Escenic Tag Library

Reference

5.3.10.154952


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

14.07.2014


vizrt

Table of Contents

1 Introduction

This manual contains reference information about the Escenic tag libraries. These tag libraries were the primary mechanism for accessing the JavaBeans created by the Escenic Content Engine in earlier versions of the Content Engine. This is, however, no longer the case. You are now recommended to use standard JSP/JSTL tags and the JSP expression language to access the Escenic beans in most cases. For a description of this technique and some simple examples, see the **Escenic Content Engine Template Developer Guide**.

The tag libraries are, however, still available for two purposes:

- In order to provide support for existing applications that rely on the tag libraries.
- To provide specialized functionality that cannot easily be realized in other ways.

You are in general recommended to avoid use of the tag libraries in new applications, and only use them where necessary. All the tags that you should avoid using in new applications are marked as **deprecated**. If you are maintaining an existing application or for some other reason really need to use one of these tags, then you should look up the description of it in an earlier version of this manual.

1.1 Taglibs

A **tag library** or **taglib** is a JSP standard for customized tag extensions. Like HTML, JSP has a set of standard tags. Unlike HTML, JSP allows the creation of new, customized tags. A set of non-standard tags is called a tag library.

1.2 Escenic Taglibs

The Escenic taglibs are:

- **`template`** - Operates on jsp template files
- **`publication`** - Operates on publications
- **`article`** - Operates on content items
- **`section`** - Operates on sections
- **`util`** - general utilities
- **`collection`** - Java Collection utilities
- **`view`** - Allows navigation in and display of trees such as the section structure
- **`relation`** - Operates on related content items

1.3 Conventions

Tag attributes in the Escenic tag libraries follow standard tag library conventions:

- All attributes are optional unless explicitly marked as required
- All attributes can be assigned a value using a scriptlet unless noted otherwise (runtime expression false)

1.4 Common attributes

Some tag attributes are used consistently across many tags:

Attribute	Description
`id`	The **`id`** attribute is not always mandatory. If **`id`** is set (mandatory or not), it is used to name the scripting variable created by the tag, as well as the key value used to locate the bean in page scope. If **`id`** is not set the tag is replaced by its value.
name	The key value used to look up an existing bean in any scope.
property	Identifies a property of the bean identified by the **`name`** attribute which is to be used as an input parameter by the tag. If not specified, the bean identified by the **`name`** attribute is used as an input parameter.

1.5 How to use JavaBeans with Escenic tags

JavaBeans are very important when using the Escenic tags. The JavaBeans the tags are working with are primary Escenic objects. To get a better picture of the Escenic JavaBeans see the **`Bean Reference`**. These beans can be in any scope: **`page`**, **`request`**, **`session`** or **`application`**. But the primary scope we use is **`page`**.

In the Escenic tag libraries there are two ways to get the properties from JavaBeans. To use **`name/property`** attributes in our tags, or use the **`<util:valueof param="...">`** tag. See **section 9.25** for further information.

In the following examples we will only show one of the possible methods to get the attributes.

1.5.1 Accessing Simple Properties

Lets us first take a look at a simple property reference within a tag:

```
property="url"
```

This is converted to a method call on the corresponding bean. Based on standard JavaBeans naming conventions, our example would do a call to the method **`getUrl()`**.

1.5.2 Accessing Nested Properties

The nested references are used to access a property through a hierarchy of property names separated by dots (.). We can take a look at a typical example for a Escenic template:

```
property="homeSection.name"
```

This will be translated into this Java expression:

```
getHomeSection().getName()
```

1.5.3 Accessing Indexed Properties

Subscripts can be used to access individual elements of a property whose value is an array, or a List. For example, when index getter are present, you may experience something like:

```
property="sections[1]"
```

This will be translated to:

```
getSection(1)
```

1.5.4 Accessing Mapped Properties

Works pretty much as indexed properties. By sending the key of an element in the map, you obtain the desired property. Consider the following example:

```
property="parameter(color)"
```

This reference will be translated into the Java expression:

```
getProperty("color")
```

2 article Tag Library

This library contains tags that operate on **PresentationArticle** beans. Most of the tags are simple query tags that return various bean properties, and are therefore deprecated. The tags that are not deprecated are:

- **define**
- **expiresCache**
- **fieldDistribution**
- **list**
- **use**

All the tags in this library operate on the current context's default article (content item), except **use** and **list**. In these cases you can specify the article to operate on using one of the following attributes or attribute combinations:

- **article**
- **articleId**
- **name**
- **name** and **property**
- **source** and **sourceId**

In order to use any article tags in a JSP file you must include the following prefix declaration in the file:

```
<%@ taglib uri="http://www.escenic.com/taglib/escenic-article" prefix="article" %>
```

article is the prefix normally used for this tag library.

Common Attributes

To specify an article you must set one of the following attributes or attribute combinations:

- **article**
- **articleId**
- **name**
- **name** and **property**
- **source** and **sourceId**

If one of these attributes or attribute combinations is specified, then the current article is ignored.

article
: The supplied bean must either be a **neo.xredsys.api.Article** or a **neo.xredsys.presentation.PresentationArticle** bean.

