XMLmind XML Editor - Customizing the User Interface

Hussein Shafie
XMLmind Software

<xmleditor-support@xmlmind.com>
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

1. Introduction ................................................................................................................................ 1
2. Tutorial ........................................................................................................................................ 4
   1. Changing the title and icon ........................................................................................................ 4
   2. Changing the About dialog box ............................................................................................... 5
   3. Adding a word count tool ......................................................................................................... 5
   4. Adding a preferences sheet for the word count tool ............................................................... 5
   5. Adding a document type specific tool bar ............................................................................. 5
   6. Adding a Characters menu ...................................................................................................... 6
3. Deployment .................................................................................................................................. 8
   1. Validation of .xxe_gui files ..................................................................................................... 8
   2. Deployment of a customization ............................................................................................... 8
   3. Specifying a custom base GUI specification which is an alternative to stock base GUI specifi-
      cation ......................................................................................................................................... 9
4. Reference .................................................................................................................................... 10
   1. action ....................................................................................................................................... 10
     1.1. Action which is a wrapper around a command ................................................................. 12
   2. command .................................................................................................................................. 13
   3. include ..................................................................................................................................... 14
   4. layout ...................................................................................................................................... 14
     4.1. The attributes of layout .................................................................................................... 16
     4.2. The menuBar child element of layout ............................................................................. 16
     4.3. The topToolBars and bottomToolBars child elements of layout .................................. 16
     4.4. The leftToolBars and rightToolBars child elements of layout ....................................... 17
     4.5. The leftPanes and rightPanes child elements of layout ................................................. 18
     4.6. The preferencesSheets child element of layout ............................................................... 19
     4.7. The hidden child element of layout ............................................................................... 19
     4.8. The insert descendant element of layout ....................................................................... 20
   5. menu ........................................................................................................................................ 21
   6. menuItems ............................................................................................................................... 26
   7. menuItems ............................................................................................................................... 26
   8. openedDocumentHook ............................................................................................................ 27
   9. pane ......................................................................................................................................... 28
   10. part ......................................................................................................................................... 29
     10.1. Bean properties .................................................................................................................. 29
   11. preferencesSheet ................................................................................................................... 31
   12. preferencesSheets .................................................................................................................. 31
   13. property .................................................................................................................................. 32
   14. statusBar .................................................................................................................................. 32
   15. tool ......................................................................................................................................... 33
   16. toolBar .................................................................................................................................... 34
   17. toolBarItems ............................................................................................................................ 35
   18. translation ............................................................................................................................... 35
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 DesktopApp.xxe_gui):

```xml
<?xml version='1.0' encoding='UTF-8'?>
  ...
  <action name="newAction" label="_New..." icon="icons/newAction.gif" accelerator="mod N">
    <class>com.xmlmind.xmleditapp.kit.part.NewAction</class>
  </action>
  ...
  <menu name="fileMenu" label="_File" helpId="fileMenu">
    <action name="newAction" />
    ...
    <menuItems name="recentFilesMenuItems" />
  </menu>
  ...
  <menuBar name="menuBar" helpId="menuBar">
    <menu name="fileMenu" />
    <menu name="selectMenu" />
    ...
  </menuBar>
  ...
  <layout width="900" height="700">
    <menuBar name="menuBar" />
  </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 by the layout element of the GUI specification file.

The archive file XXE_install_dir/bin/xxe.jar containing the code of XMLmind XML Editor (XXE for short), also contains a number of GUI specifications:

Resource /gui/DesktopApp.xxe_gui

Specifies the default user interface of the XMLmind XML Editor desktop application. By doing this, it also specifies the user interface parts which are common to all XMLmind XML Editor variants.

---

1Example: layout references a statusBar which references an action.
Resource /gui/WebStartedApp.xxe_gui

Specifies the user interface of XMLmind XML Editor when deployed using Java™ Web Start. This GUI is similar to the GUI of the desktop application, except that it has no Options → Install Add-ons and Help → Check for Updates menu items.

Resource /gui/EmbeddedApp.xxe_gui

Specifies the user interface of XMLmind XML Editor when embedded in another application. This GUI is similar to the GUI of the Java Web Start-ed application, except that its File → Quit menu item does not invoke System.exit().

Resource /gui/SingleDocApp.xxe_gui

Specifies the user interface of an embeddable single document, single document view, XML editor panel.

Resource /gui/SingleDocApp2.xxe_gui

Same as SingleDocApp.xxe_gui, except that the tool bar has New, Open, Save, Save As, Print and Close buttons.

Resource /gui/MultiDocApp.xxe_gui

Specifies the user interface of an embeddable multi document, possibly multi view per document, XML editor frame.

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

If you are curious and want to see the effect of using SingleDocApp.xxe_gui, a GUI quite different from the standard GUI of the desktop application, 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-c.bat) are found.

4. Set environment variable XXE_GUI as follows:

   C:\Program Files\XMLmind_XML_Editor\bin> set XXE_GUI=xxe-gui:SingleDocApp.xxe_gui

5. Start XXE as follows:

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

6. Drop an XML file onto the newly started XXE to open the corresponding document.

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:

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

5. Start XXE as follows:

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

6. Drop an XML file onto the newly started XXE to open the corresponding document.

Don’t worry, all this will be detailed in the deployment [8] chapter of this document.
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 → Preferences 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 found just below the standard tool bar.

6. 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 Chapter 9, Extending the GUI of XMLmind XML Editor 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
<?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">
    <translation location="custom_gui_en.properties" />
    <layout label="Document Editor" icon="docedit.png">
        <insert />
    </layout>
</gui>
```

