Menu Editor

Plug-in Guide

2.2.0.136470


vizrt

Disclaimer

Vizrt provides this publication “as is” without warranty of any kind, either expressed or implied.

This publication may contain technical inaccuracies or typographical errors. While every precaution has been taken in the preparation of this document to ensure that it contains accurate and up-to-date information, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained in this document.

Vizrt’s policy is one of continual development, so the content of this document is periodically subject to be modified without notice. These changes will be incorporated in new editions of the publication. Vizrt may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

Vizrt may have patents or pending patent applications covering subject matters in this document. The furnishing of this document does not give you any license to these patents.

Technical Support

For technical support and the latest news of upgrades, documentation, and related products, visit the Vizrt web site at www.vizrt.com.

Last Updated

18.02.2013


vizrt

Table of Contents


vizrt

1 Introduction

The Menu Editor plug-in is a framework for creating and maintaining the navigational structure of Escenic publications. It includes the following main components:

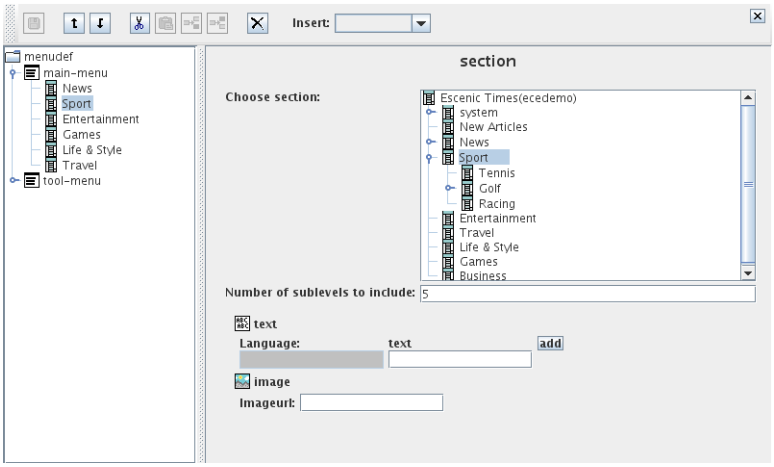
- A menu editor for creating and modifying menus. This editor is a web application that can be accessed from within Web Studio.
- A set of Java objects to represent the menus defined with the menu editor.
- A tag library that provides template developers with an easy way of accessing the menu beans.

A **menu** is a list of menu items that can be displayed for navigation purposes in a publication, typically as links. Each menu item usually has at least one **label** (the text that will be displayed in the publication). Menu items can, however, have multiple labels in different languages. They can also have image URLs, which can be used to make graphical links.

Here is a typical publication menu:

This menu (displayed in the blue upper bar) contains 6 menu items, one of which - **Sport** - is currently selected, resulting in the display of three sub-items in the lower bar.

Here is the same menu as it appears in the menu editor:



The menu (called **`main-menu`**) is defined in the tree view on the left. As you can see, it contains 6 menu items corresponding to the items in the displayed menu. You can see that the selected **`Sport`** menu item is associated with the publication's **`Sport`** section (shown in the tree view on the right), and that the **`Sport`** section has three subsections that correspond to the items in the publication sub-menu. Precisely how the menu definition, publication structure and displayed menu are related will be discussed in the rest of this manual.

The menu editor allows you to create a number of different menu item types:

section
For creating links to publication sections (the most commonly-used type).

article
For creating links to individual content items.

link
For creating links to specific URLs (usually external to the publication).

space
For inserting spaces in menus.

includeMenu
To enable re-use of menus.

Menus defined with the menu editor are stored in a publication resource file called **`menu`**. This resource is an XML file, and its structure is defined in an XML DTD supplied with the plug-in. This means that you do not have to use the menu editor to create menus: you can edit and upload them in the same way as other publication resources if you so wish.

2 Installation

The following preconditions must be met before you can install Menu Editor 2.2.0.136470:

- The Content Engine is installed and in working order.
- The Escenic assembly tool has been extracted and successfully used to set up a test EAR file as described in the **Escenic Content Engine Installation Guide**.
- The Content Engine's **XML Editor** plug-in has been installed. The Menu Editor plug-in depends on the XML Editor plug-in, and will not work without it.
- You have the required plug-in distribution file **`menu-editor-dist-2.2.0.136470.zip`**.

In a multiple-server environment:

- The Menu Editor plug-in must be installed on **all** servers.
- The Escenic event mechanism must be working correctly.

2.1 Install Menu Editor

In the following description, *escenic-home* refers to the server folder in which the Content Engine is installed.

Installing Menu Editor on the server involves the following steps:

1. **Make sure there is a plug-in folder:** If the folder *escenic-home*`/plugins` does not already exist on your server, create it. If for some reason you need to create the plug-in folder in some other location, edit the *escenic-home*`/assemblytool/assemble.properties` file and set the `plugins` property accordingly. For example:

   ```
   plugins=escenic-home/my/plugin/folder
   ```

   This folder will be referred to as *plugin-home* in the rest of this manual.
2. **Unpack the Menu Editor distribution:** Unpack the Menu Editor distribution file to *plugin-home*. This will result in the creation of a *plugin-home*`/menu-editor` folder.
3. **Rebuild the Content Engine:** Build the Escenic enterprise archive by entering the following commands:

   ```
   cd escenic-home/assemblytool
   ant ear
   ```

   The assembly tool will then add the Menu Editor plug-in to the Content Engine's classpath, including default configuration files and any required web application components.

4. **Deploy the Content Engine:** Deploy the new EAR file. For general instructions on how to deploy the EAR file on different application servers, see the **Escenic Content Engine Installation Guide**.
5. **Verify the plug-in installation:** See **section 2.2** for details of how to verify plug-in installations.

If the application server does not support EAR-based deployment, then all the JAR files located in the *plugin-home*`/menu-editor/lib` folder must be added to the application server's classpath. All the WAR files that have been rebuilt with the assembly tool should be redeployed.

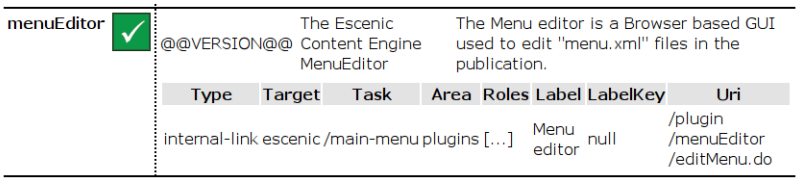
2.2 Verify The Installation

To verify the status of the Menu Editor plug-in, open the Escenic Admin web application (usually located at `http://`*server*`/admin`) and click on **View installed plugins**. The status of all currently installed plug-ins is shown here, and indicated as follows:

The plug-in is correctly installed.