  Using this attribute excludes the use of the attributes **articleId**, **name/property** and **source/sourceId**.

articleId
: The value supplied can be an **int**, **Integer** or **String**.

Using **articleId** excludes the use of the attributes **article**, **name/property** and **source/sourceId**.

name
The name (key) of a bean that is to be used to locate an article. There are two ways of using this bean:

Using this attribute excludes the use of the attributes **article**, **articleId** and **source/sourceId**.

property
The name of a bean property. This attribute is used together with the **name** attribute to locate an article. If name is specified but this attribute is not specified, then the bean identified by the **name** attribute will be used.

This attribute cannot be used without the **name** attribute.

source
The source of an article. This attribute must be used together with **sourceId**.

Using this attribute excludes the use of the attributes **article**, **articleId** and **name/property**.

sourceId
The source ID of an article. This attribute must be used together with **source**.

Using this attribute excludes the use of the attributes **article**, **articleId** and **name/property**.

2.1 article:author

This tag is deprecated. Replace with jstl.

2.2 article:authors

This tag is deprecated. Replace with jstl.

2.3 article:date

This tag is deprecated. Replace with jstl.

2.4 article:define

Creates a scripting variable and a page-scoped bean from the specified article. The tag is similar to **`article:use`**, but unlike **`article:use`** does not set the current article.

Syntax

```
<article:define
   article="..."?
   articleId="..."?
   id="..."
   name="..."?
   property="..."?
   source="..."?
   sourceId="..."?
   toScope="..."?/>
```

Attributes

`id`, mandatory, no runtime expressions
Name of the scripting-variable we make.

`toScope`
Identify the JSP scope within which the PresentationArticle-bean will be created. If not specified, then the PresentaionArticle-bean is created in page scope.

Allowed values are:

`article`
A scriptlet that returns the required article.

Using this attribute excludes the use of the attributes **`articleId`**, **`name/property`** and **`source/sourceId`**.

`articleId`
An article id identifying the article to be defined. The value supplied can be an **`int`**, **`Integer`** or **`String`**.

Using **`articleId`** excludes the use of the attributes **`article`**, **`name/property`** and **`source/sourceId`**.

`name`
The name (key) of a bean from which a new bean/scripting variable is to be created.

There are two ways of using this bean:

Using this attribute excludes the use of the attributes **`article`**, **`articleId`** and **`source/sourceId`**.

`property`
The name of a bean property. This attribute is used together with the **`name`** attribute to locate an article from which a new bean/scripting

variable is to be created. If **name** is specified but this attribute is not specified, then the bean identified by the **name** attribute will be used.

This attribute cannot be used without the **name** attribute.

source
The source of the article to define. This attribute must be used together with **sourceId**.

Using this attribute excludes the use of the attributes **article**, **articleId** and **name/property**.

sourceId
The source ID of the article to define. This attribute must be used together with **source**.

Using this attribute excludes the use of the attributes **article**, **articleId** and **name/property**.

2.5 article:expiresCache

Expires the JSP cache if the specified article has been updated. It locates the enclosing **util:cache** tag and expires that cache. If there is no enclosing **util:cache** then nothing is done.

If a **util:cache** block contains more than one **expiresCache** tag, then the cache will be expired if any one of the specified articles has been updated.

See also

util:cache (**section 9.1**)

Syntax

```
<article:expiresCache
   article="..."?
   articleId="..."?
   name="..."?
   property="..."?
   source="..."?
   sourceId="..."?/>
```

Attributes

article
The article used to expire to cache.

articleId
Article id to identify the article to expire the cache.

name
Name of an bean to specify article to expire the cache.

property
Property used to find the article to expire the cache.

source
Source of the article to expire to cache.

sourceId
SourceId used to identify the article to expire the cache.

2.6 article:field

This tag is deprecated. Replace with jstl.

2.7 article:fieldDistribution

Counts hits for each first letter.

Syntax

```
<article:fieldDistribution
   groupBy="..."?
   id="..."
   sectionId="..."?/>
```

Attributes

id, mandatory, no runtime expressions
Name of the Map returned.

groupBy
What to group the hit count on.

sectionId
Id of the section to be used.

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.util.Map**.

2.8 article:getType

This tag is deprecated. Replace with jstl.

2.9 article:hasRelation

This tag is deprecated. Replace with jstl.

2.10 article:hasNoRelation

This tag is deprecated. Replace with jstl.

2.11 article:hasValue

This tag is deprecated. Replace with jstl.

2.12 article:hasNoValue

This tag is deprecated. Replace with jstl.

2.13 article:homeSection

This tag is deprecated. Replace with jstl.

2.14 article:isType

This tag is deprecated. Replace with jstl.

2.15 article:lastChanged

This tag is deprecated. Use <article:date date="lastModified" > instead.

2.16 article:latestArticles

This tag is deprecated. Use **<article:list>** instead.

2.17 article:link

This tag is deprecated. We recommend that you ask the article-object about the url, and generate the html-tag by yourself. This is a more flexible solution. Example:

```
<a href="<util:valueof param="article.url"/> ">name</a>
```

2.18 article:list

This tag retrieves the latest **n** articles from a publication (using the Publications "all"-section), a section, a list of sections or a selection of sections. If no section is specified the current section will be used. If you specify an illegal attribute (for example, a **uniqueName** that does not exist) the **ece_all** section is used.