The namespace of GUI specification elements is "http://www.xmlmind.com/xmleditor/schema/gui".

After downloading and installing the add-on called "XMLmind XML Editor Configuration Pack" using Options → Install Add-ons, the corresponding schema is found in XXE_user_preferences_dir/addon/xxe_config_pack/gui/xsd/gui.xsd.

GUI specification files can be localized by using translation [35] elements. For more information please consult the reference [35] part of this document.

The layout [14] element has label and icon attributes specifying the title and desktop icon of the XML editor.
Without the use of insert [20], 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:

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

It is a custom action [10] 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 DesktopApp.xxe_gui, suffice to do the job.

3. Adding a word count tool

tutorial/tutorial3.xxe_gui:

```xml
<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 [33] element.

Then this tool is inserted in the standard status bar (statusBar [32] element called "statusBar" found in DesktopApp.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:

```xml
<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 [31] element.

Then this sheet is inserted in the standard set of preferencesSheets (preferencesSheets [31] element called "preferencesSheets" found in DesktopApp.xxe_gui), after all the other sheets, by the means of the <insert/> facility.

5. Adding a document type specific tool bar

tutorial/tutorial5.xxe_gui:

```xml
<toolBar name="toolBar" replace="configSpecificToolBarItems" />
```
6. Adding a Characters menu

tutorial/tutorial6.xxe_gui:

```
<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">
  <action name="insertLeftAction" label="_L - &amp;#x2190;"/>
  <command name="insertString" parameter="&amp;#x2190;" />
  </action>
  <action name="insertRightAction" label="_R - &amp;#x2192;"/>
  <command name="insertString" parameter="&amp;#x2192;" />
  </action>
  <action name="insertAlphaAction" label="_A - &amp;#x03B1;"/>
  <command name="insertString" parameter="&amp;#x03B1;" />
  </action>
  <action name="insertBetaAction" label="_B - &amp;#x03B2;"/>
  <command name="insertString" parameter="&amp;#x03B2;" />
  </action>
  <action name="insertGammaAction" label="_C - &amp;#x03B3;"/>
  <command name="insertString" parameter="&amp;#x03B3;" />
  </action>
</menu>
```
1 Define actions which insert the chosen special characters by wrapping an action [10] around standard command "insertString" (see Section 49, “insertString” in XMLmind XML Editor - Commands).
2 Define the Characters menu [21] and its two submenus: the Arrows menu and the Greek menu.
3 This inserts menu "charactersMenu" after menu "editMenu" in the standard menu bar (menuBar [26] element called "menuBar" found in DesktopApp.xxe_gui). More information about attribute insert in Customizing a composite part without redefining it from scratch [23].
Chapter 3. Deployment

1. Validation of .xxe_gui files

GUI specification files may be written using XMLmind XML Editor\(^1\) or using a text editor.

