XMLmind XML Editor - Customizing the User Interface

Hussein Shafie
Pixware

<xmleditor-support@xmlmind.com>

XMLmind XML Editor - Customizing the User Interface

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

Published December 2, 2005

Abstract

This document describes how to customize the user interface of XMLmind XML Editor by writing a GUI specification (`.xxe_gui` XML files) and by deploying it.

Table of Contents

Chapter 1. Introduction

The user interface (GUI) of XMLmind XML Editor is made of *parts*. Parts are high level building blocks such as menus, menu bars, tool bars, status bars, actions (for use in menus, tool bars and status bars), etc.

These parts are declared in a special GUI specification file having a `.xxe_gui` suffix. Such GUI specification files also contain a `layout` element which specifies which *assembly of parts* to use to create the user interface of XMLmind XML Editor.

Example (excerpts of default.xxe_gui):

```
<?xml version='1.0' encoding='UTF-8'?>
<gui xmlns="http://www.xmlmind.com/xmleditor/schema/gui"
     xmlns:gui="http://www.xmlmind.com/xmleditor/schema/gui"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.xmlmind.com/xmleditor/schema/gui
                         ../../../addon/config/gui/xsd/gui.xsd">
  ...
  <action name="newAction" label="_New..."1
          icon="icons/newAction.gif"
          accelerator="mod N">
    <class>com.xmlmind.xmleditapp.kit.part.NewAction</class>
  </action>
  ...
  <menu name="fileMenu" label="_File"2
        helpId="fileMenu">
    <action name="newAction" />
    <action name="openAsTemplateAction" />
    ...
    <menuItems name="recentFilesMenuItems" />
  </menu>
  ...
  <menuBar name="menuBar" helpId="menuBar">3
    <menu name="fileMenu" />
    <menu name="selectMenu" />
    ...
  </menuBar>
  ...
  <layout width="850" height="650">4
    <menuBar name="menuBar" />5
    ...
  </layout>
</gui>
```

1 Declares action "`newAction`".
2 Declares menu "`fileMenu`".
3 Declares menu bar "`menuBar`".
4 Element `layout` actually specifies which GUI to create.
5 It is menu bar "`menuBar`" which will be used in the GUI of XXE because it is referenced in the `layout` element.

Important

Declaring a part does not mean that this part will be created and then, displayed in the GUI of XXE. For this to happen, a part must be referenced directly or indirectly[1] by the `layout` element of the GUI specification file.

`Xxe_app.jar`, the archive file containing the code of XMLmind XML Editor (XXE for short), also contains 2 GUI specifications:

Resource `/gui/app/default.xxe_gui`
: Specifies the default user interface of XMLmind XML Editor.

[1] Example: layout references a statusBar which references an action.

Resource `/gui/app/simple.xxe_gui`
: Specifies a somewhat simpler user interface which could be preferred by some users.

A copy of these files is found in *`XXE_install_dir`*`/doc/gui/gui/app/`. This, because using default.xxe_gui as a reference when creating custom GUIs for XXE is *absolutely required*.

If you want to see by yourself the effect of using simple.xxe_gui (instead of default default.xxe_gui), please proceed as follows:

Windows

1. Open a command prompt.
2. Change working directory to the directory where XXE has been installed (typically `C:\Program Files\XMLmind_XML_Editor\`).
3. Change to subdirectory `bin\` where `xxe.exe` (and `xxe.bat`) are found.
4. Set environment variable `XXE_GUI` as follows:

   ```
   C:\Program Files\XMLmind_XML_Editor\bin> set XXE_GUI=xxe-gui:app/simple.xxe_gui
   ```

5. Start XXE as follows:

   ```
   C:\Program Files\XMLmind_XML_Editor\bin> xxe.exe
   ```

Linux, Mac (with `bash`)

1. Open a terminal.
2. Change working directory to the directory where XXE has been installed (example: `/opt/xxe/`).
3. Change to subdirectory `bin\` where the `xxe` shell script is found.
4. Set environment variable `XXE_GUI` as follows[2]:

   ```
   /opt/xxe/bin$ XXE_GUI=xxe-gui:app/simple.xxe_gui; export XXE_GUI
   ```

5. Start XXE as follows:

   ```
   /opt/xxe/bin$ ./xxe &
   ```

Don't worry, all this will be detailed in the deployment chapter of this document.

[2] Quicker, directly execute this:

```
/opt/xxe/bin$ XXE_GUI=xxe-gui:app/simple.xxe_gui ./xxe &
```

Chapter 2. Tutorial

What follows is a description of the customization we want to do:

1. Change the (desktop) title and icon of XMLmind XML Editor.

2. Change the dialog box displayed by Help|About.

3. Add to the status bar a small tool which allows to count the words of a document.

4. Extend the dialog box displayed by Options|Options by adding a custom ``preferences sheet''. This preferences sheet is needed to parametrize the above word count tool.

5. Move iconic buttons which are specific to a given document type (i.e. DocBook buttons, XHTML buttons, etc) from their normal place, the standard tool bar, to a tool bar of their own. This new tool bar will be vertical and will be found at the left of the document views.

6. Replace by a *contextual action* the action taken when File|Print is selected. If the document being edited is a DocBook, Simplified DocBook, Slides or XHTML one, the contextual action will convert the document to PDF using a method which is specific to its document type. Otherwise, the document will be printed normally.

7. Add a custom Characters menu after standard Edit menu. This Characters menu will allow to quickly insert special characters (Greek letters, arrows) in the document being edited.

To do this, we need not only to change the specification of the GUI of XXE, but also to develop custom parts in the Java™ language. These custom parts are:

- an action displaying the custom About dialog box;

- a word count tool;

- a preferences sheet for the above word count tool.

All this is explained in *XMLmind XML Editor - Developer's Guide* and the custom parts are found in *`XXE_install_dir`*`/doc/gui/tutorial/custom_parts.jar` with the other files of this tutorial. This being said, we can now focus on the specification of the user interface of XMLmind XML Editor.

1. Changing the title and icon

tutorial/tutorial1.xxe_gui:

```
<?xml version='1.0' encoding='UTF-8'?>
<gui xmlns="http://www.xmlmind.com/xmleditor/schema/gui"1
     xmlns:gui="http://www.xmlmind.com/xmleditor/schema/gui">

  <include location="xxe-gui:app/default.xxe_gui" />2

  <translation location="custom_gui_en.properties" />3

  <layout label="Document Editor" icon="docedit.png">4
    <insert />5
  </layout>
</gui>
```

1 The namespace of GUI specification elements is "`http://www.xmlmind.com/xmleditor/schema/gui`". The corresponding schema is found in *`XXE_install_dir`*`/addon/config/gui/xsd/gui.xsd`.

2 All customizations should start by including "`xxe-gui:app/default.xxe_gui`" (or "`xxe-gui:app/simple.xxe_gui`" if you intend to customize simple.xxe_gui) because default.xxe_gui contains the declarations of all stock parts.

Note

URLs starting with "xxe-gui:" work because XXE dynamically adds to its XML catalog a rule similar to this one:

```
<rewriteURI uriStartString="xxe-gui:"
            rewritePrefix="jar:file:/opt/xxe/bin/xxe-app.jar!/gui/" />
```

3 GUI specification files can be localized by using translation elements. For more information please consult the reference part of this document.
4 The layout element has label and icon attributes specifying the title and desktop icon of the XML editor.
5 Without the use of insert, the above layout would have been understood as a redefinition from scratch of the standard layout of the XML editor. Such XML editor would have had a blank main window.

