of the document view. Note that these margins are by default absent (Preferences dialog box, Window tab, "Add interactive margins to styled views" toggle).

The name of this application event is composed as follows:

```
event_name -> margin press_or_click

margin -> '-left-margin' | '-right-margin'

press_or_click -> '-popup-trigger' | click

click -> modifiers? click_count? button

modifiers -> '-altgr'? '-alt'? '-meta'? '-ctrl'? '-shift'?

click_count -> '-double' | '-triple' | '-quadruple'
               | '-' NUMBER >= 5

button-> '-click1' | '-click2' | '-click3'
```

Examples: `right-margin-click2`, `left-margin-double-click1`, `right-margin-popup-trigger`, `right-margin-ctrl-shift-click1`.

By default, XXE uses the following bindings (plus same bindings for the right margin):

```
  <binding>
    <appEvent name="left-margin-click1" />
    <command name="selectBlockAtY" parameter="orParent" />
  </binding>

  <binding>
    <appEvent name="left-margin-popup-trigger" />
    <command name="editMenu" />
  </binding>
```

2. command

```
<command
  name = NMTOKEN
>
  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 [32] `command` or `menu`, menu [53] `item`, toolBar [66] `item` and command [36] `macro` using name *`name`*.

All commands are registered in the *same global registry* using name *`name`*. Therefore, it is strongly recommended to use a prefix (not related to XML namespace prefixes) for the name of your commands. Example of commands

written by XMLmind: `docb.moveListItemUp`, `docb.moveListItemDown`, `wxs.crossReference`, `wxs.tableColumn`, `wxs.tableRow`. (We always use *`short_lower_case_prefix.camelCaseCommandName`*.)

Example:

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

In the above example, custom command `com.xmlmind.xmleditapp.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 6, 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. configuration

```
<configuration
  name = non empty token
>
  Content: [ include|help|translation|saveOptions|command|parameterGroup|
             binding|toolBar|menu|template|css|dtd|schema|relaxng|detect|
             elementTemplate|newElementContent|property|documentHook|
             documentResources|imageToolkit|spreadsheetFunctions|
             preserveSpace|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 [3].

Example:

```
<configuration name="Example1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.xmlmind.com/xmleditor/schema/configuration"
  xmlns:cfg="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"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.xmlmind.com/xmleditor/schema/configuration"
  xmlns:cfg="http://www.xmlmind.com/xmleditor/schema/configuration">

  <include location="file:/D:/xxe/addon/config/docbook/docbook.xxe" />

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

  <template name="Chapter (part of a master document)" />
  <template name="Section (part of a master document)" />

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

</configuration>
```

The configuration in previous example can be described as follows:

- It includes bundled configuration from `file:/D:/xxe/addon/config/docbook/docbook.xxe` to reuse its detect, elementTemplate, toolBar, etc, elements.
- 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 (part of a master document)`" and "`Section (part of a master document)`" (template [65] with no `location` attribute).
- Its binds key stroke **Shift**-**Ctrl**-L command "`insert into literal`". (`mod` is the Command key on Mac and the Control key on other platforms).

4. 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" />
```

5. 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 <?xxe-relaxng-schema location="..."?> processing instruction [44],

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" />
```

It possible to use both a schema [61] or relaxng [58] configuration element and a `dtd` configuration element but in this case, the `dtd` configuration element cannot be used to specify a content model. It may be used to specify a set of character entities.

6. 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.

  ## Important

  - The detection step is always namespace-aware, and this, even when the document to be opened conforms to a DTD.[1].

    XHTML example:

    ```
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml">
        ...
    ```

    In the above case, the namespace of the root element is "`http://www.w3.org/1999/xhtml`", even if the document starts with a `<!DOCTYPE>` and thus, conforms to a DTD.

  - Attribute default values are not considered during the detection step.

    XHTML example:

    ```
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
    <html>
        ...
    ```

    In the above case, the root element has no namespace, even if "`<!ATTLIST html xmlns %URI; #FIXED 'http://www.w3.org/1999/xhtml'>`" is declared in the DTD.

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.

[1] Remember that XXE is *not* namespace-aware when the document being edited conforms to a DTD.

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 <?xxe-relaxng-schema location="..."?> processing instruction [44]) 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.