The plug-in is not correctly installed.

So if the Menu Editor plug-in is correctly installed, you should see something like this in the displayed plug-in list:



If this is the case, then you should be able to use the Menu Editor to define a menu structure, as described in **chapter 3**.

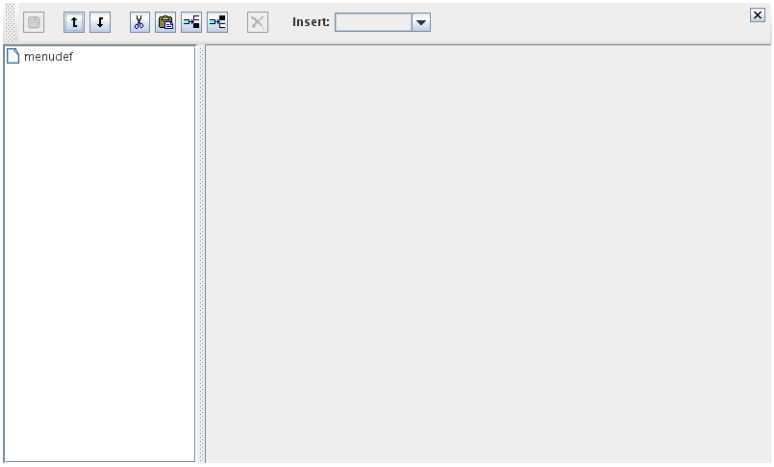
3 Using The Menu Editor

The menu editor is an XML editor that allows you to easily create a **menu** resource that is compliant with the menu DTD.

To start the menu editor:

1. Start Web Studio.
2. Select **Menu Editor** from the **Components** menu.

If you have not yet created a menu, then an empty window is displayed, as follows:



A message is also displayed indicating that a new **menu** resource will be created.

The editor window has three main components:

Toolbar
Containing various editing tools. See **section 3.1** for details.

Element tree
A collapsible tree view showing the elements in the **menu** resource. See **section 3.1.1** for details.

Attribute editor
Displays the attributes of the element that is currently selected in the element tree. See **section 3.1.2** for details.

The basic method of working with the editor is as follows:

1. Select an element in the element tree (initially, **menudef** is the only element you can choose).
2. Insert a child element under the selected element by selecting an option from the **Insert** pull-down menu in the toolbar.
3. Specify the new element's attributes using the attribute editor.

The insert function ensures that only valid elements are inserted, and the attribute editor ensures that the correct attributes are defined for each element.

3.1 The Toolbar

The toolbar contains the following tools:

Saves any changes you have made.

Moves the currently selected element up one place.

Moves the currently selected element down one place.

Cuts the currently selected element and places it on the clipboard.

Inserts the clipboard contents as a child of the currently selected element.

Inserts the clipboard contents before the currently selected element.

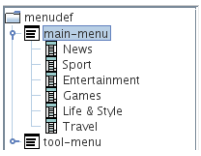
Inserts the clipboard contents after the currently selected element.

Deletes the currently selected element and all its children.

Inserts an element of the type selected from the list as a child of the currently selected element.

3.1.1 The Element Tree

The pane on the left hand side of the editor contains a tree showing the current structure of the **`menu`** resource. The tree can be expanded and collapsed as required.



The elements are represented by different icons according to their type. Right-clicking on an element displays a context menu containing a subset of the options displayed in the toolbar.

To insert a new element:

1. Select the element into which you want to insert a new element.
2. Select the **Insert** tool in the toolbar. A pull-down menu is displayed listing the element types you can insert at this point.
3. Select the element type you want to insert.

A new element of the selected type is added to the tree and automatically selected. Use the attribute editor (see **section 3.1.2**) to set the new element's attributes.

The element tree may contain the following element types:

`menu`
The root **`menudef`** element may contain any number of menu elements, each representing a different menu.

`section`
The **`section`** element represents a menu option associated with one of the publication's sections (and optionally, submenus associated with the section's sub-sections).

`article`
The **`article`** element represents a menu option associated with one of the content items in the publication.

`link`
The **`link`** element represents a link to an external URL.

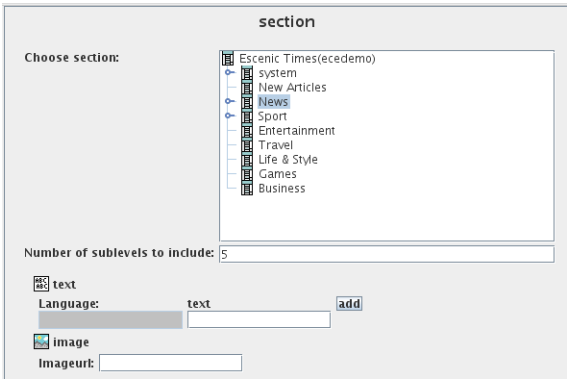
`space`
The **`space`** element represents a spacer of some kind in a menu.

`includeMenu`
The **`includeMenu`** element can be used to include one menu in another, thus making it possible to re-use menus.

3.1.2 The Attribute Editor

The pane on the right contains an attribute editor that you can use to set the attributes of the currently selected element. The contents of this pane will

therefore depend on what kind of element is currently selected. For a section element, you might see something like this:



The attributes of the different element types are described in the following sections.

3.1.2.1 Menu Attributes

Only one attribute, **Name of menu** is displayed for `menu` elements. The name specified here determines the `menuName` attribute that template developers will need to specify in order to retrieve the menu. See **section 4.1** for further information about this.

3.1.2.2 Section Attributes

The following attributes are displayed for `section` elements:

Choose section
A reference to the publication section that the element is to represent. The attribute editor displays the publication's section structure, from which you can select the required section.

Number of sublevels to include
The selected section may own sub-sections, which may in turn have sub-sections of their own. This option determines how many levels of this subtree are to be included with the section. Note that setting this attribute does not automatically create a multi-level menu in the publication: it creates a multi-level menu object, giving the template developer the opportunity to create multi-level menus if that is what is required.

text
This is actually a sub-element rather than an attribute, and is optional. A `section` element can have 0 or more `text` elements, each of which can be assigned a different **Language** attribute. If you don't specify a text element, then the section title is used by default. If, however, you want to supply alternative text for the menu item, then you can use this element to do so, one element for each language required.

image
This is also actually a sub-element rather than an attribute, and is optional. You can use it to supply the URL of an image for use in the menu link. Whether or not the image is actually used is determined by the template programmer.