2. Changing the About dialog box

tutorial/tutorial2.xxe_gui:

```
<action name="aboutAction" label="_About This Document Editor">
  <class>AboutAction</class>
</action>
```

It is a custom action implemented by Java™ class AboutAction (code contained in *XXE_install_dir*/doc/gui/tutorial/custom_parts.jar) which displays our custom About dialog box.

Naming our action "aboutAction", just like the standard action "aboutAction" found in default.xxe_gui, suffice to do the job.

3. Adding a word count tool

tutorial/tutorial3.xxe_gui:

```
<tool name="countWordsTool">
  <class>CountWordsTool</class>
</tool>

<statusBar name="statusBar">
  <tool name="countWordsTool" />
  <insert />
</statusBar>
```

The custom tool implemented by Java™ class CountWordsTool (code contained in *XXE_install_dir*/doc/gui/tutorial/custom_parts.jar) is declared using a tool element.

Then this tool is inserted in the standard status bar (statusBar element called "statusBar" found in default.xxe_gui), before all the other components of this status bar, by the means of the <insert/> facility.

4. Adding a preferences sheet for the word count tool

tutorial/tutorial4.xxe_gui:

```
<preferencesSheet name="countWordsOptions">
  <class>CountWordsOptions</class>
</preferencesSheet>

<preferencesSheets name="preferencesSheets">
  <insert />
  <preferencesSheet name="countWordsOptions" />
</preferencesSheets>
```

The custom preferences sheet implemented by Java™ class CountWordsOptions (code contained in *XXE_install_dir*/doc/gui/tutorial/custom_parts.jar) is declared using a preferencesSheet element.

Then this sheet is inserted in the standard set of `preferencesSheets` (preferencesSheets element called "`preferencesSheets`" found in default.xxe_gui), after all the other sheets, by the means of the `<insert/>` facility.

5. Adding a vertical tool bar

tutorial/tutorial5.xxe_gui:

```
 <toolBar name="toolBar" helpId="toolBar">1
   <action name="newAction" />
   <action name="openAction" />
   <action name="saveAction" />
   <action name="saveAllAction" />
   <action name="toggleUseURLChooserAction" />
   <separator />
   <action name="undoAction" />
   <action name="redoAction" />
   <action name="repeatAction" />
   <action name="commandHistoryAction" />
   <separator />
   <action name="cutAction" />
   <action name="copyAction" />
   <action name="pasteBeforeAction" />
   <action name="pasteAction" />
   <action name="pasteAfterAction" />
   <action name="deleteAction" />
   <separator />
   <action name="splitAction" />
   <action name="joinAction" />
 </toolBar>

 <toolBar name="configSpecificToolBar">2
   <toolBarItems name="configSpecificToolBarItems" />3
 </toolBar>

 <layout label="Document Editor" icon="docedit.png">
   <insert />

   <leftToolBars>4
     <toolBar name="configSpecificToolBar" />
   </leftToolBars>
</layout>
```

1 There is no facility similar to the `<insert/>` which would allow us to *remove* the set of configuration specific buttons from the standard tool bar. Therefore we have to redefine the standard tool bar (element `toolBar` called "`toolBar`" found in default.xxe_gui) from scratch.

2 Declare a new tool bar called "`configSpecificToolBar`" by using a toolBar element.

3 This new `toolBar` only contains the set of configuration specific buttons. This set is declared in default.xxe_gui as the toolBarItems element called "`configSpecificToolBarItems`". Therefore, we just need to reference this stock part in our new tool bar.

4 Add a leftToolBars child to the layout element. This child element can contain one or more vertical `toolBars`.

6. Making File|Print dependant on the type of the document to be printed

tutorial/tutorial6.xxe_gui:

```
 <action name="printAction" label="_Print..."
         icon="xxe-gui:app/icons/printAction.gif"
         accelerator="mod P">
   <context editingContextSensitive="false">
     <configuration name="XHTML">
       <command name="xhtml.convertToPS" parameter='pdf |pdf' />
     </configuration>
```

```
    <configuration name="DocBook">
      <command name="docb.convertToPS"
               parameter='pdf |pdf "/book toc /article toc" 1' />
    </configuration>

    <configuration name="Simplified DocBook">
      <command name="docb.convertToPS"
               parameter='pdf |pdf "/book toc /article toc" 1' />
    </configuration>

    <configuration name="Slides">
      <command name="slides.convertToPS" parameter='pdf |pdf' />
    </configuration>

    <default>
      <class>com.xmlmind.xmleditapp.kit.part.PrintAction</class>
    </default>
  </context>
</action>
```

The standard print action (i.e. the action referenced by the standard File menu) is defined as follows in default.xxe_gui:

```
<action name="printAction" label="_Print..."
        icon="icons/printAction.gif"
        accelerator="mod P">
  <class>com.xmlmind.xmleditapp.kit.part.PrintAction</class>
</action>
```

Therefore if we define a contextual action having the same name, "`printAction`", this suffices to install our custom action. Note how the `icon` attribute is specified in our redefinition of the print action.

The above contextual action can be described as follows:

- If the document to be printed is associated with a configuration named "`XHTML`", invoke command `xhtml.convertToPS` with parameter "`pdf |pdf`" to print it.
- Same with documents associated to configurations named "`DocBook`", "`Simplified DocBook`" or "`Slides`".
- If the document to be printed has no associated configuration or is associated with a configuration other than the above ones, print it using action implemented by Java™ class `com.xmlmind.xmleditapp.kit.part.PrintAction` (same as stock print action).

Elements `configuration` and `default` can contain actions implemented in Java™ or can contain commands (see *XMLmind XML Editor - Commands*). Referencing an action is easy to understand, but what is this command `xhtml.convertToPS` used to print XHTML documents?

The configuration file for XHTML documents is *`XXE_install_dir`*`/addon/config/xhtml/xhtml.xxe`. The `name` attribute of the root element of this configuration file is `XHTML` (see *XMLmind XML Editor - Configuration and Deployment*).

This configuration file includes another configuration file *`XXE_install_dir`*`/addon/config/xhtml/xslMenu.incl`. A process command called `xhtml.convertToPS` is defined in `xslMenu.incl`. This process command, with parameter "`pdf |pdf`", converts the document being edited to PDF[1].

7. Adding a Characters menu

tutorial/tutorial7.xxe_gui:

```
<action name="insertLeftAction" label="_L - &#x2190;">1
  <command name="insertString" parameter="&#x2190;" />
```

[1] In the future, a process command will be able to directly send the result of the conversion to a printer — typically a PostScript™ printer — chosen by the user.

```
 </action>
 <action name="insertRightAction" label="_R - &#x2192;">
   <command name="insertString" parameter="&#x2192;" />
 </action>

 <action name="insertAlphaAction" label="_A - &#x03B1;">
   <command name="insertString" parameter="&#x03B1;" />
 </action>
 <action name="insertBetaAction" label="_B - &#x03B2;">
   <command name="insertString" parameter="&#x03B2;" />
 </action>
 <action name="insertGammaAction" label="_C - &#x03B3;">
   <command name="insertString" parameter="&#x03B3;" />
 </action>

 <menu name="arrowsMenu" label="_Arrows">
   <action name="insertLeftAction" />
   <action name="insertRightAction" />
 </menu>

 <menu name="greekMenu" label="_Greek">
   <action name="insertAlphaAction" />
   <action name="insertBetaAction" />
   <action name="insertGammaAction" />
 </menu>

 <menu name="charactersMenu" label="_Characters">2
   <menu name="arrowsMenu" />
   <menu name="greekMenu" />
 </menu>

 <menuBar name="customMenuBar">
   <menu name="fileMenu" />
   <menu name="selectMenu" />
   <menu name="editMenu" />
   <menu name="charactersMenu" />3
   <menu name="searchMenu" />
   <menu name="viewMenu" />
   <menu name="toolsMenu" />
   <menu name="configSpecificMenu" />
   <menu name="windowMenu" />
   <menu name="optionsMenu" />
   <menu name="helpMenu" />
 </menuBar>

 <layout label="Document Editor" icon="docedit.png">
   <insert />

   <menuBar name="customMenuBar" />4

   <leftToolBars>
     <toolBar name="configSpecificToolBar" />
   </leftToolBars>
</layout>
```

1 Define actions which insert the chosen special characters by wrapping an action around standard command "`insertString`" (see *XMLmind XML Editor - Commands*).