If you use a text editor, do not forget to validate your GUI specification against its schema. After downloading and installing the add-on called "XMLmind XML Editor Configuration Pack" using Options → Install Add-ons, this schema is found in \(XXE\_user\_preferences\_dir/addon/xxe\_config\_pack/gui/xsd/gui.xsd\). Linux example:

\[
~/xxe7/addon$ /opt/xxe/bin/xmltool validate \(\-s ~/xxe7/addon/xxe\_config\_pack/gui/xsd/gui.xsd\) customize.xxe_gui
\]

2. Deployment of a customization

The GUI customization created during this tutorial tutorial6.xxe_gui is used here as an example:

1. Rename tutorial6.xxe_gui to customize.xxe_gui.

   Using this name is mandatory if you want XXE to dynamically discover your GUI customization during its start-up.

2. Copy directory \(XXE\_install\_dir/doc/gui/tutorial/\) and all its content (which now includes customize.xxe_gui) to directory \(XXE\_user\_preferences\_dir/addon/\).

XXE user preferences directory is:

- \$HOME/.xxe7/ on Linux.
- \$HOME/Library/Application Support/XMLmind/XMLEditor7/ on the Mac.
- %APPDATA%\XMLmind\XMLEditor7\ on Windows XP, Vista, 7 and 8.

   Example: C:\Documents and Settings\john\Application Data\XMLmind\XMLEditor7\ on Windows XP. C:\Users\john\AppData\Roaming\XMLmind\XMLEditor7\ on Windows Vista, 7 and 8.

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

3. Start XXE.

4. Clear or temporarily disable the Quick Start cache by unchecking the corresponding checkbox in Options → Preferences, Advanced|Cached Data section. More information about this cache in Section 7.11.1, “Cached data options” in XMLmind XML Editor - Online Help.

5. Restart XXE.

   Here what happens when XXE is started (with an empty or no Quick Start cache):

   a. The editor collects all files called customize.xxe_gui found in either of its two addon/ directories.

   b. It merges their contents with its base GUI specification. The base GUI specification is by default\(^2\) xxe_gui:DesktopApp.xxe_gui, which is a resource contained in xxe.jar.

   c. It creates its GUI according to this combined specification.

---

\(^1\)You’ll need to download and install the add-on called "XMLmind XML Editor Configuration Pack" in order to do that.

\(^2\)See below [9] how this default can be changed.
3. Specifying a custom base GUI specification which is an alternative to stock base GUI specification

Unlike customize.xxe_gui files which are often used, specifying a custom base GUI specification which is an alternative to stock base GUI specification is rarely needed. You probably don’t need to read this section.

Such alternate base GUI specification should start by including "xxe-gui:DesktopApp.xxe_gui" because DesktopApp.xxe_gui contains the declarations of all stock parts. Example:

```xml
<?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">
    <include location="xxe-gui:DesktopApp.xxe_gui" />
...
</gui>
```

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

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

The base GUI specification is specified by the XXE_GUI system property (i.e. `java -DXXE_GUI=xxx`). For a greater convenience, all XXE launchers define this system property as being the value of environment variable XXE_GUI.

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

1. Open a command prompt.
2. Set environment variable XXE_GUI as follows:

   `C:\...	utorial> set XXE_GUI=file:///C:/Doc...Settings/john/.../addon/my_custom_xxe.xxe_gui`

3. Start XXE as follows:

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

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

1. Open a terminal.
2. Start XXE as follows:

   `/opt/xxe/doc/gui/tutorial$ XXE_GUI=file:///home/john/.xxe7/addon/my_custom_xxe.xxe_gui \
    /opt/xxe/bin/xxe &`
Chapter 4. Reference

GUI specifications file must have a .xxe_gui suffix. Customization files automatically detected by XXE during its startup must be called customize.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
<?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

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

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.

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

Child element command

```xml
<command
    name = NMTOKEN
    parameter = string
    selectedParameter = string
    selectedValue = string : "true"
    editingContextSensitive = boolean : true
/>
```

Specifies an action which delegates its job to a command (see Chapter 6, Commands written in the Java™ programming language in XMLmind XML Editor - Commands). See example [6] in the tutorial part of this document. Explanations below [12].

