Menu Editor

Plug-in Guide

3.6.0-1

escenic
A CCI COMPANY

Table of Contents

1 Introduction 5

2 Installation 6

- 2.1 Conventions 7
- 2.2 Package-based Installation 7
- 2.3 Old-style Installation 8
- 2.4 Re-assembling Applications 8
- 2.5 Verifying The Installation 9
- 2.6 Upgrading 9
  - 2.6.1 Package-based Upgrade 9
  - 2.6.2 Old-style Upgrade 10
- 2.7 CUE Plug-in Activation 11

3 Using the CUE Menu Editor 12

- 3.1 Editing Menus 12
  - 3.1.1 Working with Menu Items 13
  - 3.1.2 Working with Links and Spacers 13

4 Template Development 15

- 4.1 A Simple Example 15
- 4.2 A More Complete Example 16
- 4.3 Using The View Tag Library 17
- 4.4 Accessing Menu Items Directly 19

5 Taglib Reference 20

- 5.1 menu Tag Library 20
  - 5.1.1 menu:iterate 20
  - 5.1.2 menu:get 20
  - 5.1.3 menu:sectionView 22
  - 5.1.4 menu:spaceView 22
  - 5.1.5 menu:urlView 23
  - 5.1.6 menu:articleView 23
  - 5.1.7 menu:use 23
  - 5.1.8 menu:item 24

6 Menu Resource Reference 26

- 6.1 Example Menu Resource 26
- 6.2 Menu Resource Elements 27
  - 6.2.1 menuDef 27

6.2.2 menu 27
6.2.3 article 27
6.2.4 link 27
6.2.5 section 28
6.2.6 space 28
6.2.7 includeMenu 29
6.2.8 image 29
6.2.9 text 29
7 Class Reference 30
7.1 MenuItem 30
7.1.1 URL 30
7.1.2 children 30
7.1.3 imageURL 30
7.1.4 level 31
7.1.5 parent 31
7.1.6 text 31
7.1.7 type 31
7.2 SectionItem 32
7.2.1 URL 32
7.2.2 children 32
7.2.3 imageURL 32
7.2.4 level 32
7.2.5 parent 33
7.2.6 section 33
7.2.7 sectionId 33
7.2.8 text 33
7.2.9 type 33
7.3 ArticleItem 34
7.3.1 URL 34
7.3.2 articleId 34
7.3.3 children 34
7.3.4 imageURL 34
7.3.5 level 35
7.3.6 parent 35
7.3.7 publicationId 35
7.3.8 text 35
7.3.9 type 35

7.4 LinkItem..36
7.4.1 URL..36
7.4.2 children..36
7.4.3 imageURL..36
7.4.4 level..36
7.4.5 parent..37
7.4.6 text..37
7.4.7 type..37
7.5 SpaceItem..37
7.5.1 URL..37
7.5.2 children..38
7.5.3 imageURL..38
7.5.4 level..38
7.5.5 parent..38
7.5.6 text..38
7.5.7 type..39
8 Troubleshooting..40

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 set of Java objects for representing menus.
- A tag library that provides template developers with an easy way of accessing the menu beans.
- A CUE plugin allowing menu structures to be edited in the CUE editor.

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.

Menus are defined in special Content Engine publication resources called **menu resources**. Like most of the standard Content Engine publication resources described in the Escenic Content Engine Resource Reference, the menu resource is an XML file. Once you have created a menu resource for a publication you can upload it to the Content Engine in the same way as you upload other resources, using the **`escenic-admin`** web application (for details, see http://docs.escenic.com/ece-server-admin-guide//upload_resources1.html)

The menu resource file can be used to define 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.

For a detailed description of the menu resource format, see chapter 6.

> Prior to version 3.0, the Menu Editor plug-in included an editor, a small Java applet that ran inside Web Studio. This very old editor has now been withdrawn due to Java/browser compatibility problems and has been replaced by a menu editor plug-in for CUE in version 3.1. The menu resource format is in fact very simple, so if you don't use CUE you can edit menu resources directly in a text editor.

2 Installation

The following preconditions must be met before you can install Menu Editor 3.6.0-1:

- A suitable version of the Escenic Content Engine and Escenic assembly tool have been installed as described in the **Escenic Content Engine Installation Guide** and are in working order.
- You have the credentials needed to access Escenic's SW repositories.