2 Define the Characters menu and its two submenus: the Arrows menu and the Greek menu.

3 There is no way to insert `charactersMenu` after `editMenu` using the `<insert/>` facility. Therefore we define a new menuBar called "`customMenuBar`" from scratch.

4 Without this reference, the menu bar used by XXE would have been the standard menu bar (called "`menuBar`" — see default.xxe_gui).

Chapter 3. Deployment

GUI specification files may be written using XMLmind XML Editor (a simple configuration for `.xxe_gui` files is included in the distribution of XXE) or using a text editor.

If you use a text editor, do not forget to validate your GUI specification against its schema. This schema is found in *`XXE_install_dir`*`/addon/config/gui/xsd/gui.xsd`. Unix example:

```
~/.xxe/addon$ /opt/xxe/bin/xsdvalid -s /opt/xxe/addon/config/gui/xsd/gui.xsd custom.xxe_gui
```

There are two ways to deploy a custom GUI specification for XXE:

1. Set environment variable `XXE_GUI` as shown in the introduction of this document.

2. OR rename your `.xxe_gui` file to `custom.xxe_gui` (this special name is mandatory).

   Then copy this file and all its resources (icons, translations, etc) to one of the `addon/` directories scanned by XXE during its startup.

   You need to use this second method if you deploy XXE with Java™ Web Start (described in *XMLmind XML Editor - Configuration and Deployment*). Note that this is consistent with the way all the other configuration files are deployed.

First method applied to the GUI created during this tutorial:

Windows (assuming that XXE has been installed in `C:\Program Files\XMLmind_XML_Editor\`)

1. Open a command prompt.

2. Change working directory to `C:\Program Files\XMLmind_XML_Editor\doc\gui\tutorial\`.

3. Set environment variable `XXE_GUI` as follows:

   ```
   C:\...\tutorial> set XXE_GUI=tutorial1.xxe_gui
   ```

4. Start XXE as follows:

   ```
   C:\...\tutorial> C:\Program Files\XMLmind_XML_Editor\bin\xxe.exe
   ```

Linux, Mac (with `bash`; assuming that XXE has been installed in /op/xxe/)

1. Open a terminal.

2. Change working directory to `/opt/xxe/doc/gui/tutorial`.

3. Start XXE as follows:

   ```
   /opt/xxe/doc/gui/tutorial$ XXE_GUI=tutorial1.xxe_gui /opt/xxe/bin/xxe &
   ```

Important

This method works fine with `tutorial1.xxe_gui`, but not with `tutorial2.xxe_gui`, `tutorial3.xxe_gui`, etc. Why? Because the files other than `tutorial1.xxe_gui` require XXE to load *code* found in *`XXE_install_dir`*`/doc/gui/tutorial/custom_parts.jar`.

In practice this means that if you want to use this method with a custom GUI making use of custom parts, you'll nevertheless have to deploy the custom code by copying its jar to one of the `addon/` directories scanned by XXE during its startup.

Second method applied to the GUI created during this tutorial:

1. Copy directory *XXE_install_dir*/doc/gui/tutorial/ and all its content (which includes custom.xxe_gui, a file having a content identical to tutorial7.xxe_gui) to directory *XXE_user_preferences_dir*/addon/.

   XXE user preferences directory is:

   - *$HOME*/.xxe/ on Unix,
   - *%SystemDrive%*\Documents and Settings*%USERNAME%*\Application Data\XMLmind\XMLeditor\ on Windows 2000/XP,
   - *%SystemDrive%*\winnt\Profiles*%USERNAME%*\Application Data\XMLmind\XMLeditor\ on Windows NT.

2. Restart XXE as you usually do it.

Chapter 4. Reference

GUI specifications file must have a `.xxe_gui` suffix. The customization file automatically detected by XXE during its startup is called `custom.xxe_gui`.

All the elements described in this chapter belong to the "`http://www.xmlmind.com/xmleditor/schema/gui`" namespace. The local name of the root element of a GUI specification must be `gui`:

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

A `gui` root element may contain any number of the following elements, and that, in any order.

1. action

```
<action
  name = NMTOKEN
  label = non empty token
  icon = anyURI
  selectedIcon = anyURI
  toolTip = non empty token
  accelerator = non empty token
  acceleratorOnMac = non empty token
>
  Content: class | command | context
</action>

<class
  implementsCommand = NMTOKEN
>
  Content: Java class name
</class>

<command
  name = NMTOKEN
  parameter = string
  editingContextSensitive = boolean : true
/>

<context
  editingContextSensitive = boolean : false
>
  Content: [ configuration ]+ default
</context>

<configuration
  name = non empty token
>
  Content: class | command
</configuration>

<default>
  Content: class | command
</default>
```

Specifies an action, that is, an instance of class `com.xmlmind.xmleditapp.kit.AppAction`.

The class derived from `com.xmlmind.xmleditapp.kit.AppAction` is specified by the `class` child element. In a few cases, this class also implements interface `com.xmlmind.xmledit.gadget.Command`. In such case, attribute `implementsCommand` specifies the name of the command.

The `action` element also allows to specify different kinds of AppActions without having to write Java™ code for that:

Child element `command`
: Specifies an action which delegates its job to a command (see *XMLmind XML Editor - Commands*). See example in the tutorial part of this document. Explanations below.

Child element `context`
: Specifies an action which is contextual, that is, the action actually taken (delegated to a command or to an actual, non-contextual, action) depends of the configuration associated to the active document. See example in the tutorial part of this document. Explanations below.

Attributes:

name
: Required. Unique name identifying the action in this GUI specification.

label
: One of `label` and `icon` is required. Label of the action (an action is used to create buttons, menu items and tool bar items).

icon
: One of `label` and `icon` is required. Icon of the action (an action is used to create buttons, menu items and tool bar items).

  This URI may be resolved using XML catalogs.

selectedIcon
: Only used for classes derived from `com.xmlmind.xmleditapp.kit.AppToggleAction` or from `com.xmlmind.xmleditapp.kit.EditToggleAction`. These classes are used to implement ``toggles''. The icon specified by attribute `selectedIcon` is used when the toggle is turned *on*. The icon specified by attribute `icon` is used when the toggle is turned *off*.

  This URI may be resolved using XML catalogs.

toolTip
: The tool tip of the action (an action is used to create buttons, menu items and tool bar items).