Attributes:

name

Required. Unique name identifying the action in this GUI specification.
One of label and icon is required. Label of the action (an action is used to create buttons, menu items and tool bar items).

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.

Attributes icon and selectedIcon are both required for an action which acts as a toggle. 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.

Class com.xmlmind.xmleditapp.kit.ToggleOptionAction may be used implement an action which acts as a toggle. See example below [12]. This is also the case when the command child element has a selected-Parameter attribute. See Section 1.1, “Action which is a wrapper around a command” [12].

This URI may be resolved using XML catalogs.

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.

The hot key used to trigger the action.

Hot keys are specified using the following syntax:

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

```plaintext
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_ABOVEUNDER | DEAD_ACUTE | DEAD_BREVE | DEAD_CARON | DEAD_CEDILLA | DEAD_CIRCUMFLEX | DEAD_DIACRISIS | 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_PARENTHESES | LESS | M | MINUS | MODECHANGE | MULTIPLY | N |
Note that `mod` is the Command key on Mac and the Control key on other platforms.

**Important**

Such hot key cannot override a binding (factory binding or a binding defined in `customize.xxe` — see Chapter 4, *Customizing mouse and key bindings used by XXE in XMLmind XML Editor - Configuration and Deployment*).

`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.

If this attribute is specified as the empty string, the hot key normally specified by `accelerator` is suppressed on the Mac.

Example: a simple action:

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

Example: a more elaborate action:

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

Example: a action which acts as a toggle:

```xml
<action name="toggleUseURLChooserAction" label="_Use the URL Chooser">
  <icon>icons/toggleUseURLChooserAction.png</icon>
  <selectedIcon>icons/toggleUseURLChooserAction_selected.png</selectedIcon>
  <class>com.xmlmind.xmleditapp.kit.ToggleOptionAction</class>
  <property name="option" type="String" value="useURLChooser" />
  <property name="defaultValue" type="boolean" value="false" />
</action>
```

These are bean properties parameterizing `com.xmlmind.xmleditapp.kit.ToggleOptionAction`. More information in Section 10.1, “Bean properties” [29].

### 1.1. Action which is a wrapper around a command

Action which delegates its job to a command (see Chapter 6, *Commands written in the Java™ programming language in 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 [13] element;

parameter

Specifies the parameter of the command delegate, if any.

selectedIndex

Used to implement an action which acts as toggle. The command delegate is invoked with this special parameter in order to query it about its “selected” state. The command delegate is expected to return an object. The command result is converted to a string. This string is compared to the value of attribute selectedValue (by default, this value is "true"). If these strings are equal, then the command delegate is in the “selected” state. Otherwise, the command delegate is in the “unselected” state. See example below [13].

Example:

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

Example: a action which acts as a toggle:

```xml
<action name="toggleAutoSpellCheckAction" label="_Automatic Spell Checker"
  icon="icons/toggleAutoSpellCheckAction.png"
  selectedIcon="icons/toggleAutoSpellCheckAction_selected.png">
  <command name="autoSpellChecker" parameter="toggle"
    selectedParameter="isOn" selectedValue="true"
    editingContextSensitive="false" />
</action>
```

2. command

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

<class>
Java class name

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:

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

3. include

```xml
<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:

```xml
<include location="common_parts.xxe_gui" />
```

If the file containing the above snippet is `/home/john/.xxe7/addon/gui/customize.xxe_gui`, the included file is then `/home/john/.xxe7/addon/gui/common_parts.xxe_gui`.

Example 2:

```xml
<include location="xxe-gui:DesktopApp.xxe_gui" />
```

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

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

4. layout

```xml
<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: action [10]s, tool [33]s, pane [28]s, toolBar [34]s, statusBar [32]s. 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 build alternate GUls. 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):

```xml
<layout width="900" height="700">
  <menuBar name="menuBar" />

  <topToolBars>
    <toolBar name="toolBar" />
    <group name="nodePathToolBar">
      <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>
    <command name="XXE.new" />
    <command name="XXE.open" />
    <command name="XXE.save" />
    <command name="XXE.saveAs" />
    <command name="XXE.saveAll" />
    <command name="XXE.editInclusion" />
    <command name="XXE.close" />
  </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="viewOptionsPart" />
  <part name="spellOptionsPart" />
  <part name="helperApplicationsOptionsPart" />
  <part name="autoSavePart" />