3.1.2.3 Article Attributes

The following attributes are displayed for **`article`** elements:

The id of the article
A reference to the publication content item that the element is to represent.

text
This is actually a sub-element rather than an attribute, and is optional. An **`article`** element can have 0 or more **`text`** elements, each of which can be assigned a different **Language** attribute. If you don't specify a text element, then the referenced content item's title is used by default. If, however, you want to supply alternative text for the menu item, then you can use this element to do so, one element for each language required.

image
This is also actually a sub-element rather than an attribute, and is optional. You can use it to supply the URL of an image for use in the menu link. Whether or not the image is actually used is determined by the template programmer.

3.1.2.4 Link Attributes

The following attributes are displayed for **`link`** elements:

Url
The URL to link to.

text
This is actually a sub-element rather than an attribute, and is optional. A **`link`** element can have 0 or more **`text`** elements, each of which can be assigned a different **Language** attribute. If you don't specify a text element, then the URL is used by default. If, however, you want to supply alternative text for the menu item, then you can use this element to do so, one element for each language required.

image
This is also actually a sub-element rather than an attribute, and is optional. You can use it to supply the URL of an image for use in the menu link. Whether or not the image is actually used is determined by the template programmer.

3.1.2.5 Space Attributes

The following attributes are displayed for **`link`** elements:

text
This is actually a sub-element rather than an attribute, and is optional. A **space** element can have 0 or more **text** elements, each of which can be assigned a different **Language** attribute. If you want to supply text for the spacer, then you can use this element to do so, one element for each language required.

image
This is also actually a sub-element rather than an attribute, and is optional. You can use it to supply the URL of an image to be used as a spacer. Whether or not the image is actually used is determined by the template programmer.

3.1.2.6 IncludeMenu Attributes

The following attributes are displayed for **includeMenu** elements:

Publication
The name of the menu from which a menu is to be included.

Include menu with name
Select the name of the menu to be included from the pull-down list.

In the current version of the menu editor, the **Include menu with name** list does not respond correctly to changes in the **Publication** field: it only ever lists menus from the current publication. To work around this limitation:

1. Create an empty menu in the current publication with the same name as the external menu you want to include.
2. Insert an **includeMenu** element at the point where you want to include the external menu.
3. Enter the name of the external publication in the **includeMenu** element's **Publication** field.
4. Select the dummy menu name from the **includeMenu** element's **Include menu with name** list.

Alternatively, you can edit the **menu** resource by hand (see **chapter 6**).

---

4 Template Development

Menus defined with the menu editor are stored in a publication's **menu** resource, an XML file stored in the publication's **META-INF/escenic/publication-resources/escenic/plugins** folder. The menus defined in the file are made available to the template developer as **MenuItem** Java objects, which can be accessed using the **menu** tag library. This tag library is supplied as part of the Menu Editor plugin.

The menus created using the menu tag library can potentially contain large amounts of dynamic content, in which case rendering them might become a time-consuming process. It such cases it may be a good idea to use the **<util:cache>** JSP tag to improve menu performance.

Caching the menus may have a significant effect, since menus are usually identical for all page views in a section; sometimes even for all page views.

4.1 A Simple Example

The **bean** and **logic** tag libraries used in the following example are standard Apache Struts tag libraries. For information about Struts, see http://struts.apache.org/.

First the required tag libraries are imported, and required publication and section variables are created.

```
<%@ taglib uri="http://www.escenic.com/taglib/escenic-menu" prefix="MENU" %>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="BEAN" %>

<BEAN:define id="pub" name="apipublication"
                      type="neo.xredsys.api.Publication" />
<BEAN:define id="sec" name="section"
                      type="neo.xredsys.api.Section" />
```

A menu object is then retrieved from the current publication and made available as a scripting variable.

```
<MENU:get id="menu" menuName="main" pubId='<%="" + pub.getId()%>'/>
```

The value specified with the **menuName** attribute must be the name of one of the publication's menus, as defined in the Menu Editor. The retrieved menu object is made available as a scripting variable called **menu**.

The following code iterates over the entries, displaying a link to each section in the menu.

```
<MENU:iterate id="currentObject" menu="<%=menu%>"
              currentSectionID="<%= sec.getId() %>">
  <MENU:sectionView lang="no" levelName="menu-level">
    <a href='<BEAN:write name="url"/>'><BEAN:write name="text"/></a>
  </MENU:sectionView>
</MENU:iterate>
```

The **iterate** tag executes the code in its body once for each menu item. The **sectionView** tag is therefore executed once for item in the menu, but only actually does anything for menu items of type **section**. Other tags are available for handling other menu item types:

articleView
For handling **article** menu items.

urlView
For handling **link** menu items.

spaceView
For handling **space** menu items.

4.2 A More Complete Example

Here is a more complete **iterate** tag, which renders all types of menu items:

```
<MENU:iterate id="currentObject"  menu="&lt;%=menu%&gt;" currentSectionID="<%= sec.getId() %>">
  <MENU:sectionView>
    <a class="section" href='<BEAN:write name="url"/>'>
      <BEAN:write name="text"/>
    </a>
  </MENU:sectionView>
  <MENU:articleView>
    <a class="article" href='<BEAN:write name="url"/>'>
      <BEAN:write name="text"/>
    </a>
  </MENU:articleView>
  <MENU:urlView>
    <a href='<BEAN:write name="url"/>'>
      <BEAN:write name="text"/>
    </a>
  </MENU:urlView>
  <MENU:spaceView>
    <BEAN:write name="text"/>
  </MENU:spaceView>
</MENU:iterate>
```

This example is still, however, very simple. It renders links and sets the HTML class attributes to appropriate values for each menu item type, but does not do much more. The **sectionView**, **articleView**, **urlView** and **spaceView** tags all have a number of attributes that can be used in various ways.