  If this attribute is not specified, and if a tool tip is absolutely needed by the representation of the action (button, menu item and tool bar item), the value of attribute `label` will be used.

accelerator
: The hot key used to trigger the action.

  Hot keys are specified using the following syntax:

```
[ ctrl|shift|alt|meta|mod ]* key_code

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)
```

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

acceleratorOnMac
: The hot key used to trigger the action on the Mac. This is needed because on the Mac, so many hot keys are reserved for the desktop.

  If this attribute is not specified, and if attribute `accelerator` is specified, the hot key specified by `accelerator` will also be used on Macs.

Example: a simple action:

```
<action name="addViewAction" label="_Add...">
  <class>com.xmlmind.xmleditapp.app.part.AddViewAction</class>
</action>
```

Example: an action which implements a command:

```
<action name="openAction" label="_Open..."
        icon="icons/openAction.gif"
        accelerator="mod O">
  <class
    implementsCommand="XXE.open"
  >com.xmlmind.xmleditapp.kit.part.OpenAction</class>
</action>
```

Example: a toggle action:

```
<action name="toggleUseURLChooserAction"
        toolTip="Use URL chooser rather than file chooser"
        icon="icons/toggleUseURLChooserAction.gif"
        selectedIcon="icons/toggleUseURLChooserAction_selected.gif">
  <class>com.xmlmind.xmleditapp.kit.part.ToggleUseURLChooserAction</class>
</action>
```

1.1. Action which is a wrapper around a command

Action which delegates its job to a command (see *XMLmind XML Editor - Commands*). This command delegate is specified using the `command` child element.

Attributes of the `command` child element:

name
: Specifies the name under which the command has been registered.

This cannot be a configuration specific command. For example, this cannot be `xhtml.convertToPS`, which converts XHTML documents to PostScript™ and PDF.

If the command delegate is not a built-in command, but instead, is a custom generic command specially written to be wrapped in an action, this command:

- must be declared in this GUI specification using a command element;
- must be referenced in the hidden child element of the layout element.

parameter
: Specifies the parameter of the command delegate, if any.

editingContextSensitive
: Specifies whether the command delegate is editing context sensitive, that is, is enabled or disabled depending on the editing context (Which node is selected? Is some text selected? Etc).

  The majority of commands are editing context sensitive. That's why the default value of this attribute is `true`.

Example:

```
<action name="insertAfterAction" label="Insert _After..."
        icon="icons/insertAfterAction.gif">
  <command name="insert" parameter="after[implicitElement]" />
</action>
```

1.2. Contextual action

Action which is contextual, that is, the action actually taken (delegated to a command or to an actual, non-contextual, action) depends of the configuration associated to the active document. This kind of action is specified using the `context` child element.

In a `context` child element:

Element `configuration`
: Specifies what to do for a document associated to a configuration having a name equal to the value of attribute `name`.

  Example: this `configuration` element

  ```
  <configuration name="Simplified DocBook">
    <command name="docb.convertToPS"
             parameter='pdf |pdf "/book toc /article toc" 1' />
  </configuration>
  ```

  specifies what to do for a document associated to a configuration called "`Simplified DocBook`" (see *XMLmind XML Editor - Configuration and Deployment*).

Element `default`
: Specifies what to do for a document

  - which has no associated configuration;
  - or which has a configuration, but this configuration is not referenced in one of its `configuration` siblings.

  Example: this `default` element:

  ```
  <default>
    <class>com.xmlmind.xmleditapp.kit.part.PrintAction</class>
  </default>
  ```

  specifies what to do for a document not having a configuration or having a configuration different from "`XHTML`", "`DocBook`", "`Simplified DocBook`" and "`Slides`".

The `configuration` element and the `default` element have the same content model. Their child element, `class` or `command`, specifies the action actually taken:

Element `class`
: Specifies a Java™ class derived from `com.xmlmind.xmleditapp.kit.AppAction`.

Element `command`
: Specifies a command which can be a generic one (e.g. command `convert`) or a configuration specific one (e.g. command `xhtml.preview`).

The `editingContextSensitive` attribute of the context child element specifies whether the contextual action is editing context sensitive, that is, is enabled or disabled depending on the editing context (Which node is selected? Is some text selected? Etc). The majority of contextual actions are expected to be editing context *insensitive*. That's why the default value of this attribute is `false`.

Example:

```
<action name="printAction" label="_Print..."
          icon="xxe-gui:app/icons/printAction.gif"
          accelerator="mod P">
    <context editingContextSensitive="false">
      <configuration name="XHTML">
        <command name="xhtml.convertToPS" parameter='pdf |pdf' />
      </configuration>

      <configuration name="DocBook">
        <command name="docb.convertToPS"
                 parameter='pdf |pdf "/book toc /article toc" 1' />
      </configuration>

      <configuration name="Simplified DocBook">
        <command name="docb.convertToPS"
                 parameter='pdf |pdf "/book toc /article toc" 1' />
      </configuration>

      <configuration name="Slides">
        <command name="slides.convertToPS" parameter='pdf |pdf' />
      </configuration>

      <default>
        <class>com.xmlmind.xmleditapp.kit.part.PrintAction</class>
      </default>
    </context>
  </action>
```

2. command

```
<command
  name = NMTOKEN
>
  Content: class
</command>

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

Specifies a command, that is, a Java™ Object implementing interface `com.xmlmind.xmledit.gadget.Command`.

The class implementing interface `com.xmlmind.xmledit.gadget.Command` is specified by the `class` child element.

Attributes:

name
: Required. Unique name identifying the command in this GUI specification.

Example:

```
<command name="showStatistics">
  <class>com.acme.custom_xxe.ShowStatistics</class>
</command>
```

3. editorListener

```
<editorListener
  name = NMTOKEN
>
  Content: class
</editorListener>

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

Specifies an Editor listener, that is, a Java™ Object implementing interface `com.xmlmind.xmleditapp.kit.EditorListener`.

The class implementing interface `com.xmlmind.xmleditapp.kit.EditorListener` is specified by the `class` child element.

Attributes:

name
: Required. Unique name identifying the listener in this GUI specification.

Example:

```
<editorListener name="editorListener">
  <class>com.acme.custom_xxe.EditorListenerImpl</class>
</editorListener>
```

4. include

```
<include
  location = anyURI
/>
```

Include all elements contained in specified user interface specification file in current file.

The URI found in the `location` attribute may be resolved using XML catalogs.

Example 1:

```
<include location="common.xxe_gui" />
```

If the file containing the above snippet is `/home/john/.xxe/addon/gui/custom.xxe_gui`, the included file is then `/home/john/.xxe/addon/gui/common.xxe_gui`.

Example 2:

```
<include location="xxe-gui:app/default.xxe_gui" />
```

If the code of the XML editor is found in `/opt/xxe/bin/xxe_app.jar`, the included file is `jar:file:/opt/xxe/bin/xxe-app.jar!/gui/app/default.xxe_gui` because XXE dynamically adds the following rule to its XML catalog:

```
<rewriteURI uriStartString="xxe-gui:"
            rewritePrefix="jar:file:/opt/xxe/bin/xxe-app.jar!/gui/" />
```

5. layout

```
<layout
  label = non empty token
  icon = anyURI
  width = positive int
  height = positive int
>
  Content (in any order): [ insert ]?
                          [ menuBar ]?
                          [ topToolBars ]? [ bottomToolBars ]?
                          [ leftToolBars ]? [ rightToolBars ]?
                          [ leftPanes ]? [ rightPanes ]?
                          [ preferencesSheets ]?
                          [ hidden ]?
</layout>
```

Specifies the layout of visible parts: actions, tools, panes, toolBars, statusBars. Also specifies which hidden parts (given the visible parts which have been chosen) are needed to get a functioning XML Editor.