  <!-- Required by setStyleSheetMenuItems -->
</layout>
```

\(^1\)Example: `layout` references a `statusBar` which references an `action`. 
4.1. 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:

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

4.2. The menuBar child element of layout

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

Specifies which menuBar to use for this XML editor. The referenced menuBar [26] must have been declared in this GUI specification.

Example:

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

4.3. The topToolBar and bottomToolBar child elements of layout

```xml
<topToolBar
    insert = non empty token
    replace = non empty token
    replaceEnd = non empty token
>  Content: [ insert | toolBar | statusBar | tool | group ]+
</topToolBar>

<insert />

<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 `tool`, `toolBar` and `statusBar` which will be found *above* the document views. Element `bottomToolBars` specifies the list of `tool`, `toolBar` and `statusBar` which will be found *below* the document views. The `tool` [33], `toolBar` [34] and `statusBar` [32]s 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 `tool`, `toolBar` and `statusBar` 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.

A `group` element may have a `name` attribute. The only use of this `name` attribute is to make simpler customizing the `toolBars` element which is the parent of the `group` element. See Customizing a composite part without redefining it from scratch [23].

Examples:

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

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

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

Element `leftToolBars` specifies the list of (vertical) `toolBar`s which will be found at the *left* of the document views. Element `rightToolBars` specifies the list of (vertical) `toolBar`s which will be found at the *right* the document views. The `toolBar` [34]s referenced in these lists must have been declared in this GUI specification.

```xml
<leftToolBars
  insert = non empty token
  replace = non empty token
  replaceEnd = non empty token
>
  Content: [ insert | toolBar | group ]+
</leftToolBars>

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

<group
  Content: [ toolBar ](2,)
</group>
```
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:

```xml
<leftToolBars>
  <toolBar name="myToolBar" />
</leftToolBars>
```

4.5. The leftPanes and rightPanes child elements of layout

```xml
<leftPanels
  width = double between 0 and 1 inclusive : 0.25
  topHeight = double between 0 and 1 inclusive : 0.5
  insert = non empty token
  replace = non empty token
  replaceEnd = non empty token
>
  Content: [ insert | pane ]+
</leftPanels>

<insert />

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

Element leftPanels specifies the list of panes which will be found at the left of the document views. Element rightPanels specifies the list of panes which will be found at the right the document views. The pane [28]s 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:

```xml
<leftPanels>
  <pane name="editPane"/>
</leftPanels>
```
Example: right pane area is split in two parts with editPane at top and all the other panes at bottom:

```xml
<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>
```

4.6. **The preferencesSheets child element of layout**

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

Specifies which preferencesSheets (set of preferencesSheet) 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:

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

4.7. **The hidden child element of layout**

```xml
<hidden insert = non empty token replace = non empty token replaceEnd = non empty token >
  Content: [ command|property|openedDocumentHook|part|action|accelerator|insert ]*
</hidden>
```

```xml
<insert />
```

```xml
<command name = NM_TOKEN />
```

```xml
<property name = NM_TOKEN />
```

```xml
<openedDocumentHook name = NM_TOKEN />
```

```xml
<part name = NM_TOKEN />
```

```xml
<action name = NM_TOKEN />
```
<accelerator
    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 [19], topToolBars [16], rightPanes [18], etc.

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

Example:

```xml
<hidden>
    <command name="XXE.new" />
    <command name="XXE.open" />
    <command name="XXE.save" />
    <command name="XXE.saveAs" />
    <command name="XXE.saveAll" />
    <command name="XXE.editInclusion" />
    <command name="XXE.close" />
    <!-- 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="viewOptionsPart" />
    <part name="spellOptionsPart" />
    <part name="helperApplicationsOptionsPart" />
    <part name="autoSavePart" />

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

The accelerator child element is a variant of the action child element. When the application has no menu bar, the accelerator (e.g. Ctrl+O) of an action referenced using the action child element is disabled, while the accelerator of an action referenced using the accelerator child element is enabled.