You can also directly access the object referenced by the menu item, by using the **currentObject** variable exposed by the **iterate** tag. For a section, the **currentObject** is a **SectionItem**, which has direct access to the section:

```
<MENU:sectionView>
  <a href='<BEAN:write name="url"/>'
     class="<BEAN:write name="currentObject"
                  property="section.parameter(menuclass)"/>">
    <BEAN:write name="text"/>
  </a>
</MENU:sectionView>
```

The property called **section.parameter(menuclass)** retrieves the section parameter called 'menuclass' from each section in the menu. This way, the top level section could define a default value, and entire section trees could be configured to a different value.

Here is another example that uses the information in a content item field to enrich the menu.

```
<%@ taglib uri="http://www.escenic.com/taglib/escenic-article" prefix="article"%>
<MENU:articleView>
  <BEAN:define id="articleItem" name="currentObject"
               type="com.escenic.menu.ArticleItem" />

  <a class="article" href='<BEAN:write name="url"/>'>
    <BEAN:write name="text"/>
  </a>

  <ARTICLE:use articleId="<%=articleItem.getArticleId()%>">
    <ARTICLE:field field="menutext"/>
  </ARTICLE:use>

</MENU:articleView>
```

In this example, the content item's **menutext** field is retrieved. However, the same method could be used to retrieve related images or an icon. The menu could even be expanded to include related content items.

4.3 Using The View Tag Library

The following example uses the **view:iterate** tag to iterate over the menu items instead of the specialized **menu:iterate** tag. This method requires some experience with Java and makes direct use of the **MenuItem** class.

First of all three tag libraries are declared:

```
<%@ page language="java" %>
<%@ taglib uri="http://www.escenic.com/taglib/escenic-menu" prefix="MENU" %>
<%@ taglib uri="http://www.escenic.com/taglib/escenic-util" prefix="UTIL" %>
<%@ taglib uri="http://www.escenic.com/taglib/escenic-view" prefix="VIEW" %>
```

The **menu:use** tag is used to retrieve the menu and assign it to the scripting variable **menu** in the the tag body. This variable is suitable for use with the **view** tag library, and **view:iterate** to iterate through its contents. This loop flattens the menu hierarchy to a list of plain links:

```
<MENU:use id="menu" treeName="main">
  <VIEW:iterate id="menuItem" view="<%= menu %>"
                type="com.escenic.menu.MenuItem">
    <a href="<UTIL:valueof param="menuItem.URL" />">
      <UTIL:valueof param="menuItem.text"> </UTIL:valueof>
    </a>
  </VIEW:iterate>
</MENU:use>
```

menu:iterate, used in the first example is a specialized iteration tag for menus. It is very simple to use and the **menu** tag library has specialized tags for rendering different types of menu items. **view:iterate**, on the other hand, offers full control over the iteration process but is more difficult to use. The **view** tag library does not have specialized rendering tags, but on the other hand it allows you to navigate the menu structure. You can, for example, write code to selectively expand parts of the tree if relevant, or highlight all **parent sections** (i.e. the path) to create a "breadcrumbs" menu.

The key to this flexibility is the **<view:relationships>** tag. This tag lets you find out whether two menu items are related to one another, and if so, how.

In the following example, **`menu:item`** is used to retrieve the current menu item (that is, the menu item representing the current page view) and assign it to the variable **`base`**. **`view:iterate`** then iterates through the menu items in the menu and uses **`view:relationships`** to return the relationship between each menu item and **`base`** in the form of a **`Relation`** object.

```
<MENU:use id="menu" treeName="main">
  <MENU:item id="base"/>
  <VIEW:iterate id="menuItem">
    <VIEW:relationships id="relation" name="base"/>
    <li class="<%=relation.getState()%>">
      <%=menuItem.getText()%>
    </li>
  </VIEW:iterate>
</MENU:use>
```

The **`Relation`** object's **`getState()`** method returns one of the following values for each menu item:

isCurrent
: This item is the current menu item.

isSibling
: This item is a sibling of the current menu item

isChild
: This item is a child of the current menu item.

isParent
: This item is the parent of the current menu item.

isInPath
: This item is an ancestor of the current menu item.

isParentInPath
: This item's parent is an ancestor of the current menu item.

isDefault
: This item is not closely related to the current menu item.

These states are not mutually exclusive: any parent node is also an ancestor, for example, and other duplications might occur. There are therefore boolean properties which can be read from the **`Relationships`** object for each state. For details, see the Javadoc for the class **`neo.util.tree.Relationships`**.

Other information that can be obtained from the **`Relationships`** object includes:

- Does this item have any children? This may be so irrespective of whether or not the section referenced by the menu item has sub-sections, since the menu definition created with the menu editor may explicitly exclude the creation of menu items for subsections (see **section 3.1.2.2**). Conversely, article menu items can have children if they have been explicitly created in the menu editor.
- Does this item have siblings, or is it the only child of its parent?

This technique can easily be used to create a "breadcrumbs" menu, for example:

```
<MENU:use id="menu" treeName="main">
  <MENU:item id="base"/>
  <VIEW:iterate id="menuItem">
    <VIEW:relationships id="relation" name="base"/>
    <UTIL:equal name="relation" property="ancestor" value="true">
      <%=menuItem.getText()%> &gt;
     </UTIL:equal>
  </VIEW:iterate>
</MENU:use>
```

4.4 Accessing Menu Items Directly

When iterating through menus, either using the **`menu:iterate`** tag, or **`view:iterate`** tag, the items exposed by the iteration are all **`com.escenic.menu.MenuItem`** objects. You can use inline Java code to directly access these objects.

You could, for example, use code like this to display images for items that have them:

```
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="LOGIC"%>
.
.
.
<MENU:use id="menu" treeName="main">
  <VIEW:iterate id="menuItem">
    <LOGIC:present name="menuItem" property="imageURL">
      <img src="<%=menuItem.getImageURL()%>">
    </LOGIC:present>
    <LOGIC:notPresent name="menuItem" property="imageURL">
      <%=menuItem.getText()%>
    </LOGIC:notPresent>
  </VIEW:iterate>
</MENU:use>
```

It is also possible to use the **`MenuItem`** obejcts to access the Escenic objects they reference. This code, for example renders some of the content of a section:

```
<LOGIC:present name="menuItem" property="sectionId">
  <BEAN:define id="menusection" name="menuItem" property="section"/>
  <SECTION:use name="menusection">
    <TEMPLATE:insert typeName="group" groupName="menu" />
  </SECTION:use>
</LOGIC:present>
```

`logic:present` is used to check whether the current menu item references a section. If it does, then the section is retrieved and displayed as part of the menu.