Examples:

Example 7.1. DocBook DTD

```
<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 previous 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`".

Example 7.2. DocBook RELAX NG

```
<detect>
  <rootElementNamespace>http://docbook.org/ns/docbook</rootElementNamespace>
</detect>
```

Example 7.3. XHTML Strict DTD

```
<detect>
  <or>
    <dtdPublicId>-//W3C//DTD XHTML 1.0 Strict//EN</dtdPublicId>
    <and>
       <schemaType xsi:nil="true" />
       <or>
         <rootElementLocalName>body</rootElementLocalName>
         <rootElementLocalName>div</rootElementLocalName>
         <rootElementLocalName>html</rootElementLocalName>
       </or>
    </and>
  </or>
</detect>
```

Example 7.4. XHTML RELAX NG

```
<detect>
  <and>
    <rootElementNamespace>http://www.w3.org/1999/xhtml</rootElementNamespace>
    <not>
      <or>
        <dtdPublicId>-//W3C//DTD XHTML 1.0 Strict//EN</dtdPublicId>
        <dtdPublicId>-//W3C//DTD XHTML 1.0 Transitional//EN</dtdPublicId>
      </or>
    </not>
  </and>
</detect>
```

The xxe-relaxng-schema processing instruction

This processing instruction is a non standard, proprietary, way to associate a document to a RELAX NG schema. *Its use should be restricted to testing and other quick and dirty experiments.*

DocBook RELAX NG example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xxe-relaxng-schema name="-//OASIS//RELAX NG DocBook V4.3//EN"
  location="http://www.docbook.org/rng/4.3/docbook.rng" ?>
<article>
  <title></title>
  <section>
    <title></title>
    <para></para>
  </section>
</article>
```

Like the standard `<?xml-stylesheet?>` standard processing instruction, the xxe-relaxng-schema processing instruction (which is understood only by XMLmind XML Editor) contains a number of pseudo-attributes:

location
: Required. Specifies the URL of the RELAX NG schema.

  The `location` pseudo-attribute is XML-catalog-aware.

name
: A unique name for the RELAX NG schema (similar to the public ID of a DTD). Without such name, a RELAX NG schema cannot be cached.

  When possible, the ``target namespace'' of the RELAX NG schema is a sensible choice for this attribute.

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.

7. documentHook

```
<documentHook
  name = non empty token
>
  Content: [ class ]?
</documentHook>

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

Register `documentHook` specified by *`class`* with XXE.

A `documentHook` is some code notified by XXE each time a document is created, opened, checked for validity, saved to disk and closed.

This is a very general mechanism which has been created to perform semantic validation beyond what can be done using a DTD or XML-Schema alone but which can also be used to perform many other tasks. See Chapter 9, Writing a documentHook in *XMLmind XML Editor - Developer's Guide*.

Child elements of `documentHook`:

class
: Register `documentHook` implemented in the Java™ language by class *`class`* (implements interface `com.xmlmind.xmleditapp.dochook.DocumentHook` -- SeeChapter 9, Writing a documentHook in *XMLmind XML Editor - Developer's Guide*).

Attributes of `documentHook`:

name
: This name is useful to remove or replace a previously registered `documentHook`. Anonymous `documentHooks` cannot be removed or replaced.

  When a `documentHook` element is used to remove a registered `documentHook`, a `name` attribute must be specified and there must be no `class` child element.

Example: In this example, a Java™ class named `com.xmlmind.xmleditapp.docbook.DocumentHookImpl` is contained in `docbook.jar` (among other DocBook commands and extensions).

```
<documentHook>
  <class>com.xmlmind.xmleditapp.docbook.DocumentHookImpl</class>
</documentHook>
```

A `documentHook` is always specific to a document type.

For example, the DocBook `documentHook` is used to fix the `cols` attribute of `tgroups` and `entrytbls` (if needed to) just before a DocBook document is saved to disk.

These `documentHooks` are specified in the XXE configuration file associated to the document type. For example, the DocBook documentHook is specified in `docbook.xxe`.

Several `documentHooks` can be associated to the same document type. In such case, they are notified in the order of their registration.

8. documentResources

```
<documentResources>
  Content: [ resource|selector ]+
</documentResources>

<resource>
  path = Absolute XPath (subset [46])
  action = (auto|reference|copy) : auto
/>

<selector
  action = (auto|reference|copy) : auto
>
  <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.

action
: Suggested action for the resources matched by the above XPath. This suggested action is displayed by the Resources dialog box of XXE. See the section called “The "Resources" dialog box” in *XMLmind XML Editor - Online Help*.