Important

A GUI specification file may contain declarations for dozens of parts, but only the parts which are referenced directly or indirectly[1] by the `layout` element are created and activated during the startup of the XML editor.

Declaring a large number of parts in a GUI specification file, even if most of these parts are *not* actually used by the GUI specified by `layout`, makes it quick and easy to built *alternate* GUIs. Simply include the file containing all these predeclared parts and define a custom `layout` making use of some of them.

Example (this is the default layout):

```
<layout width="850" height="650">
  <menuBar name="menuBar" />

  <topToolBars>
    <toolBar name="toolBar" />
    <group>
      <toolBar name="documentReferenceToolBar" />
      <tool name="nodePathTool" stretch="1" />
      <toolBar name="selectToolBar" />
    </group>
  </topToolBars>

  <bottomToolBars>
    <statusBar name="statusBar" />
  </bottomToolBars>

  <rightPanes width="0.33" topHeight="0.33">
    <pane name="editPane" selected="true" />
    <pane name="editAttributePane" position="bottom" selected="true" />
    <pane name="textSearchReplacePane" position="bottom" />
    <pane name="checkSpellingPane" position="bottom" />
    <pane name="insertCharacterPane" position="bottom" />
    <pane name="checkValidityPane" position="bottom" />
  </rightPanes>

  <preferencesSheets name="preferencesSheets" />

  <hidden>
    <!-- These are required to be able to use editPane -->
    <command name="replace" />
    <command name="insert" />
    <command name="convert" />
```

[1] Example: `layout` references a `statusBar` which references an `action`.

```
    <command name="wrap" />

    <part name="editOptionsPart" />
    <part name="spellOptionsPart" />
    <part name="spreadsheetOptionsPart" />
    <part name="webDAVOptionsPart" />

    <part name="autoSavePart" />

    <!-- Required by setStyleSheetMenuItems -->
    <action name="setStyleSheetAction" />
  </hidden>
</layout>
```

5.1. The attributes of `layout`

label
: Specifies a (desktop) name for the XML editor application.

icon
: Specifies an (desktop) icon for the XML editor application. This URI may be resolved using XML catalogs.

width, height
: Specify the *initial* size in pixels of the XML editor application. The position and size of the main window of XMLmind XML Editor are saved in *`XXE_user_preferences_dir`*`/preferences.properties`. Therefore, these attributes are only used the very first time XXE is started.

Example:

```
<layout label="Document Editor" icon="docedit.png">
  <insert />
</layout>
```

5.2. The `menuBar` child element of `layout`

```
<menuBar
  name = NMTOKEN
/>
```

Specifies which `menuBar` to use for this XML editor. The referenced menuBar must have been declared in this GUI specification.

Example:

```
<menuBar name="menuBar" />
```

5.3. The `topToolBars` and `bottomToolBars` child elements of `layout`

```
<topToolBars>
  Content: [ insert | toolBar | statusBar | tool | group ]+
</topToolBars>

<insert />

<toolBar
  name = NMTOKEN
/>

<statusBar
  name = NMTOKEN
/>

<tool
  name = NMTOKEN
/>
```

```
<group>
  Content: [ toolBar | statusBar | tool ]{2,}
</group>

Inside group:

<toolBar
  name = NMTOKEN
  stretch = non negative double : 0
/>

<statusBar
  name = NMTOKEN
  stretch = non negative double : 0
/>

<tool
  name = NMTOKEN
  stretch = non negative double : 0
/>
```

Element `topToolBars` specifies the list of `tools`, `toolBars` and `statusBars` which will be found *above* the document views. Element `bottomToolBars` specifies the list of `tools`, `toolBars` and `statusBars` which will be found *below* the document views. The tools, toolBars and statusBars referenced in these lists must have been declared in this GUI specification.

Each `tool`, `toolBar` or `statusBar` referenced in `topToolBars` or in `bottomToolBars` elements will have its own row. If you want to group several `tools`, `toolBars` and `statusBars` per row, you need to use `group` elements.

In a `group` element, an item can be ``stretched'', that is, it can be enlarged to fill all the available horizontal space. If several items are to be stretched, the numeric value of the `stretch` attribute specifies the amount of space given to each of them. An item with a large `stretch` attribute is given more space than an item with a small `stretch` attribute.

Examples:

```
    <topToolBars>
      <toolBar name="toolBar" />
      <group>
        <toolBar name="documentReferenceToolBar" />
        <tool name="nodePathTool" stretch="1" />
        <toolBar name="selectToolBar" />
      </group>
    </topToolBars>

    <bottomToolBars>
      <statusBar name="statusBar" />
    </bottomToolBars>
```

5.4. The `leftToolBars` and `rightToolBars` child elements of `layout`

```
<leftToolBars>
  Content: [ insert | toolBar | group ]+
</leftToolBars>

<insert />

<toolBar
  name = NMTOKEN
/>

<group>
  Content: [ toolBar ]{2,}
</group>

Inside group:
```

```
<toolBar
  name = NMTOKEN
  stretch = non negative double : 0
/>
```

Element `leftToolBars` specifies the list of (vertical) `toolBars` which will be found at the *left* of the document views. Element `rightToolBars` specifies the list of (vertical) `toolBars` which will be found at the *right* the document views. The toolBars referenced in these lists must have been declared in this GUI specification.

Each `toolBar` referenced in `leftToolBars` or in `leftToolBars` elements will have its own column. If you want to group several `toolBars` per column, you need to use `group` elements.

In a `group` element, an item can be ``stretched'', that is, it can be enlarged to fill all the available vertical space. If several items are to be stretched, the numeric value of the `stretch` attribute specifies the amount of space given to each of them. An item with a large `stretch` attribute is given more space than an item with a small `stretch` attribute.

Example:

```
    <leftToolBars>
      <toolBar name="configSpecificToolBar" />
    </leftToolBars>
```

5.5. The `leftPanes` and `rightPanes` child elements of `layout`

```
<leftPanes
  width = double between 0 and 1 inclusive : 0.25
  topHeight = double between 0 and 1 inclusive : 0.5
>
  Content: [ insert | pane ]+
</leftPanes>

<insert />

<pane
  name = NMTOKEN
  position = top|bottom : top
  selected = boolean : false
/>
```

Element `leftPanes` specifies the list of `panes` which will be found at the *left* of the document views. Element `rightPanes` specifies the list of `panes` which will be found at the *right* the document views. The panes referenced in these lists must have been declared in this GUI specification.

If the pane area contains several panes, these panes will be contained in a special, splittable, tabbed, container. In such case:

Attribute `topHeight`
: Specifies the size of the top area relatively to the bottom area, when the container is split in two parts.

Attribute `position` of the reference to the pane
: Specifies the position, top or bottom, of the pane when the container is split in two parts. (Specifying `top` for one or more panes and `bottom` for all the other panes will cause the container to be split in two parts.)

Attribute `selected` of the reference to the pane
: Specifies whether the tab showing the pane should be selected or not.

Otherwise, the above attributes are ignored.

In all cases, attribute `width` specifies the size of the pane area. 0 means that the pane area should be minimized (it will be hidden). 1 means that the pane area should be maximized (it will entirely fill the main window).