5 Taglib Reference

The Menu Editor plug-in includes one tag library, called **menu**.

5.1 menu Tag Library

This tag library contains tags for retrieving menus and displaying menu items.

5.1.1 menu:iterate

Iterates through all menuitems in a menu

Syntax

```
<menu:iterate
   currentSectionID="..."?
   depth="..."?
   id="..."
   menu="..."
   startID="..."?>
   <menu:sectionView.../>   <menu:urlView.../>   <menu:spaceView.../>   <menu:articleView.../>
</menu:iterate>
```

Attributes

id, mandatory, no runtime expressions
The current object when iterating the menu

menu, mandatory
The menu to iterate

currentSectionID
The id of the current section.

depth
Defines how many levels to iterate

startID
Specifies which element in the menutree we should start iterating from.

Scripting variable (id)

A scripting variable will be defined using the value of the **id** attribute as its name. The variable is of type **java.lang.Object**.

5.1.2 menu:get

Retrieves a menu from the menu manager. This tag looks up the menu name in the menu manager, and retrieves that menu. The menu is then made available as a scripting variable.

The name of the menu to retrieve is found using the following algorithm:

1. The **menuName** section parameter of the section specified by **sectionName** and **pubId**. If the section does not exist, skip this item. If the section exists and has a **menuName** section parameter, the corresponding menu will be retrieved. If the section parameter is missing, or names a non-existent menu, no menu (null) will be returned.
2. The **menuName** section parameter of the section specified by **sectionId**. If the section does not exist, skip this item. If the section exists and has a **menuName** section parameter, the corresponding menu will be retrieved. If the section parameter is missing, or names a non-existent menu, no menu (null) will be returned.
3. The menuName attribute is the final fallback, if the previous sections don't exist. The menuName is simply the name of the menu in the menu manager, for the publication. If menuName is unspecified, or names a non-existent menu, no menu (null) will be returned.

Syntax

```
<menu:get
   id="..."
   menuName="..."?
   pubId="..."?
   sectionId="..."?
   sectionName="..."?>
   ...
</menu:get>
```

Attributes

id, mandatory, no runtime expressions
The name of the scripting variable to create, which will hold the menu.

menuName
The name of the menu to retrieve. The name of the menu can be specified directly by using the **menuName** attribute. This menu name must correspond exactly to the name of the menu. Menu names are case sensitive.

If sectionId or sectionName are specified and exist, then menuName will be ignored. **menuName** will only be used if the specified section ID or name doesn't exist, or if they aren't specified at all.

pubId
The id of the publication to get the menu from. This attribute should be used in conjunction with **sectionName** or **menuName**.

sectionId
The id of the section to get the menu from. The menu to use in a given section is defined in section.properties. Use this in combination with the section parameter called **menuName**. The specified section's **menuName** parameter will be used to figure out the name of the menu to retrieve.

If the attribute is not set, or the specified section does not exist, the tag will attempt to use the **menuName** to look up the menu directly, if specified.

sectionName
The unique name of the section to get the menu from. The menu to use in a given section is defined in section.properties. Use this in combination with the section parameter called **menuName**. The specified section's **menuName** parameter will be used to figure out the name of the menu to retrieve.

If the attribute is not set, or the specified section does not exist, the tag will attempt to use the sectionId to look up the section, if specified. The **pubId** attribute must be set to use this attribute.

Scripting variable (id)

A scripting variable will be defined using the value of the **id** attribute as its name. The variable is of type **com.escenic.menu.Menu**.

5.1.3 menu:sectionView

Views the current sectionMenuitem. This tag defines the following variables: isSibling,isNormal,isInPath,isCurrent,name,url,imageUrl,text,levelName,level, sectionName,sectionUrl

Syntax

```
<menu:sectionView
   lang="..."?
   levelName="..."?>
   ...
</menu:sectionView>
```

Attributes

lang
This attribute defines which language specific text to use. If language is set, the tag returns the text of a given language, else it returns default languagetext.

levelName
Defines the name of the level. The current level will be appended to the value of this attribute

5.1.4 menu:spaceView

Views the current spaceMenuitem. This tag defines the following variables: isSibling,isNormal,isInPath,isCurrent,name,url,imageUrl,text,levelName,level

Syntax

```
<menu:spaceView
   lang="..."?
   levelName="..."?>
   ...
</menu:spaceView>
```

Attributes

lang
This attribute defines which language specific text to use. If language is set, the tag returns the text of a given language, else it returns default languagetext.

levelName
Defines the name of the level. The current level will be appended to the value of this attribute

5.1.5 menu:urlView

Views the current urlMenuitem. This tag defines the following variables: isSibling,isNormal,isInPath,isCurrent,name,url,imageUrl,text,levelName,level

Syntax

```
<menu:urlView
   lang="..."?
   levelName="..."?>
   ...
</menu:urlView>
```

Attributes

lang
This attribute defines which language specific text to use. If language is set, the tag returns the text of a given language, else it returns default languagetext.

levelName
Defines the name of the level. The current level will be appended to the value of this attribute

5.1.6 menu:articleView

Views the current articleMenuitem. This tag defines the following variables: isSibling,isNormal,isInPath,isCurrent,name,url,imageUrl,text,levelName,level, articleId

Syntax

```
<menu:articleView
   lang="..."?
   levelName="..."?>
   ...
</menu:articleView>
```

Attributes

lang
This attribute defines which language specific text to use. If language is set, the tag returns the text of a given language, else it returns default languagetext.

levelName
Defines the name of the level. The current level will be appended to the value of this attribute

5.1.7 menu:use

Creates a generic view of the menu, suitable for iteration using the **view** tag library. This method of iteration does not use the **sectionView** or similar tags. Rather, the view is iterated over using the core **view:iterate** tag.

By using this tag along with the **<menu:item>** tag, it is possible to make the menu context-sensitive. See the documentation for **>view:relationships<** for more information.

Syntax

```
<menu:use
   id="..."
   publication="..."?
   publicationId="..."?
   treeName="...">
   <menu:item.../>
</menu:use>
```

Attributes

id, mandatory, no runtime expressions
Name of the scripting variable to export. The variable will be of type com.escenic.common.util.tree.View. The variable will is suitable for recursion using **<view:iterate>**.

publicationId
Id of the publication to get the tree, if different from the current publication. If not specified, the request's current publication will be used.

publication
The name of publication to get the tree, if different from the current publication. If not specified, the request's current publication will be used.

treeName, mandatory
Name of the tree. This is the actual name of the menu to use.