  `Auto` means: suggest simplest action, `copy` or `reference`, for each resource.

In complex cases, specifying document resources using simple XPath expressions (see XPath subset [46] 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.xmledit.doc.XNodeSelector`.

XHTML example:

```
  <cfg:documentResources xmlns="">
    <cfg:resource path="//img/@src" />
    <cfg:resource path="//object/@data" />
  </cfg:documentResources>
```

DocBook example:

```
  <cfg:documentResources xmlns="">
    <cfg:resource path="//@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. elementTemplate

```
<elementTemplate
  name = NMTOKEN
  parent = XPath (subset [46])
  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 XML Schema, different element types can 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 using title *`element_name`*(*`element_template_name`*) in the dialog box displayed by the Edit|Replace, Edit|Insert Before, Edit|Insert, Edit|Insert After commands.

Value `false` or `override` prevents XXE to list the element template in the dialog box displayed by Edit commands.

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 example below.

Example 7.5. DocBook 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. help

```
<help
  location = anyURI
/>
```

Add specified JavaHelp jar (Java™ Archive) to XXE online help. This JavaHelp jar must contain file `/help/jhelpset.hs` describing its help set.

Example:

```
  <help location="docbook_help.jar" />
```

Such JavaHelp jars can be localized. If, for the previous example, the language of current locale if `fr` (variant such as `CA` in `fr_CA` is ignored by XXE), XXE will try to load `docbook_help_fr.jar` and if this fails, it will attempt to load `docbook_help.jar`.

11. 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 Names
  rootQNames = 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 or the Jimi 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).

Example:

```
<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 `rootQNames` contains a list of such `QNames` (qualified names: data type which is part of the W3C XML Schema standard).

  But remember that in XXE, documents which are conforming to a DTD are not namespace-aware. In such case (for example: DocBook+SVG, that is, "`-//OASIS//DTD DocBook SVG Module V1.0//EN`"), `QNames` are not usable. That's why the `input` element also has a `rootNames` attribute which contains all the possible XML 1.0 Names for the root element of the XML image.

  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"
           rootQNames="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\XMLeditor\addon`. `%c` is the URL of the above directory. Example: `file:///C:/Documents%20and%20Settings/john/Application%20 Data/XMLmind/XMLeditor/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"
```

12. 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/.xxe/addon/mydocbook.xxe`, the included file is then `/home/john/.xxe/addon/toolBar.incl`.

Example 2:

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

If XXE has been installed in `/opt/xxe/`, the included file is `/opt/xxe/addon/config/docbook/toolBar.incl` because the XML catalog bundled with XXE contains the following rule:

```
<rewriteURI uriStartString="xxe-config:" rewritePrefix="." />
```

13. inclusionProcessor

```
<inclusionProcessor
  name = non empty token
>
  Content: [ class ]?
</inclusionProcessor>

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

Register `inclusionProcessor` specified by *`class`* with XXE.

An `inclusionProcessor` is associated to a type of document. Each time a document for which an inclusion processor has been declared is opened, XXE invokes this processor in order to evaluate the inclusion directives it contains. To make it simple, evaluating the inclusion directives means replacing these directives by up-to-date included elements. Each time a document for which an inclusion processor has been declared is saved, XXE invokes this processor in order to convert included elements back to inclusion directives.

`Xi:include` (XInclude) elements are inclusion directives processed by the "`XInclude`" inclusion processor. DITA elements having a `conref` attribute are inclusion directives processed by the "`Conref`" inclusion processor.

An inclusion processor is always associated with a command which can be used to interactively create included elements managed by the processor. For the "`XInclude`" inclusion processor, these commands are `copyAsInclusion` and `include` (see the section called “copyAsInclusion” in *XMLmind XML Editor - Commands*). For the "`Conref`" inclusion processor, this command is `dita.setConref`.

Several `inclusionProcessors` can be associated to the same document type. In such case, they are notified in the order of their registration.

There is no need to declare the "`XInclude`" inclusion processor because this standard processor is declared by default. However, in some case, it may be useful to disable the "`XInclude`" inclusion processor. Learn how below.

Child elements of `inclusionProcessor`:

class
: Register `inclusionProcessor` implemented in the Java™ language by class *`class`* (implements interface `com.xmlmind.xmledit.edit.InclusionProcessor` -- SeeChapter 10, Specializing the generic inclusion facility of XMLmind XML Editor in *XMLmind XML Editor - Developer's Guide*).

Attributes of `inclusionProcessor`:

name
: This name is useful to remove or replace a previously registered `inclusionProcessor`. Anonymous `inclusionProcessors` cannot be removed or replaced.

  When a `inclusionProcessor` element is used to remove a registered `inclusionProcessor`, a `name` attribute must be specified and there must be no `class` child element.

DITA Example:

```
<inclusionProcessor name="Conref">
  <class>com.xmlmind.xmleditapp.dita.ConrefProcessor</class>
</inclusionProcessor>

<!-- Suppress XInclude processing -->
<inclusionProcessor name="XInclude"/>
```

13.1. The "`GenericInclusion`" inclusion processor

The "`GenericInclusion`" inclusion processor is a general purpose, extensible, inclusion processor which allows to run commands which generate one or more elements on the fly in order to include these generated elements in the document being opened.

For example, using this inclusion processor, it is possible to execute a shell script (or `.bat` file) which queries a database, which converts what has been returned by the query to `row` elements and which adds these `row` elements to the `tbody` of a DocBook `table`. Like any other included element, the `row` elements included this way are made read-only, are automatically refreshed each time the document is reopened, may be refreshed by hand by using View|Redraw (**Ctrl**-**L**), etc.

Example 7.6. Simple date inclusion example

DocBook document `sample.xml` contains:

```
...
<?xxe-begin-inclusion %C/date.sh?>
<phrase>Wed Jun 7 18:03:34 CEST 2006</phrase>
<?xxe-end-inclusion ?>
...
```

where `date.sh` is this simple shell script:

```
#!/bin/sh
now=`date`
echo "<?xml version='1.0'?><phrase><![CDATA[$now]]></phrase>"
```

Unlike the "`XInclude`" inclusion processor which is predeclared, the "`GenericInclusion`" inclusion processor needs to be declared in a configuration in order to be used by XXE. Let's suppose the user has added this snippet[2] to `docbook.xxe`, the DocBook configuration for XXE:

```
  <inclusionProcessor name="GenericInclusion">
    <class>com.xmlmind.xmleditapp.genericincl.GenericInclusionProcessor</class>
  </inclusionProcessor>
```

Each time `sample.xml` is opened, XXE invokes the "`XInclude`" inclusion processor and then, because this processor has been declared in `docbook.xxe`, the "`GenericInclusion`" inclusion processor.

The "`GenericInclusion`" inclusion processor replaces all the elements found between `<?xxe-begin-inclusion?>`/`<?xxe-end-inclusion?>` pairs by the elements generated by the command which is contained in `<?xxe-begin-inclusion?>`. For this example, the command is "`%C/date.sh`"[3].

The `<?xxe-begin-inclusion?>`/`<?xxe-end-inclusion?>` pairs themselves are removed and included elements are made read-only and are marked as being managed by the "`GenericInclusion`" inclusion processor.

Then, each time `sample.xml` is saved, XXE requests the included elements having the "`GenericInclusion`" inclusion mark to restore their inclusion directives, that is, to restore the `<?xxe-begin-inclusion?>`/`<?xxe-end-inclusion?>` pairs.

Just like commands `copyAsInclusion` and `include` create included elements which are managed by the "`XInclude`" inclusion processor, there is command, generally called `genericInclude`, which may be used to create included elements which are managed by the "`GenericInclusion`" inclusion processor. This generic inclusion command is documented in the section called “A generic inclusion command” in *XMLmind XML Editor - Commands*.

14. 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
```

[2] The easiest way to do so is to include genericincl.incl in the XXE configuration file (for this example, `docbook.xxe`).

[3] `%C` is the name of the directory containing the XXE configuration file where the "`GenericInclusion`" inclusion processor has been declared.