4.8. The insert descendant element of layout

```xml
<insert />
```

Adding an insert element to the layout element, or to any of the child elements of layout which allows this (topToolBars [16], bottomToolBars [16], leftToolBars [17], rightToolBars [17], leftPanes [18], rightPanes [18], hidden [19]), 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 [19] 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 [16], bottomToolBars [16], leftToolBars [17], rightToolBars [17], leftPanes [18], rightPanes [18], hidden [19] elements, the insert child element, the insert, replace, replaceEnd attributes may be used to customize the previous definition of these elements. More information in Customizing a composite part without redefining it from scratch [23].

Example: change label and icon:
Example: replace standard menu bar and add a custom tool bar at the left of the leftPanes:

```
<layout>
  <menuBar name="customMenuBar" />
  <leftToolBars>
    <toolBar name="customToolBar" />
  </leftToolBars>
  <insert />
</layout>
```

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

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

5. menu

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

Specifies a menu. A menu contains references to action [10], menu [21] and menuItems [26] elements declared elsewhere in the GUI specification.

Unless the referenced menuItems element has a count attribute, the reference to the menuItems element, if any, must be the last reference contained in the menu element.

The insert child element, the insert, replace, replaceEnd attributes may be used to customize to the previous definition of a menu. More information in Customizing a composite part without redefining it from scratch [23].

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:

```xml
<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:

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

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

```xml
<menu name="viewMenu" label="_View"
      helpId="viewMenu">
     <action name="addViewAction" />    
     <action name="closeViewAction" />  
     <menuItems name="setStyleSheetMenuItems" />
</menu>
```
Customizing a composite part without redefining it from scratch

The `insert` child element, the `insert`, `replace`, `replaceEnd` attributes may be used to customize to the previous definition of composite parts such as menu [21], menuBar [26], statusBar [32], toolBar [34], preferencesSheets [31], or layout groups such as topToolBars [16], bottomToolBars [16], leftToolBars [17], rightToolBars [17], leftPanes [18], rightPanels [18], hidden [19].

Extending a composite part

There are two ways to extend a composite part:

1. by using an `insert` child element;
2. by using an `insert` attribute.

Both methods cannot be used in the same element. The `insert` attribute is silently ignored when an `insert` child element has been specified.

1. Using the `insert` child element. Example:

```
<!-- ==============================================================
Let's suppose this tool bar is initially defined as follows:
<toolBar name="toolBar1">
  <action name="openAction" />
</toolBar>
-->

<toolBar name="toolBar1">
  <action name="newAction" />
  <insert />
  <action name="saveAction" />
</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:

```
<!-- ==============================================================
Let's suppose this tool bar is initially defined as follows:
<toolBar name="toolBar1">
  <action name="newAction" />
  <action name="saveAction" />
</toolBar>
-->

<toolBar name="toolBar1" insert="saveAction">
  <action name="openAction" />
</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 name of a part found in the previous definition of the same composite part. In the case of the above example, `saveAction` is the value of the `name` attribute of an action contained in the previous definition of the tool bar.

This name 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:
or by using:

```xml
<toolBar name="toolBar1" insert="after newAction">
  <action name="openAction" />
</toolBar>
```

Instead of using the name of a part, it is also possible to use `##first` or `##last`. `##first` specifies the first part referenced in the previous definition of the part container. `##last` specifies the last part referenced in the previous definition of the part container.

Example:

```xml
<toolBar name="toolBar1" insert="after ##last">
  <action name="printAction" />
</toolBar>
```

The value of the `insert` attribute may start with `ifDefined(system_property_name)`. In such case, the previously defined toolbar is extended if and only if a system property called `system_property_name` has been defined (no matter its value). Example:

```xml
<toolBar name="toolBar1"
  insert="ifDefined(XXE.Feature.Spreadsheet) after newAction">
  <action name="openAction" />
</toolBar>
```

In addition to `ifDefined(system_property_name)`, the following conditional processing constructs are also supported:

- In `ifDefined(test)`, the test is not limited to testing the existence of a system property. It is also possible to specify:

  ```xml
  system_property_name=value
  ```

  The test evaluates to true when specified system property exists and is equal to specified value.

  ```xml
  system_property_name=value
  ```

  The test evaluates to true when specified system property exists and starts with specified value.

  ```xml
  system_property_name$=value
  ```

  The test evaluates to true when specified system property exists and ends with specified value.