5.1.8 menu:item

Look in the menu for a specific menu item within the context of a **use** tag, and make this item available as a scripting variable. The normal use of this tag is to search the menu for the menu item to highlight:

```
<menu:item id="currentitem"/>
```

This will take the current request (request for an article or section, and look for it in the enclosing **<menu:use>** menu. If the menu item is found, the item will be made available as a scripting variable.

Another common usage scenario is to find the menu item for the root section. This is often done to highlight the front page in some special way:

```
<menu:item id="rootitem" sectionUniqueName="ece_frontpage"/>
```

When iterating over the menu, it is normal to highlight certain menu items in different ways. By using **<view:relationships>** it is possible to do simple calculations to highlight the current item, its siblings, its ancestors, children, and so on.

Syntax

```
<menu:item
   article="..."?
   id="..."
   link="..."?
   section="..."?
   sectionId="..."?
   sectionUniqueName="..."?/>
```

Attributes

id, mandatory, no runtime expressions
: Name of the scripting variable to export. The scripting variable will be one of the menu items in the enclosing **<menu:use>** tag.

section
: The section object to look for in the menu. If not specified, the current section will be used.

sectionUniqueName
: The unique-name of the section to look for in the menu.. If not specified, the current section will be used.

sectionId
: The ID of the section to look for in the menu. If not specified, the current section will be used.

article
: The ID of the article to look for in the menu. If not specified, the current article will be used, if there is a current article. If not, articles will be ignored..

link
: The URL of the link to look for in the menu. If not specified, links will be ignored.

6 Menu Resource Reference

Menu structures defined with menu editor included in the Menu Editor plug in are stored in a resource file called **menu**. This file is publication-specific and is stored in a publication's **META-INF/escenic/publication-resources/escenic/plugins** folder. The resource is an XML file, so you can, if you wish, hand edit it or use a different XML editor to edit it. It can be uploaded to the Content Engine in exactly the same way as other publication resources (see the **Escenic Content Engine Template Developer Guide** for information about uploading resources).

This section contains an example menu resource, plus descriptions of the elements and attributes that can appear in a menu resource. It is not guaranteed to be complete: for a complete, formally correct description of the file format, see the DTD included with the installation. You will find the DTD here:

plugins/menueditor/escenic/webapp/plugin/menuEditor/menu.dtd

6.1 Example Menu Resource

The menu resource is called **menu** and must be located in a publication **META-INF/escenic/publication-resources/escenic/plugins** folder. A publication has only one menu **resource**, but it may define any number of menus.

This example defines a menu called **main**. The menu contents are, in order:

- A link to the section "ece_frontpage", and nested links to subsections three levels deep.
- A link to the section with the **id** "19".
- An external link to the Vizrt home page, with the specified text and image.
- A spacer.
- A link to the content item with the **id** "11515".

```
<?xml version="1.0" encoding="utf-8"?>
<menuDef>
 <menu name="main">
   <section uniqueName="ece_frontpage" includeLevel="3">
   </section>
   <section id="19">
   </section>
   <link value="http://www.escenic.com">
     <text lang="en">A link to escenic
     </text>
     <image src="http://www.escenic.com/images/escenic.gif">
     </image>
   </link>
   <space>
   </space>
   <article articleId="11515">
   </article>
  </menu>
</menuDef>
```

The **menu** resource only defines a menu structure, it does not guarantee display of a menu. How the structure is used in the publication is determined by the publication templates.

6.2 Menu Resource Elements

This section lists all the elements that may appear in a **menu** resource.

6.2.1 menuDef

The root document tag, wraps one or more **menu** definitions.

Attributes
None

Context
Root

Contains
One or more **menu** tags.

6.2.2 menu

Specifies the structure of a single named menu, in terms of internal (section, article) and external (link) links, spaces and included menus.

Attributes
- **name** - The name of this menu. Required. The menu name must be unique within the publication.

Context
- **/menuDef**

Contains
- **article** - 0 or more
- **link** - 0 or more
- **section** - 0 or more
- **space** - 0 or more
- **includeMenu** - 0 or more

6.2.3 article

Specifies an internal link to a content item, specified by ID.

Attributes
- **articleId** - The ID of the linked content item. Required.

Context
- **/menuDef/menu**

Contains
None

6.2.4 link

An external link with an optional link text and/or image.

Attributes
- **`value`** - The URL of this link, for example **`http://www.uio.no`**. Required.
- **`target`** - The target of this link, example "_self" or "_blank" or "_parent" or "_top" or any named frame (HTML like).

Context
- **`/menuDef/menu`**

Contains
- **`text`** - 0 or more
- **`image`** - optional

6.2.5 section

An internal link to a section specified by unique name or ID.

Attributes
- **`id`** - The section ID. Required unless **`uniqueName`** is specified.
- **`uniqueName`** - The publication-unique name of the section. Required unless **`id`** is specified.
- **`includeLevel`** - If set to a number greater than 0, sub-sections of this section will be recursively included in the menu. This number specifies the maximum number of levels to include. Optional. By default no subsections are included.

Context
- **`/menuDef/menu`**

Contains
- **`text`** - 0 or more
- **`image`** - optional
- **`includeMenu`** - 0 or more
- **`section`** - 0 or more
- **`link`** - 0 or more
- **`space`** - 0 or more
- **`article`** - 0 or more

6.2.6 space

A spacer in the menu that may contain text and/or an image.

Attributes
none

Context

- **`/menuDef/menu`**

Contains

- **`text`** - 0 or more
- **`image`** - optional

6.2.7 includeMenu

Includes another menu in this menu. When specified, this menu item is replaced by the menu specified with the **`name`** attribute.

Attributes

- **`name`** - The name of the menu to include. Required.
- **`publication`** - The name of the publication containing the menu to include. Optional. By default, the menu specified with **`name`** is assumed to belong to the current publication.

Context

- **`/menuDef/menu`**

Contains

Emtpy

6.2.8 image

Specifies an image to be displayed in the menu.

Attributes

- **`src`** - The local path or URL of the image. Required.

Context

- **`section`**
- **`article`**
- **`space`**
- **`link`**

Contains

Empty

6.2.9 text

Specifies text that can be used as a menu item's label.

Attributes

- **`lang`** - The text language. Optional.