This makes it easy deploying the shell scripts invoked by the "`GenericInclusion`" inclusion processor: suffice to copy them in the directory containing the XXE configuration file.

```
  icon = anyURI
  command = NMTOKEN
  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="xhtml.crossReference" 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.

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>
```

14.1. 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>
```

15. 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 [46].

`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" />
```

16. property

```
<property
  name = non empty token
  url = boolean
  xml:space = preserve
>text</property>
```

Define system property 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 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>
```

17. parameterGroup

```
<parameterGroup
  name = non empty token
>
  Content: [ parameter | parameterGroup ]*
</parameterGroup>

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

Define a named group of XSLT style sheet parameters for use inside element `transform` of a process command [36].

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\XMLeditor\addon\customize.xxe` (*`$HOME`*`/.xxe/addon/customize.xxe` on Unix) 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>
```

18. preserveSpace

```
<preserveSpace
  elements = list of XPath (subset [46])
/>
```

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" />
```

19. relaxng

```
<relaxng
  location = anyURI
  name = non empty token
  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.

name
: A unique name for the RELAX NG schema (similar to the public ID of a DTD). Without such name, a RELAX NG schema cannot be cached.

  When possible, the ``target namespace'' of the RELAX NG schema is a sensible choice for this attribute.

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 <?xxe-relaxng-schema location="..."?> processing instruction [44],

the grammar specified this way is used and the RELAX NG schema specified in the configuration file is ignored.

XHTML RELAX NG example:

```
<relaxng name="http://www.w3.org/1999/xhtml"
         location="rng/xhtml-strict.rng" />
```

Compact syntax example:

```
<relaxng compactSyntax="true" encoding="ISO-8859-1"
         location="example3.rnc"
         name="http://www.xmlmind.com/xmleditor/schema/example3"/>
```

It possible to use both a `relaxng` configuration element and a dtd [39] configuration element but in this case, the `dtd` configuration element cannot be used to specify a content model. It may be used to specify a set of character entities.

20. saveOptions

```
<saveOptions
  encoding = (ISO-8859-1|ISO-8859-13|ISO-8859-15|ISO-8859-2|
              ISO-8859-3|ISO-8859-4|ISO-8859-5|ISO-8859-7|
              ISO-8859-9|KOI8-R|MacRoman|US-ASCII|UTF-16|UTF-8|
              Windows-1250|Windows-1251|Windows-1252|Windows-1253|
              Windows-1257)
  indent = none | (int >= 0)
  maxLineLength = unbounded | (int > 0)
  addOpenLines = boolean
  cdataSectionElements = list of XPath (subset [46])
  saveCharsAsEntityRefs = boolean
  charsSavedAsEntityRefs = list of character ranges
  favorInteroperability = boolean
/>
```

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 [46] 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).

Examples:

```
<saveOptions addOpenLines="false" />

<cfg:saveOptions xmlns="" cdataSectionElements="head/script"/>
```

```
<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" />
```

21. 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 <?xxe-relaxng-schema location="..."?> processing instruction [44],

the grammar specified this way is used and the W3C XML Schema specified in the configuration file is ignored.

Example:

```
  <schema>
    <location>http://www.xmlmind.com/xmleditor/schema/configuration
              xsd/configuration.xsd</location>
  </schema>
```

It possible to use both a `schema` configuration element and a dtd [39] configuration element but in this case, the `dtd` configuration element cannot be used to specify a content model. It may be used to specify a set of character entities.

22. schematron

```
<schematron
  location = anyURI
```

```
  namespaceAware = boolean : true
  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`".

Attributes:

location
: URL of the Schematron schema. Both ISO Schematron or Schematron 1.5 are supported.

  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).

namespaceAware
: Specify `true` for documents conforming to a DTD.

  Remember that with XXE, documents conforming to a DTD are not namespace aware. Example, if such document contains an element having `mml:math` as its name, this name is understood as being `{}mml:math` (no namespace, "`mml:math`" local part) and not as being `{http://www.w3.org/1998/Math/MathML}math` ("`http://www.w3.org/1998/Math/MathML`" namespace, "`math`" localpart).

  When a document is not namespace aware, then the schematron used to validate such document must interpret qualified names accordingly. Attribute `namespaceAware=true` ensures that this is the case.