Examples:

```
    <leftPanes>
      <pane name="editPane"/>
    </leftPanes>

    <rightPanes>
      <pane name="editAttributePane" />
      <pane name="textSearchReplacePane" />
      <pane name="checkSpellingPane" />
      <pane name="insertCharacterPane" />
      <pane name="checkValidityPane" />
    </rightPanes>
```

Example: right pane area is split in two parts with `editPane` at top and all the other panes at bottom:

```
    <rightPanes topHeight="0.33">
      <pane name="editPane" selected="true" />
      <pane name="editAttributePane" position="bottom" selected="true" />
      <pane name="textSearchReplacePane" position="bottom" />
      <pane name="checkSpellingPane" position="bottom" />
      <pane name="insertCharacterPane" position="bottom" />
      <pane name="checkValidityPane" position="bottom" />
    </rightPanes>
```

5.6. The `preferencesSheets` child element of `layout`

```
<preferencesSheets
  name = NMTOKEN
/>
```

Specifies which preferencesSheets (set of preferencesSheets) to use for this XML editor.

If, for example, your XML editor makes use of action `editOptionsAction`, you need to declare at least one preferencesSheets in the GUI specification and you need to reference one of these declared preferencesSheets in the `layout` by the means of this `preferencesSheets` child element.

Example:

```
<preferencesSheets name="preferencesSheets" />
```

5.7. The `hidden` child element of `layout`

```
<hidden>
  Content: [ command|property|openedDocumentHook|editorListener|
             part|action|insert ]*
</hidden>

<insert />

<command
  name = NMTOKEN
/>

<property
  name = NMTOKEN
/>

<openedDocumentHook
  name = NMTOKEN
/>

<editorListener
  name = NMTOKEN
/>

<part
  name = NMTOKEN
/>
```

```
<action
  name = NMTOKEN
/>
```

Specifies which parts are needed, even if they are not visible, to make a functioning XML editor, given the visible parts which are referenced in preferencesSheets, topToolBars, rightPanes, etc.

All the child elements of `hidden` are references to parts declared elsewhere in this GUI specification.

Example:

```
  <hidden>
    <!-- These are required to be able to use editPane -->
    <command name="replace" />
    <command name="insert" />
    <command name="convert" />
    <command name="wrap" />

    <part name="editOptionsPart" />
    <part name="spellOptionsPart" />
    <part name="spreadsheetOptionsPart" />
    <part name="webDAVOptionsPart" />

    <part name="autoSavePart" />

    <!-- Required by setStyleSheetMenuItems -->
    <action name="setStyleSheetAction" />
  </hidden>
```

5.8. The `insert` descendant element of `layout`

```
<insert />
```

Adding an `insert` element to the `layout` element, or to any of the child elements of `layout` which allows this (topToolBars, bottomToolBars, leftToolBars, rightToolBars , leftPanes, rightPanes, hidden), means that the `layout` is being extended rather than being redefined.

Inside the `layout` element, an `insert` child element simply means that the `layout` is being extended. Its rank as a child is not significant.

Inside the hidden element, an `insert` child element means that the `hidden` is being extended by adding references to those found in the previous definition of this element. The rank of `insert` as a child of `hidden` is not significant.

Inside topToolBars, bottomToolBars, leftToolBars, rightToolBars, leftPanes, rightPanes elements, the `insert` child element also specifies where to insert the references found in the previous definition of these elements. The `insert` element must be the first or the last child of these elements.

Example: change label and icon:

```
  <layout label="Document Editor" icon="docedit.png">
    <insert />
  </layout>
```

Example: replace menu bar and add a tool bar at the left of the `leftPanes`:

```
  <layout>
    <insert />

    <menuBar name="customMenuBar" />

    <leftToolBars>
      <toolBar name="configSpecificToolBar" />
    </leftToolBars>
 </layout>
```

Example: insert extra tool bar `toolBar2` above the standard status bar:

```
  <layout>
    <bottomToolBars>
      <toolBar name="toolBar2" />
      <insert />
    </bottomToolBars>
  </layout>
```

6. menu

```
<menu
  name = NMTOKEN
  label = non empty token
  helpId = NMTOKEN
>
  Content: ([ insert | action | menu | separator ]+ [ menuItems ]?)
           | menuItems
</menu>

<insert />

<action
  name = NMTOKEN
/>

<menu
  name = NMTOKEN
/>

<separator />

<menuItems
  name = NMTOKEN
/>
```

Specifies a menu. A menu contains references to action, menu and menuItems elements declared elsewhere in the GUI specification. The reference to the `menuItems` element, if any, must always be the last reference contained in the `menu` element.

The `insert` child element may be used to extend the previous declaration of the menu. Without an `insert` child element, a new declaration for menu "`foo`" is understood as being a redefinition of menu "`foo`". The `insert` child element specifies where to insert the items found in the previous declaration. The `insert` element must be the first or the last child of a `menu`.

Attributes:

name
: Required. Unique name identifying the menu in this GUI specification.

label
: Required unless this declaration is an extension of the previous one (that is, `<insert/>` is used). Label of the menu.

helpId
: Online help ID of the menu.

Example: an ordinary menu:

```
  <menu name="helpMenu" label="_Help"
        helpId="helpMenu">
    <action name="helpAction" />
    <action name="contextualHelpAction" />
    <separator />
    <action name="showContentModelAction" />
    <separator />
    <action name="listBindingsAction" />
    <action name="listPluginsAction" />
```

```
    <separator />
    <action name="aboutAction" />
  </menu>
```

Example: a menu where all items are dynamic:

```
  <menu name="configSpecificMenu" label="XML"
        helpId="configSpecificMenu">
    <menuItems name="configSpecificMenuItems" />
  </menu>
```

Example: a menu which has two static items followed by dynamic items:

```
  <menu name="viewMenu" label="_View"
        helpId="viewMenu">
    <action name="addViewAction" />
    <action name="closeViewAction" />
    <menuItems name="setStyleSheetMenuItems" />
  </menu>
```

7. menuBar

```
<menuBar
  name = NMTOKEN
  helpId = NMTOKEN
>
  Content: [ insert | menu ]+
</menuBar>

<insert />

<menu
  name = NMTOKEN
/>
```

Specifies a menu bar. A menu bar contains references to menu elements declared elsewhere in the GUI specification.

The `insert` child element may be used to extend the previous declaration of the menu bar. Without an `insert` child element, a new declaration for menu bar "`foo`" is understood as being a redefinition of menu bar "`foo`". The `insert` child element specifies where to insert the menus found in the previous declaration. The `insert` element must be the first or the last child of a `menuBar`.

Attributes:

name
: Required. Unique name identifying the menu bar in this GUI specification.

helpId
: Online help ID of the menu bar.