  ```xml
  system_property_name*=value
  ```

  The test evaluates to true when specified system property exists and contains specified value.

- It's also possible to specify `ifNotDefined(test)`. Example (a customization which is applied on Windows and on the Mac, but not on Linux):

  ```xml
  <toolBar name="toolBar1"
    insert="ifNotDefined(os.name*=Linux) after newAction">
    <action name="openAction" />
  </toolBar>
  ```
Removing or replacing some parts inside a composite part

Removing or replacing some parts inside a composite part is done by the means of the replace attribute and, optionally, also the replaceEnd attribute. The replaceEnd attribute is needed to specify a range of sibling parts.

Note that the replace and replaceEnd attributes are silently ignored when an insert child element or an insert attribute has been specified.

Remove parts example:

```xml
<!-- ====================================================
Let's suppose this tool bar is initially defined as follows:
<toolBar name="toolBar1">
  <action name="newAction" />
  <action name="saveAction" />
</toolBar>
===================================================== -->
<toolBar name="toolBar1" replace="newAction" />
```

results in the following toolBar:

```xml
<toolBar name="toolBar1">
  <action name="saveAction" />
</toolBar>
```

This could have been specified as follows:

```xml
<toolBar name="toolBar1" replace="before saveAction" />
```

or as follows:

```xml
<toolBar name="toolBar1" replace="#first" />
```

In fact, the replace and replaceEnd attributes support exactly the same values as the insert attribute. See above [23]. However, there is a pitfall. While attributes insert="before saveAction" and insert="saveAction" are equivalent, attributes replace="before saveAction" and replace="saveAction" are not equivalent.

Replace parts example:

```xml
<!-- ====================================================
Let's suppose this tool bar is initially defined as follows:
<toolBar name="toolBar1">
  <action name="newAction" />
  <action name="saveAction" />
  <action name="saveAsAction" />
</toolBar>
===================================================== -->
<toolBar name="toolBar1" replace="newAction" replaceEnd="saveAction">
  <action name="openAction" />
</toolBar>
```

results in the following toolBar:

```xml
<toolBar name="toolBar1">
  <action name="openAction" />
  <action name="saveAsAction" />
</toolBar>
```
6. menuBar

```
<menuBar
    name = NMTOKEN
    helpId = NMTOKEN
    insert = non empty token
    replace = non empty token
    replaceEnd = non empty token
>
   Content: [ insert | menu ]+
</menuBar>
```


The `insert` child element, the `insert`, `replace`, `replaceEnd` attributes may be used to customize to the previous definition of a menu bar. More information in Customizing a composite part without redefining it from scratch [23].

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` after menu `editMenu`:

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

7. menuItems

```
<menuItems
    name = NMTOKEN
    count = strictly positive int
>
   Content: class [ property ]*
</menuItems>
```

```
<class>
   Content: Java class name
```
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 [29].

Attributes:

name

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

count

The fixed number of items which will be dynamically added to this part. This attribute must not be specified for truly dynamic set of menu items.

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>
```

8. `openedDocumentHook`

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

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:
9. pane

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

<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 [29].

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:

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

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

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

<property
    name = NM_TOKEN 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 [10], tool [33], pane [28], etc, generic parts are "behind the scene workers", which are referenced in the hidden [19] section of a layout [14].

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 [29].