Context

- **`section`**
- **`article`**
- **`space`**

- **`link`**

Contains
Text

7 Class Reference

This chapter contains descriptions of the Java classes that are most likely to be used by JSP template developers. These are:

`MenuItem`
This is the superclass to which all menu items belong.

`SectionItem`, `ArticleItem`, `LinkItem` and `SpaceItem`.
These subclasses of **`MenuItem`** represent the different types of menu items.

Any object belonging to one of the subclasses also belongs to the **`MenuItem`** class, and has all of the **`MenuItem`** properties. It may also, however, depending on which of the subclasses it belongs to, have additional properties that are specific to its class.

7.1 MenuItem

A generic **menu item**. All nodes in a menu are menu items.

`MenuItem` has the properties described in the following sections.

7.1.1 URL

The URL of the menu item. If this **`MenuItem`** is an **`ArticleItem`**, then the URL of the referenced content item is used. If it is a **`SectionItem`**, then the URL of the referenced section is used, and so on.

Type: **`java.lang.String`**

Example usage

```
${myMenuItem.URL}
```

7.1.2 children

A list of all this **`MenuItem`**'s child **`MenuItem`**s. Each object in the collection will be a member of one of **`MenuItem`**s subclasses, just like this object. If this object has no children, then an empty list is returned.

Type: **`java.util.List`**

Example usage

```
${myMenuItem.children}
```

7.1.3 imageURL

The URL of the image associated with this menu item. If no image has been defined, then this method returns null.

Type: `java.lang.String`

Example usage

```
${myMenuItem.imageURL}
```

7.1.4 level

The level of this menu item. Root elements have a **`level`** of 1, their children have a **`level`** of 2, and so on. There can be several level 1 items in a menu. There are no level 0 menu items.

Type: `int`

Example usage

```
${myMenuItem.level}
```

7.1.5 parent

This menu item's parent menu item. If this is a level 1 menu item then **`null`** is returned.

Type: `com.escenic.menu.MenuItem`

Example usage

```
${myMenuItem.parent}
```

7.1.6 text

The text of the menu item. The default text depends on what type of menu item this is. The default text of a **`SectionItem`**, for example, will be the name of the referenced section. However, the default text may have been overridden by specifying a **`text`** element in the menu editor. If a menu item has been given several such **`text`** elements with different language attributes, then you can select the required language by passing in a language ID. The requested language is returned if possible, otherwise the default language text is returned.

Type: `java.lang.String`

Example usage

```
${myMenuItem.text}
```

7.1.7 type

The menu item type. Use this to identify the type (i.e subclass) of this **`MenuItem`** Refer to the description of **`com.escenic.menu.MenuItem`** constants in the JavaDoc to see which constants you may use.

Type: `int`

Example usage

```
${myMenuItem.type}
```

7.2 SectionItem

A menu item that represents a section. Note that the **`SectionItem`**'s children property may be recursively filled with further **`SectionItem`** elements representing the section's subsections. Whether or not this happens (and the depth of the recursion) is determined by the definition of the menu item in the **`menu`** resource.

`SectionItem` has the properties described in the following sections.

7.2.1 URL

The URL of the menu item. If this **`MenuItem`** is an **`ArticleItem`**, then the URL of the referenced content item is used. If it is a **`SectionItem`**, then the URL of the referenced section is used, and so on.

Type: **`java.lang.String`**

Example usage

```
${myMenuItem.URL}
```

7.2.2 children

A list of all this **`MenuItem`**'s child **`MenuItem`**s. Each object in the collection will be a member of one of **`MenuItem`**s subclasses, just like this object. If this object has no children, then an empty list is returned.

Type: **`java.util.List`**

Example usage

```
${myMenuItem.children}
```

7.2.3 imageURL

The URL of the image associated with this menu item. If no image has been defined, then this method returns null.

Type: **`java.lang.String`**

Example usage

```
${myMenuItem.imageURL}
```

7.2.4 level

The level of this menu item. Root elements have a **`level`** of 1, their children have a **`level`** of 2, and so on. There can be several level 1 items in a menu. There are no level 0 menu items.

Type: `int`

Example usage

```
${myMenuItem.level}
```

7.2.5 parent

This menu item's parent menu item. If this is a level 1 menu item then **`null`** is returned.

Type: **`com.escenic.menu.MenuItem`**

Example usage

```
${myMenuItem.parent}
```

7.2.6 section

The section referenced by this **`SectionItem`**. The section is accessed via the API. You should not use API methods to navigate through the section tree, as they are not optimized for this purpose: it is better to use the menu plugin and presentation layer.

Type: **`neo.xredsys.api.Section`**

Example usage

```
${mySectionItem.section}
```

7.2.7 sectionId

The ID of the section to be rendered. This ID can be used with a tag.

Type: `int`

Example usage

```
${mySectionItem.sectionId}
```

7.2.8 text

The text of the menu item. The default text depends on what type of menu item this is. The default text of a **`SectionItem`**, for example, will be the name of the referenced section. However, the default text may have been overridden by specifying a **`text`** element in the menu editor. If a menu item has been given several such **`text`** elements with different language attributes, then you can select the required language by passing in a language ID. The requested language is returned if possible, otherwise the default language text is returned.

Type: **`java.lang.String`**

Example usage

```
${myMenuItem.text}
```

7.2.9 type

The menu item type. Use this to identify the type (i.e subclass) of this **`MenuItem`** Refer to the description of **`com.escenic.menu.MenuItem`** constants in the JavaDoc to see which constants you may use.

Type: **`int`**

Example usage

```
${myMenuItem.type}
```

7.3 ArticleItem

A menu item that represents a content item.

`ArticleItem` has the properties described in the following sections.

7.3.1 URL

The URL of the menu item. If this **`MenuItem`** is an **`ArticleItem`**, then the URL of the referenced content item is used. If it is a **`SectionItem`**, then the URL of the referenced section is used, and so on.

Type: **`java.lang.String`**

Example usage

```
${myMenuItem.URL}
```

7.3.2 articleId

The ID of the content item referenced by this **`ArticleItem`**.

Type: **`int`**

Example usage

```
${myArticleItem.articleId}
```

7.3.3 children

A list of all this **`MenuItem`**'s child **`MenuItem`**s. Each object in the collection will be a member of one of **`MenuItem`**s subclasses, just like this object. If this object has no children, then an empty list is returned.

Type: **`java.util.List`**

Example usage

```
${myMenuItem.children}
```

7.3.4 imageURL

The URL of the image associated with this menu item. If no image has been defined, then this method returns null.