Example: standard menu bar:

```
  <menuBar name="menuBar" helpId="menuBar">
    <menu name="fileMenu" />
    <menu name="selectMenu" />
    <menu name="editMenu" />
    <menu name="searchMenu" />
    <menu name="viewMenu" />
    <menu name="toolsMenu" />
    <menu name="configSpecificMenu" />
    <menu name="windowMenu" />
    <menu name="optionsMenu" />
    <menu name="helpMenu" />
  </menuBar>
```

Example: add extra menu `charactersMenu` at the end of the standard menu bar:

```
  <menuBar name="menuBar">
    <insert />
    <menu name="charactersMenu" />
  </menuBar>
```

8. menuItems

```
<menuItems
  name = NMTOKEN
>
  Content: class [ property ]*
</menuItems>

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

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

Specifies a dynamic set of menu items, that is, a Java™ Object implementing interface `com.xmlmind.xmleditapp.kit.AppMenuItems`.

The class implementing interface `com.xmlmind.xmleditapp.kit.AppMenuItems` is specified by the `class` child element.

`Property` child elements may be used to parametrize a newly created part. See bean properties.

Attributes:

name
: Required. Unique name identifying the dynamic set of menu items in this GUI specification.

Examples:

```
<menuItems name="recentFilesMenuItems">
  <class>com.xmlmind.xmleditapp.kit.part.RecentFilesMenuItems</class>
</menuItems>

<menuItems name="configSpecificMenuItems">
  <class>com.xmlmind.xmleditapp.kit.part.ConfigSpecificMenuItems</class>
</menuItems>
```

9. openedDocumentHook

```
<openedDocumentHook
  name = NMTOKEN
>
  Content: class
</openedDocumentHook>

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

Specifies an OpenedDocument hook, that is, a Java™ Object implementing interface `com.xmlmind.xmleditapp.kit.OpenedDocumentHook`.

The class implementing interface `com.xmlmind.xmleditapp.kit.OpenedDocumentHook` is specified by the `class` child element.

Attributes:

name
: Required. Unique name identifying the hook in this GUI specification.

Example:

```
<openedDocumentHook name="hook">
  <class>com.acme.custom_xxe.Hook</class>
</openedDocumentHook>
```

10. pane

```
<pane
  name = NMTOKEN
  icon = anyURI
  label = non empty token
  helpId = NMTOKEN
>
  Content: class [ property ]*
</part>

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

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

Specifies a pane, that is, a form like standard Edit tool or standard Attributes tool, intended to be placed at the left or at the right of the document views. The Java™ object implementing the form must be an instance of class `java.awt.Component`, implementing interface `com.xmlmind.xmleditapp.kit.AppPane`.

The class implementing interface `com.xmlmind.xmleditapp.kit.AppPane` is specified by the `class` child element.

`Property` child elements may be used to parametrize a newly created part. See bean properties.

Attributes:

name
: Required. Unique name identifying the pane in this GUI specification.

icon
: Required. Icon of the pane (the pane is generally contained in a tabbed container and this is needed for the tab showing the pane). This URI may be resolved using XML catalogs.

label
: Required. Label of the pane (the pane is generally contained in a tabbed container and this is needed for the tab showing the pane).

helpId
: Online help ID of the pane.

Example:

```
  <pane name="textSearchReplacePane" label="Search"
        icon="icons/textSearchReplacePane.gif"
        helpId="textSearchReplacePane">
    <class>com.xmlmind.xmleditapp.kit.part.TextSearchReplacePane</class>
  </pane>
```

11. part

```
<part
  name = NMTOKEN
>
  Content: class [ property ]*
</part>

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

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

Specifies a generic part, that is, a Java™ Object implementing interface `com.xmlmind.xmleditapp.kit.AppPart`. Unlike action, tool, pane, etc, generic parts are ``behind the scene workers'', which are referenced in the hidden section of a layout.

The class implementing interface `com.xmlmind.xmleditapp.kit.AppPart` is specified by the `class` child element.

`Property` child elements may be used to parametrize a newly created part. See bean properties.

Attributes:

name
: Required. Unique name identifying the part in this GUI specification.

Examples:

```
<part name="autoSavePart">
  <class>com.xmlmind.xmleditapp.kit.part.AutoSavePart</class>
</part>
```

11.1. Bean properties

Most Java™ Objects specified in `.xxe_gui` files may be parameterized using `property` child elements. A property child element specifies a *Bean* (that is, a Java™ Object) property.

Example:

```
<property name="columns" type="int" value="40" />
```

implies that the bean to be parametrized has a public method which resembles:

```
setColumns(int number)
```

Such properties are completely specific to the bean they parametrize and therefore, cannot be described in this manual.

`type`	**Corresponding Java™ type**	**Syntax of `value`**	**Example**
boolean	boolean	true, false	true
byte	byte	integer: -128 to 127 inclusive	100
char	char	a single character	a

`type`	Corresponding Java™ type	Syntax of `value`	Example
short	short	integer: -32768 to 32767 inclusive	1000
int	int	integer: -2147483648 to 2147483647 inclusive	-1
long	long	integer: -9223372036854775808 to 9223372036854775807, inclusive	255
float	float	single-precision 32-bit format IEEE 754	-0.5
double	double	double-precision 64-bit format IEEE 754	1.0
String	java.lang.String	a string	Hello, world!
Color	java.awt.Color	• *#RRGGBB* where *RR*, *GG*, *BB* are hexadecimal numbers between 0 and 255 inclusive; • or rbg(*R*, *G*, *B*) where *R*, *G*, *B* are decimal numbers between 0 and 255 inclusive; • or standard HTML/CSS named colors: black, white, red, silver, etc.	• #FFFFFF • rgb(255,255,255) • white
Font	java.awt.Font	*fontfamily* [-BOLD|BOLDITALIC|ITALIC] [*-pointsize*] Default *style* is PLAIN. Default *pointsize* is 12.	• Monospaced • Serif-14 • SansSerif-ITALIC • Serif-BOLD-10

Example (a `StatusTool` is a kind of `JTextField`):

```
<tool name="heavilyParametrizedStatusTool">
  <class>com.xmlmind.xmleditapp.kit.part.StatusTool</class>
  <property name="text" type="String" value="   Hello,   world!   " />
  <property name="editable" type="boolean" value="true" />
  <property name="focusable" type="boolean" value="true" />
  <property name="horizontalAlignment" type="int" value="RIGHT"1 />
  <property name="font" type="Font" value="Serif-BOLD-16" />
  <property name="foreground" type="Color" value="#000080" />
  <property name="background" type="Color" value="silver" />
  <property name="selectedTextColor" type="Color" value="rgb(255,0,0)" />
  <property name="caretPosition" type="int" value="5" />
  <property name="alignmentX" type="float" value="0.5" />
</tool>
```

1 Notice here the use of symbolic integer constant `javax.swing.JTextField.RIGHT`.

12. preferencesSheet

```
<preferencesSheet
  name = NMTOKEN
>
  Content: class
</preferencesSheet>

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

Specifies a preferences sheet, that is, an instance of class `com.xmlmind.xmleditapp.dialog.PreferencesSheet`.

The class derived from `com.xmlmind.xmleditapp.dialog.PreferencesSheet` is specified by the `class` child element.

Attributes:

name
: Required. Unique name identifying the sheet in this GUI specification.

Example:

```
  <preferencesSheet name="windowOptions">
    <class>com.xmlmind.xmleditapp.app.prefsheet.WindowOptions</class>
  </preferencesSheet>
```

13. preferencesSheets

```
<preferencesSheets
  name = NMTOKEN
>
  Content: [ insert | preferencesSheet ]+
</preferencesSheets>

<insert />

<preferencesSheet
  name = NMTOKEN
  label = non empty token
>
  Content: [ preferencesSheet ]*
</preferencesSheet>
```

Specifies a set of preferences sheets. This set contains references to preferencesSheet elements declared elsewhere in the GUI specification.

The `insert` child element may be used to extend the previous declaration of the set of preferences sheets. Without an `insert` child element, a new declaration for set "`foo`" is understood as being a redefinition of set "`foo`". The `insert` child element specifies where to insert the sheets found in the previous declaration. The `insert` element must be the first or the last child of a `preferencesSheets`.

Attributes:

name
: Required. Unique name identifying the set of sheets in this GUI specification.

Example: standard preferences sheets:

```
  <preferencesSheets name="preferencesSheets">
    <preferencesSheet name="openOptions">
      <preferencesSheet name="webDAVOptions" />
      <preferencesSheet name="schemaCacheOptions" />
    </preferencesSheet>
```

```
    <preferencesSheet name="saveOptions" />
    <preferencesSheet name="printOptions" />
    <preferencesSheet name="editOptions" />
    <preferencesSheet name="viewOptions" />
    <preferencesSheet name="tools" label="Tools">1
      <preferencesSheet name="spellOptions" />
      <preferencesSheet name="spreadsheetOptions" />
    </preferencesSheet>
    <preferencesSheet name="windowOptions" />
    <preferencesSheet name="generalOptions" />
  </preferencesSheets>
```

1 Unlike all the other `preferencesSheets`, `preferencesSheet` "`tools`" is not implemented in Java™. It is created on the fly for grouping `preferencesSheet` "`spellOptions`" and `preferencesSheet` "`spreadsheetOptions`".

14. property

```
<property
  name = NMTOKEN
  url = boolean
  xml:space = preserve
>text</property>
```

Specifies system property called *name*, having *`text`* as its value.

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.

Examples:

```
<property name="color">red</property>

<property name="icon.3" url="true">resources/icon.gif</property>
```

15. statusBar

```
<statusBar
  name = NMTOKEN
  helpId = NMTOKEN
>
  Content: [ insert | action | tool | separator ]+
</statusBar>

<insert />

<action
  name = NMTOKEN
/>

<tool
  name = NMTOKEN
  stretch = non negative double : 0
/>

<separator />
```

Specifies a status bar. A status bar contains references to action and tool elements declared elsewhere in the GUI specification.

The `insert` child element may be used to extend the previous declaration of the status bar. Without an `insert` child element, a new declaration for status bar "`foo`" is understood as being a redefinition of status bar "`foo`". The `insert` child element specifies where to insert the items found in the previous declaration. The `insert` element must be the first or the last child of a `statusBar`.

Attributes:

name
: Required. Unique name identifying the status bar in this GUI specification.

helpId
: Online help ID of the status bar.

A tool contained in a status bar can be ``stretched'', that is, it can be enlarged to fill all the available horizontal space. If several tools are to be stretched, the numeric value of the `stretch` attribute specifies the amount of space given of each of them. A tool with a large `stretch` attribute is given more space than a tool with a small `stretch` attribute.

Example: standard status bar:

```
  <statusBar name="statusBar" helpId="statusBar">
    <tool name="checkValidityTool" />
    <tool name="statusTool" stretch="1" />
    <action name="showLogAction" />
    <tool name="clipboardTool" />
    <tool name="clipboardContentTool" />
  </statusBar>
```

16. tool

```
<tool
  name = NMTOKEN
  helpId = NMTOKEN
>
  Content: class [ property ]*
</part>

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

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

Specifies a tool, that is, a small gadget like the standard "View Clipboard Content" tool, intended to be placed in a status bar or in an horizontal tool bar. The gadget must be an instance of class `java.awt.Component`, implementing interface `com.xmlmind.xmleditapp.kit.AppTool`.

The class implementing interface `com.xmlmind.xmleditapp.kit.AppTool` is specified by the `class` child element.

`Property` child elements may be used to parametrize a newly created part. See bean properties.

Attributes:

name
: Required. Unique name identifying the tool in this GUI specification.

helpId
: Online help ID of the tool.

Example: standard node path tool:

```
  <tool name="nodePathTool" helpId="nodePathTool">
    <class>com.xmlmind.xmleditapp.kit.part.NodePathTool</class>
    <property name="dragEnabled" value="true" type="boolean" />
  </tool>
```

17. toolBar

```
<toolBar
  name = NMTOKEN
  helpId = NMTOKEN
>
  Content: ([ insert | action | tool | separator ]+ [ toolBarItems ]?)
           | toolBarItems
</toolBar>

<insert />

<action
  name = NMTOKEN
/>

<tool
  name = NMTOKEN
/>

<separator />

<toolBarItems
  name = NMTOKEN
/>
```

Specifies a tool bar. A tool bar contains references to action, tool and toolBarItems elements declared elsewhere in the GUI specification. The reference to the `toolBarItems` element, if any, must always be the last reference contained in the `toolBar` element.

The `insert` child element may be used to extend the previous declaration of the tool bar. Without an `insert` child element, a new declaration for tool bar "`foo`" is understood as being a redefinition of tool bar "`foo`". The `insert` child element specifies where to insert the items found in the previous declaration. The `insert` element must be the first or the last child of a `toolBar`.

Attributes:

name
: Required. Unique name identifying the tool bar in this GUI specification.

helpId
: Online help ID of the tool bar.

Example: standard select tool bar:

```
  <toolBar name="selectToolBar" helpId="selectToolBar">
    <action name="selectParentAction" />
    <action name="selectChildAction" />
    <action name="selectPreviousSiblingAction" />
    <action name="selectNextSiblingAction" />
    <separator />
    <action name="selectXPathAction" />
  </toolBar>
```

Example: add a separator and tool `countWordsTool` at the end of the standard select tool bar:

```
  <toolBar name="selectToolBar">
    <insert />
    <separator />
    <tool name="countWordsTool" />
  </toolBar>
```

18. toolBarItems

```
<toolBarItems
  name = NMTOKEN
>
  Content: class [ property ]*
</toolBarItems>

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

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

Specifies a dynamic set of tool bar items, that is, a Java™ Object implementing interface `com.xmlmind.xmleditapp.kit.AppToolBarItems`.

The class implementing interface `com.xmlmind.xmleditapp.kit.AppToolBarItems` is specified by the `class` child element.

`Property` child elements may be used to parametrize a newly created part. See bean properties.

Attributes:

name
: Required. Unique name identifying the dynamic set of tool bar items in this GUI specification.

Example:

```
<toolBarItems name="configSpecificToolBarItems">
  <class>com.xmlmind.xmleditapp.kit.part.ConfigSpecificToolBarItems</class>
</toolBarItems>
```

19. translation

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

Specifies how to translate messages found in action `label` and `toolTip`, pane `label`, layout `label`, etc.

Localizing GUI specification files works as follows:

1. The `location` attribute points to a Java™ property file. Example used in the tutorial:

   ```
   <translation location="custom_gui_en.properties" />
       ...
   <layout label="Document Editor" icon="docedit.png">
       ...
   ```

   Where `custom_gui_en.properties` contains:

   ```
   layout.label=Document Editor
       ...
   ```

   The location URL specifies:

   - The reference language of the GUI specification 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: `custom_gui`. More on this below.

The reference property file is only used to map messages to message IDs. For example, `custom_gui_en.properties` specifies that message "`Document Editor`" has ID "`layout.label`".

2. If, for example, XXE is started using a French locale, a property file called `custom_gui_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, `custom_gui_fr.properties` is supposed to be contained[2] 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 the reference property file.

   For example, `custom_gui_fr.properties` contains:

```
layout.label=Éditeur de Document
     ...
```

[2] *Directly* contained, and not contained in a ``folder''. That is, "`jar tvf foo.jar`" must display `custom_gui_fr.properties` and not `foo/bar/custom_gui_fr.properties`.