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, not namespace aware) example:

```
<schematron location="docbook.sch" namespaceAware="false"
            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"/>`.)

22.1. Relationship between `schematron` and `documentHook`

This `schematron` configuration element is a documentHook [44] configuration element in disguise. A `schematron` element is equivalent to:

```
<documentHook name="Schematron">
  <class>com.xmlmind.xmleditapp.config.SchematronHook</class>
</documentHook>
```

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. -->
   <documentHook>
     <class>com.xmlmind.xmleditapp.docbook.table.DocumentHookImpl</class>
   </documentHook>

   <schematron location="docbook.sch" />
   ```

2. The snippet below may be used to *remove* previously declared `schematron`.

   ```
   <documentHook name="Schematron" />
   ```

23. spellCheckOptions

```
<spellCheckOptions
  useAutomaticSpellChecker = boolean
  languageAttribute = QName
  defaultLanguage = language
  checkComments = boolean
  checkedProcessingInstructions = list of Names
  checkedAttributes = list of XPath (subset [46])
  skippedElements = list of XPath (subset [46])
/>
```

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 [63].

  This setting may be overridden by the user with Options|Preferences, Tools|Spell section, Automatic Spell Checker radio buttons.

languageAttribute
: Specifies which attribute, if any, specifies the language of an element and all its descendants. This is typically `xml:lang` or `lang`.

  Default: there is no such attribute; see language lookup [63].

defaultLanguage
: Specifies the default language of a document of that type. (This option is rarely used.)

  Default: no default language; see language lookup [63].

Note

XMLmind XML Editor determines the language of an element by examining, in that order:

1. the value of the attribute 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 [46]) 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 [46]) 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, XHTML/RELAX NG):

```
 <cfg:spellCheckOptions xmlns=""
    useAutomaticSpellChecker="true"
    languageAttribute="lang"
    skippedElements="address funcsynopsisinfo classsynopsisinfo
                     literallayout programlisting screen synopsis" />

  <cfg:spellCheckOptions xmlns=""
    useAutomaticSpellChecker="true"
    languageAttribute="xml:lang"
    skippedElements="pre style script" />

  <cfg:spellCheckOptions xmlns:html="http://www.w3.org/1999/xhtml"
    useAutomaticSpellChecker="true"
    languageAttribute="xml: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"
    skippedElements="pre script" />
```

```
        .
        .
        .
<cfg:spellCheckOptions xmlns=""
    languageAttribute="xml:lang"
    defaultLanguage="en-US"
    checkComments="true"
    checkedProcessingInstructions="annotation remark"
    checkedAttributes="@alt table/@summary table/@title" />
```

is equivalent to:

```
<cfg:spellCheckOptions xmlns=""
    useAutomaticSpellChecker="true"
    languageAttribute="xml:lang"
    defaultLanguage="en-US"
    checkComments="true"
    checkedProcessingInstructions="annotation remark"
    checkedAttributes="@alt table/@summary table/@title"
    skippedElements="pre script">
```

24. 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.

25. template

```
<template
  name = non empty token
  location = anyURI
/>
```

Add document template named *`name`*, contained in file *`location`*, to the dialog box displayed by File|New.

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

Example:

```
  <template name="Div (part of a master document)"
            location="template/div.html" />
```

26. toolBar

```
<toolBar
  name = NMTOKEN
  insert = non empty token
>
  Content: [ separator | button | insert ]*
</toolBar>

<separator />

<insert />

<button
  icon = anyURI
  toolTip = non empty token
>
  Content: command | menu
</button>

<command
  name = NMTOKEN
  parameter = string
/>

<menu>
  Content: [ item | separator ]+
</menu>

<item
  label = non empty token
  icon = anyURI
  command = NMTOKEN
  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>
```

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>
```

26.1. 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>
   ```

27. translation

```
<translation
  location = anyURI matching [path/]resourcename_lang.properties
/>
```

Specifies how to translate messages found in menu [53] `item` `label`, toolBar [66] `button` `toolTip`, template [65] `name`, elementTemplate [46] `name`, css [38] `name`, binding [32] `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[4] 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
     ...
```

28. windowLayout

```
<windowLayout>
  Content (in any order): center [ top ]? [ bottom ]?
                          [ left ]? [ right ]?
</windowLayout>
```

[4] *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`.

```
<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="Images displayed as thumbnails" alternate="true"
       location="css/thumbnails.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" />
```