In a multi-host environment:

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

Choosing an installation method

Until recently, all Escenic-supplied components had to be installed by manually downloading and unpacking archive files. This is no longer the case. The Content Engine itself, the assembly tool, the ece scripts and all plug-ins are now available as `deb` packages (for installing on Ubuntu/Debian systems) and as `rpm` packages (for installation on RedHat/CentOS systems). This is now the recommended method of installing Escenic systems, although it is still possible to use the old method based on manually downloading and unpacking archives. Note, however, that when you are installing a plug-in on an existing Content Engine installation, you should use the same installation method as you used for the Content Engine itself.

Using the new package installation method offers many advantages over the old manual installation procedure:

- It is faster and quicker
- It is significantly less error-prone
- It supports a fully automated upgrade process in which not only the installed packages themselves are upgraded, but also deployed EAR files. An upgrade script automatically checks the EAR files for copies of JAR files that have been updated, and replaces them with the new versions.

The old installation method will continue to be documented for the moment. Components installed using the new method are installed in `/usr/share/escenic`, whereas the old method recommends installation in `/opt/escenic`, and some other path components are slightly different. To cope with these differences, the following placeholders are used in some paths:

Placeholder	NEW METHOD path	OLD METHOD path
engine-installation	`/usr/share/escenic/escenic-content-engine-`*engine-version*	`/opt/escenic/engine`
assemblytool_installation	`/usr/share/escenic/escenic-assemblytool`	`/opt/escenic/assemblytool`

When installing the Menu Editor plug-in, you should use the same method as you used to install the Escenic Content Engine.

> Note that if you are planning to install the Escenic Content Engine using the new package-based method, you can install the Menu Editor plug-in simultaneously by simply including the Menu Editor plug-in package name (`escenic-menu-editor`) in the `apt-get` installation command. You only need to follow this procedure if you have already installed the Content Engine and now need to install the Menu Editor plug-in.

If you use CUE as your Escenic editor, then in addition to installing the Escenic Menu Editor plug-in, you must also activate CUE's Menu Editor plugin. For details, see section 2.7.

2.1 Conventions

The instructions in the following section assume that you have a standard Content Engine installation, as described in the Escenic Content Engine Installation Guide. *escenic-home* is used to refer to the `/opt/escenic` folder under which both the Content Engine itself and all plug-ins are installed.

The Content Engine and the software it depends on may be installed on one or several host machines depending on the type of installation required. In order to unambiguously identify the machines on which various installation actions must be carried out, the Escenic Content Engine Installation Guide defines a set of special host names that are used throughout the manual.

Some of these names are also used here:

assembly-host
: The machine used to assemble the various Content Engine components into an enterprise archive or .EAR file.

engine-host
: The machine(s) used to host application servers and Content Engine instances.

editorial-host
: **engine-host**(s) that are used solely for (internal) editorial purposes.

The host names always appear in a bold typeface. If you are installing everything on one host you can, of course, ignore them: you can just do everything on the same machine. If you are creating a larger multi-host installation, then they should help ensure that you do things in the right places.

2.2 Package-based Installation

To install Menu Editor on an Ubuntu or other Debian-based system, do the following on your **assembly-host** and on each of your **engine-hosts**:

1. Log in as `root`.
2. If necessary, add the Escenic `apt` repository to your list of sources:

   ```
   # echo "deb http://user:password@apt.escenic.com stable main non-free" >> /etc/
   apt/sources.list.d/escenic.list
   ```

   where *user* and *password* are your Escenic download credentials (the same ones you use to access the Escenic Maven repository). If you do not have any download credentials, please contact Escenic support.
3. Enter the following commands:

```
# apt-get update
# apt-get install escenic-menu-editor
```

On RedHat / CentOS systems, enter the following command as **`root`** on your **assembly-host** and each of your **engine-hosts**:

```
# rpm -Uvh https://user:password:yum.escenic.com/rpm/escenic-menu-
editor-3.6.0-1.x86_64.rpm
```

2.3 Old-style Installation

Installing the Menu Editor plug-in involves the following steps:

1. Log in as **`escenic`** on your **assembly-host**.
2. Download the Menu Editor distribution from the Escenic software repository. If you have a multi-host installation with shared folders as described in the **Escenic Content Engine Installation Guide**, then it is a good idea to download the distribution to your shared **`/mnt/download`** folder:

   ```
   $ cd /mnt/download
   $ wget https://user:password@maven.escenic.com/com/escenic/plugins/menu-editor/
   menu-editor/3.6.0-1/menu-editor-3.6.0-1.zip
   ```

   Otherwise, download it to some temporary location of your choice.
3. If the folder *engine-installation*`/plugins` does not already exist, create it:

   ```
   $ mkdir engine-installation/plugins
   ```
4. Unpack the downloaded distribution file:

   ```
   $ cd engine-installation/plugins
   $ unzip /mnt/download/menu-editor-3.6.0-1.zip
   ```

2.4 Re-assembling Applications

After installation, you **may** need to reassemble your web applications. You need you reassemble in the following cases:

- You installed the Menu Editor plugin-in using the old-style installation method.
- You installed the Menu Editor plugin-in using the package-based installation method **and** your installation includes any old-style JSP-based Escenic publications.

If you installed the Menu Editor plugin-in using the package-based installation method and all your publications are based on DPRES, the new decoupled presentation layer then you do not need to reassemble any applications.

1. Log in as **`escenic`** on your **assembly-host**.
2. Run the **`ece`** script to re-assemble your Content Engine applications

   ```
   $ ece assemble
   ```

This generates an EAR file (`/var/cache/escenic/engine.ear`) that you can deploy on all your **engine-host**s.

3. If you have a single-host installation, then skip this step.

   On each **engine-host**, copy `/var/cache/escenic/engine.ear` from the **assembly-host**. If you have installed an SSH server on the `assembly-host` and SSH clients on your `engine-host`s, then you can do this as follows:

   ```
   $ scp escenic@assembly-host-ip-address:/var/cache/escenic/engine.ear /tmp/
   ```

   where *assembly-host-ip-address* is the host name or IP address of your **assembly-host**.

4. On each **engine-host**, deploy the EAR file and restart the Content Engine by entering:

   ```
   $ ece stop deploy start --file /tmp/engine.ear
   $ ece restart
   ```

2.5 Verifying The Installation

To verify the status of the Menu Editor plug-in, open the Escenic Admin web application (usually located at `http://`*server*`/escenic-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.

2.6 Upgrading

How you upgrade the Menu Editor plug-in depends upon how you installed it in the first place. If you installed it using the new package-based method as described in section 2.2, then upgrading is very easy. If you installed it using the old method described in section 2.3, then it requires a bit more work.

2.6.1 Package-based Upgrade

You can only use this upgrade method if you have previously installed the Menu Editor plug-in as described in section 2.2. Otherwise you need to follow the method described in section 2.6.2.

All you need to do to upgrade your Menu Editor plug-in is:

1. Read the release notes for your planned upgrade. Make a note of any special tasks that need to be carried out in connection with the upgrades.
2. Log in as `root` on your **assembly-host** and on each of your **engine-hosts**, and enter the following commands:

   ```
   # apt-get update
   # apt-get upgrade
   ```

3. Carry out any required upgrade tasks.

2.6.2 Old-style Upgrade

You should only use this upgrade method if you have previously installed the Menu Editor plug-in as described in section 2.3. Otherwise you need to follow the method described in section 2.6.1.

To upgrade the Menu Editor plug-in to version 3.6.0-1:

1. Read the release notes for your planned upgrade. Make a note of any special tasks that need to be carried out in connection with the upgrades.
2. Log in as **`escenic`** on your **assembly host**.
3. Download the distribution file to a temporary location by entering:

   ```
   $ cd /tmp/
   $ wget http://user:password@maven.escenic.com/com/escenic/menu-editor/menu-
   editor/3.6.0-1/menu-editor-3.6.0-1.zip
   ```

   where *user* and *password* are the user name and password you have received from Escenic.
4. Remove the old version of the Menu Editor plug-in from your Content Engine **`plugins`** folder.