Type: `java.lang.String`

Example usage

```
${myMenuItem.imageURL}
```

7.3.5 level

The level of this menu item. Root elements have a **`level`** of 1, their children have a **`level`** of 2, and so on. There can be several level 1 items in a menu. There are no level 0 menu items.

Type: `int`

Example usage

```
${myMenuItem.level}
```

7.3.6 parent

This menu item's parent menu item. If this is a level 1 menu item then **`null`** is returned.

Type: `com.escenic.menu.MenuItem`

Example usage

```
${myMenuItem.parent}
```

7.3.7 publicationId

The publication ID of the content item referenced by this **`ArticleItem`**. For a cross-published content item, this is the ID of the content item's owning publication, not the ID of this publication.

Type: `int`

Example usage

```
${myArticleItem.publicationId}
```

7.3.8 text

The text of the menu item. The default text depends on what type of menu item this is. The default text of a **`SectionItem`**, for example, will be the name of the referenced section. However, the default text may have been overridden by specifying a **`text`** element in the menu editor. If a menu item has been given several such **`text`** elements with different language attributes, then you can select the required language by passing in a language ID. The

requested language is returned if possible, otherwise the default language text is returned.

Type: `java.lang.String`

Example usage

```
${myMenuItem.text}
```

7.3.9 type

The menu item type. Use this to identify the type (i.e subclass) of this **`MenuItem`** Refer to the description of **`com.escenic.menu.MenuItem`** constants in the JavaDoc to see which constants you may use.

Type: `int`

Example usage

```
${myMenuItem.type}
```

7.4 LinkItem

Links have no additional properties other than the one found in MenuItem.

`LinkItem` has the properties described in the following sections.

7.4.1 URL

The URL of the menu item. If this **`MenuItem`** is an **`ArticleItem`**, then the URL of the referenced content item is used. If it is a **`SectionItem`**, then the URL of the referenced section is used, and so on.

Type: `java.lang.String`

Example usage

```
${myMenuItem.URL}
```

7.4.2 children

A list of all this **`MenuItem`**'s child **`MenuItem`**s. Each object in the collection will be a member of one of **`MenuItem`**s subclasses, just like this object. If this object has no children, then an empty list is returned.

Type: `java.util.List`

Example usage

```
${myMenuItem.children}
```

7.4.3 imageURL

The URL of the image associated with this menu item. If no image has been defined, then this method returns null.

Type: `java.lang.String`

Example usage

```
${myMenuItem.imageURL}
```

7.4.4 level

The level of this menu item. Root elements have a `level` of 1, their children have a `level` of 2, and so on. There can be several level 1 items in a menu. There are no level 0 menu items.

Type: `int`

Example usage

```
${myMenuItem.level}
```

7.4.5 parent

This menu item's parent menu item. If this is a level 1 menu item then `null` is returned.

Type: `com.escenic.menu.MenuItem`

Example usage

```
${myMenuItem.parent}
```

7.4.6 text

The text of the menu item. The default text depends on what type of menu item this is. The default text of a `SectionItem`, for example, will be the name of the referenced section. However, the default text may have been overridden by specifying a `text` element in the menu editor. If a menu item has been given several such `text` elements with different language attributes, then you can select the required language by passing in a language ID. The requested language is returned if possible, otherwise the default language text is returned.

Type: `java.lang.String`

Example usage

```
${myMenuItem.text}
```

7.4.7 type

The menu item type. Use this to identify the type (i.e subclass) of this **`MenuItem`** Refer to the description of **`com.escenic.menu.MenuItem`** constants in the JavaDoc to see which constants you may use.

Type: **`int`**

Example usage

```
${myMenuItem.type}
```

7.5 SpaceItem

Space items are simply blank portions of a menu, which don't link anywhere. Space items can have text, or images associated, but don't have URLs. There are no special additional properties for Space items.

`SpaceItem` has the properties described in the following sections.

7.5.1 URL

The URL of the menu item. If this **`MenuItem`** is an **`ArticleItem`**, then the URL of the referenced content item is used. If it is a **`SectionItem`**, then the URL of the referenced section is used, and so on.

Type: **`java.lang.String`**

Example usage

```
${myMenuItem.URL}
```

7.5.2 children

A list of all this **`MenuItem`**'s child **`MenuItem`**s. Each object in the collection will be a member of one of **`MenuItem`**s subclasses, just like this object. If this object has no children, then an empty list is returned.

Type: **`java.util.List`**

Example usage

```
${myMenuItem.children}
```

7.5.3 imageURL

The URL of the image associated with this menu item. If no image has been defined, then this method returns null.

Type: **`java.lang.String`**

Example usage

```
${myMenuItem.imageURL}
```

7.5.4 level

The level of this menu item. Root elements have a **`level`** of 1, their children have a **`level`** of 2, and so on. There can be several level 1 items in a menu. There are no level 0 menu items.

Type: **`int`**

Example usage

```
${myMenuItem.level}
```

7.5.5 parent

This menu item's parent menu item. If this is a level 1 menu item then **`null`** is returned.

Type: **`com.escenic.menu.MenuItem`**

Example usage

```
${myMenuItem.parent}
```

7.5.6 text

The text of the menu item. The default text depends on what type of menu item this is. The default text of a **`SectionItem`**, for example, will be the name of the referenced section. However, the default text may have been overridden by specifying a **`text`** element in the menu editor. If a menu item has been given several such **`text`** elements with different language attributes, then you can select the required language by passing in a language ID. The requested language is returned if possible, otherwise the default language text is returned.

Type: **`java.lang.String`**

Example usage

```
${myMenuItem.text}
```

7.5.7 type

The menu item type. Use this to identify the type (i.e subclass) of this **`MenuItem`** Refer to the description of **`com.escenic.menu.MenuItem`** constants in the JavaDoc to see which constants you may use.

Type: **`int`**

Example usage

```
${myMenuItem.type}
```

8 Troubleshooting

This chapter contains advice on how to deal with various problems that can arise when using the Menu Editor plug-in.

Menu changes not visible in publication

In a multi-server environment, a failure in the passing of event messages between servers can result in changes to a menu not being reflected in the actual publication. Should this happen, you can fix the problem by forcing a manual reload of the affected publication's menu. To do this, select **Component Browser** in the Escenic admin web interface, follow the links to **`com/escenic/menu/MenuManager/revalidate`** and select **Invoke**.