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>
```

### 10.1. Bean properties

Most Java™ Objects specified in `.xse_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.

<table>
<thead>
<tr>
<th>type</th>
<th>Corresponding Java™ type</th>
<th>Syntax of value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>boolean</td>
<td>true, false</td>
<td>true</td>
</tr>
<tr>
<td>byte</td>
<td>byte</td>
<td>integer: -128 to 127 inclusive</td>
<td>100</td>
</tr>
<tr>
<td>char</td>
<td>char</td>
<td>a single character</td>
<td>a</td>
</tr>
<tr>
<td>type</td>
<td>Corresponding Java™ type</td>
<td>Syntax of value</td>
<td>Example</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------</td>
<td>-----------------</td>
<td>---------</td>
</tr>
<tr>
<td>short</td>
<td>short</td>
<td>integer: -32768 to 32767 inclusive</td>
<td>1000</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
<td>integer: -2147483648 to 2147483647 inclusive</td>
<td>-1</td>
</tr>
<tr>
<td>long</td>
<td>long</td>
<td>integer: 922372036854775808 to 922372036854775807, inclusive</td>
<td>255</td>
</tr>
<tr>
<td>float</td>
<td>float</td>
<td>single-precision format IEEE 754</td>
<td>-0.5</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
<td>double-precision format IEEE 754</td>
<td>1.0</td>
</tr>
<tr>
<td>String</td>
<td>java.lang.String</td>
<td>a string</td>
<td>Hello, world!</td>
</tr>
<tr>
<td>Color</td>
<td>java.awt.Color</td>
<td>• #RRGGBB where RR, GG, BB are hexadecimal numbers between 0 and 255 inclusive; • or rgb(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.</td>
<td>• #FFFFFF • rgb(255,255,255) • white</td>
</tr>
<tr>
<td>Font</td>
<td>java.awt.Font</td>
<td>font family [ -BOLD</td>
<td>BOLDITALIC</td>
</tr>
</tbody>
</table>

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

```xml
<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"/>
<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`.

Reference
11. preferencesSheet

```xml
<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:

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

12. preferencesSheets

```xml
<preferencesSheets
  name = NMTOKEN
  insert = non empty token
  replace = non empty token
  replaceEnd = non empty token
>
  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 [31] elements declared elsewhere in the GUI specification.

The `insert` child element, the `insert`, `replace`, `replaceEnd` attributes may be used to customize to the previous definition of a set of preferences sheets. More information in Customizing a composite part without redefining it from scratch [23].

Attributes:

name

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

Example: standard preferences sheets:
Unlike all the other preferenceSheets, preferenceSheet "tools" is not implemented in Java™. It is created on the fly for grouping preferenceSheet "spellOptions" and preferenceSheet "spreadsheetOptions".

13. 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>

14. statusBar

<statusBar
    name = NMTOKEN
    helpid = NMTOKEN
    insert = non empty token
    replace = non empty token
    replaceEnd = non empty token
    >
    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 [10] and tool [33] elements declared elsewhere in the GUI specification.

The `insert` child element, the `insert`, `replace`, `replaceEnd` attributes may be used to customize to the previous definition of a status bar. More information in Customizing a composite part without redefining it from scratch [23].

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:

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

15. tool

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

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

<property
  name = NMTOKEN matching [\_\_a-zA-2]\\[\_\_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 [29].

Attributes:

name

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

Online help ID of the tool.

Example: standard node path tool:

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

16. toolBar

```xml
<toolBar
  name = NMTOKEN
  helpId = NMTOKEN
  insert = non empty token
  replace = non empty token
  replaceEnd = non empty token
>
  Content: [ insert | action | tool | separator | toolBarItems ]+
</toolBar>
```

Specifies a tool bar. A tool bar contains references to action [10], tool [33] and toolBarItems [35] elements declared elsewhere in the GUI specification.

Unless the referenced toolBarItems element has a count attribute, the reference to the toolBarItems element, if any, must be the last reference contained in the toolBar element.

The insert child element, the insert, replace, replaceEnd attributes may be used to customize to the previous definition of a tool bar. More information in Customizing a composite part without redefining it from scratch [23].

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:

```xml
<toolBar name="selectToolBar" helpId="selectToolBar">
  <action name="selectParentAction" />
  <action name="selectChildAction" />
  <action name="selectPreviousSiblingAction" />
  <action name="selectNextSiblingAction" />
  <separator />
</toolBar>
```
Example: add a separator and tool `countWordsTool` at the end of the standard select tool bar:

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

### 17. `toolBarItems`

```xml
<toolBarItems
  name = NMTOKEN
  count = strictly positive int
>
  Content: class [ property ]*
</toolBarItems>
```

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 [29].

**Attributes:**

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

- **count**
  - The fixed number of items which will be dynamically added to this part. This attribute must not be specified for truly dynamic set of tool bar items.

Example:

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

### 18. `translation`

```xml
<translation
  location = anyURI matching [path/资源名称_lang.properties]
/>
```

Localizing GUI specification files works as follows:

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

   ```xml
   <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:

   - 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 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 
   ...
   ```

---

36

*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`. 