   ```
   $ mv /opt/escenic/engine/plugins/menu-editor/ /tmp/
   ```
5. Unpack the downloaded installation package into the **`plugins`** folder:

   ```
   $ cd /opt/escenic/engine/plugins/
   $ unzip /tmp/menu-editor-3.6.0-1.zip
   ```
6. Re-assemble the Content Engine by running the **`ece`** script:

   ```
   $ ece assemble
   ```
7. **If you are installing everything on one host, then skip this step.**

   Log in as escenic on each of your **engine-hosts** and copy **`/var/cache/escenic/engine.ear`** from the **assembly-host**. If you have installed an SSH server on the assembly-host and SSH clients on your **engine-hosts**, then you can do this as follows:

   ```
   $ ssh engine-host-ip-address
   $ scp -r escenic@assembly-host-ip-address:/var/cache/escenic/engine.ear /var/
   cache/escenic/
   ```

   where:

   engine-host-ip-address
   is the host name or IP address of one of your engine-hosts.

   assembly-host-ip-address
   is the host name or IP address of your assembly-host.
8. On each **engine-host**, deploy the EAR file by entering:

   ```
   $ ece deploy
   ```

   and then restart the Content Engine by entering:

   ```
   $ ece restart
   ```
9. Carry out any required upgrade tasks.

2.7 CUE Plug-in Activation

In order to be able to make use of Menu Editor functionality in CUE you need to activate CUE's Menu Editor plug-in. The plug-in is installed together with CUE itself, it just needs to be activated. To do this, you need to:

1. Log in as **`root`** on the host on which your CUE editor is installed.
2. Open **`/etc/escenic/cue-web-`***`x.y.z`***`/public/MenuEditor.yml`** for editing.
3. Remove the **`#`** comment character from all lines in the file.
4. Save your changes.
5. Enter the following command to reconfigure CUE with the new settings:

```
dpkg-reconfigure cue-web-x.y
```

3 Using the CUE Menu Editor

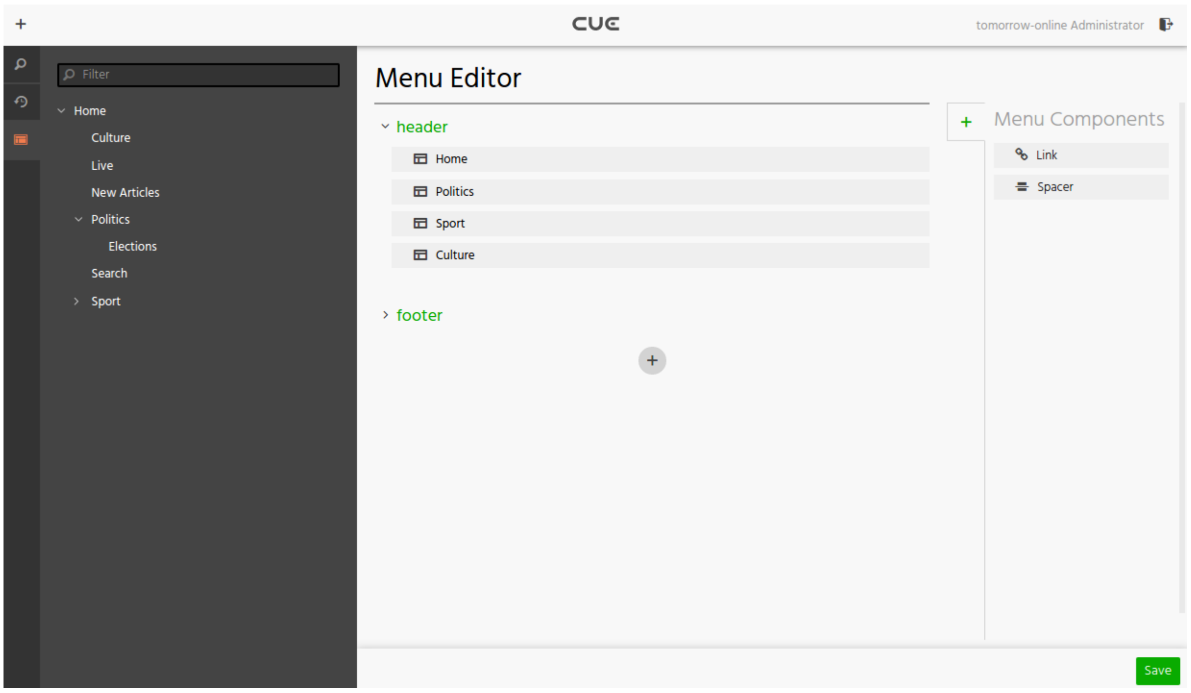
The Menu Editor plug-in includes a menu editing extension for the CUE editor.

> The old Web Studio-based menu editor has been withdrawn. The new CUE-based menu editor does not support the **`includeMenu`** element at present. If you do not yet use the CUE editor at your site or if your menus make use of the **`includeMenu`** element to include elements from other publications then you should use a standard text editor to edit your menu resource files. For a detailed description of the menu resource format, see chapter 6.

To start the CUE menu editor:

1. Go to the CUE home page.
2. Open the **Settings** panel by selecting the button.
3. Select **Menu editor** from the list of plug-ins by double-clicking.

The menu editor is opened in a new browser tab. If your publication already has menus, they will be displayed in the editor:



The menus are displayed in green and can be expanded and collapsed.

3.1 Editing Menus

To add a new menu, select the **+** button at the bottom of the menu list, and then enter a name for the menu. To change the name of a menu, just select it and edit the name. To delete a menu, right click/ long press it and select **Delete** from the displayed context menu.

A menu is a container for **menu items**, **links** and **spacers**. The following sections describe how to work with these elements. When you are finished modifying the menus, select **Save** to save your changes.

3.1.1 Working with Menu Items

To add a menu item to a menu:

1. Open one of the left hand panes.
2. Find the section or content item you want to add to the menu.
3. Drag it over the menu.
4. Drop it in the required location.

To move a menu item, just drag it to a new location. You can create tree structures by dropping menu items on top of one another: dropping one item on top of another creates a parent-child relationship between them. You can create trees of any depth in this way.

To edit a menu item, select it and then edit the attributes displayed in the metadata panel on the right. The attributes you can edit are:

Language
The language of the label.

Label
The label of the menu item. By default this is the title of the section / content item the menu item represents.

Number of sub levels
For sections only, the number of sub levels to include. If this attribute is set to a value greater than zero, then a submenu is automatically generated from the the section's subtree, to the specified depth. This automatically generated tree is not visible in the editor and cannot be edited, but is made available to the presentation layer.

Image URL
The URL of an image that can be used to represent the menu item.

A menu item can have multiple **Language** and **Label** attributes: to create additional labels, select the **+** button on the right. The **Language** attribute of the first **Language**/**Label** pair is always blank and cannot be edited: this is the default label.

To delete a menu item, right click/long press it and select **Delete** from the displayed context menu.

3.1.2 Working with Links and Spacers

To add a link or spacer to a menu:

1. If necessary, open the metadata panel on the right.
2. Pick the **Link** or **Spacer** you want to add to the menu.
3. Drag it over the menu.
4. Drop it in the required location.

To edit a link or spacer, select it and then edit the attributes displayed in the metadata panel on the right. The attributes you can edit for a link are:

Language
The language of the label.

Label
The label of the link.

URL
The URL of the link, for example `http://escenic.com`.

Target
The target of the link, for example `_self` or `_blank` or `_parent` or `_top`.

Image URL
The URL of an image that can be used to represent the link item.

A spacer only has **Language**, **Label** and **Image URL** attributes.

Both links and spacers can have multiple **Language** and **Label** attributes: to create additional labels, select the **+** button on the right. The **Language** attribute of the first **Language**/**Label** pair is always blank and cannot be edited: this is the default label.

To delete a link or spacer, right click/long press it and select **Delete** from the displayed context menu.

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`** resource. 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 may explicitly exclude the creation of menu items for subsections. Conversely, article menu items can have children if they are explicitly defined in the **`menu`** resource.
- 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

Menus are defined in an XML resource file called **`menu`**. This file is publication-specific and is stored in a publication's **`META-INF/escenic/publication-resources/escenic/plugins`** folder. It can be uploaded to the Content Engine in exactly the same way as other publication resources (see Upload 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 Escenic 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 resource. 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 resource. 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 resource. 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 resource. 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 resource. 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**.