Only published articles belonging to published home sections are retrieved. The retrieved articles are returned in a **java.util.List**

Field indexing

The **expression** and **field** attributes can only be used on **indexed** fields. Field indexing must be specified on a per-field basis in the publication's **content-type** resource.

To switch on field indexing in a particular content type, you must add a parameter element that sets the parameter **neo.xredsys.service.article.attribute** to **true**.

```
<content-type name="news">
...
  <parameter name="neo.xredsys.service.article.attribute" value="true"/>
</content-type>
```

Once you have done this, you can specify indexing of individual fields within the content type by adding **neo.xredsys.service.article.attributeField** parameters to the field definitions:

```
<field name="priority" type="number">
  ...
  <parameter name="neo.xredsys.service.article.attributeField" value="priority"/>
</field>
```

Note that the **value** attribute is set to the name of the field that is to be indexed. The **neo.xredsys.service.article.attributeField** parameter actually determines the name that you will need to use to identify the indexed field in the **article:list** tag. You could set it to some other name, but you are advised not to do so.

The examples shown above will ensure that the **priority** field of **news** content items is indexed and can be used for selection purposes by **article:list**.

Age

Two properties in the **feature** publication resource affect the behavior of this tag:

article.list.age.max
: Specifies the maximum age (in hours) of content items that may be retrieved, thus potentially overriding the value you specify with the **from** attribute. If you specify a **from** value that equates to an age greater than **article.list.age.max**, then it will be ignored, and **article.list.age.max** will be used instead. The default is 720 hours (=30 days). To disable this limit, set **article.list.age.max** to **-1**.

`article.list.age.default`
The default age limit (in hours) that will be used if no **`from`** value is specified. The default is 720 hours (=30 days). To disable this default (so that there is no **`from`** limit if one is not explicitly specified), set **`article.list.age.max`** to **`-1`**.

Caching

By default, the result will be cached for 1 minute.

When using the **`to`** property, the result may not be cached and the tag should be wrapped in a **`<util:cache />`**, to avoid slowing down the system too much. Please see the documentation of the **`to`** property for more information regarding this.

Syntax

```
<article:list
   all="..."?
   excludeArticleTypes="..."?
   expression="..."?
   field="..."?
   from="..."?
   homeSectionOnly="..."?
   id="..."
   includeArticleTypes="..."?
   includeSubSections="..."?
   max="..."?
   name="..."?
   onlyLive="..."?
   property="..."?
   publicationId="..."?
   sectionId="..."?
   sectionUniqueName="..."?
   sort="..."?
   to="..."?
   view="..."?
   view="..."?/>
```

Attributes

`id`, mandatory, no runtime expressions
A name to identify the selected articles.

`from`
How far back to list articles. The time could either be defined as hours in the past or as an exact date. The format of the exact date is **`yyyy-MM-dd hh:mm`**. The exact date should always be in the past.

If not defined or set to **`-1`** the function is disabled.

`excludeArticleTypes`
We will exclude the article type set here. May be a list of comma separated article types.

If neither **`includeArticleTypes`** or **`excludeArticleTypes`** are set, all article types will be included in the list.

homeSectionOnly
If set to true we will only list articles which has sectionId as homeSection.

includeArticleTypes
We will include the article type set here. May be a list of comma separated article types.

If neither **includeArticleTypes** or **excludeArticleTypes** are set, all article types will be included in the list.

includeSubSections
will include articles from sub sections in the list if set to true. The default value is false.

max
How many articles to list

onlyLive
will only list articles that are alive. Default value is set to true. If set to false it will include all states possible for an article.

This attribute has been deprecated and should not be used.

publicationId
The publicationId. If **publicationId** is not set we will use the default publication.

sectionId
The id of the sections to get latest articles from. May be a list of comma separated ids.

sectionUniqueName
The name of the sections to get latest articles from. May be a list of comma separated names. It must be the sections uniquename!

name
Specifies the attribute name of the Section we are going to search for Articles.

The bean you specify must be of type Section, or you also have to specify property.

property
Specifies the name of the property to be accessed on the bean specified by the name attribute. You can not use this attribute without the name attribute.

This value may be a simple, indexed, or nested property reference expression. If not specified, the Section identified by name (we then assume the the bean specified is a Section) will be used in the search.

to
If set, the tag will return articles up to this point in time. The time could either be defined in hours in the past or as an exact date. The format of the exact date is **yyyy-MM-dd hh:mm**. The exact date should always be in the past.

If not defined, set to **-1** or an exact date is specified, the result will be cached. If set to hours, the result will not be cached.

view
a view of sections we will use in the search for the latest articles

sort
What we will sort the article list on. Valid values are: **publishDate**, **lastChangedDate**. It is possible to send none to disable the sort function.

To decide if the sorting shall be ascending or descending you simply add either +/- before the desired sort criteria.

E.g. **-publishDate** is sorting descending on the published date of the articles

---

If this attribute is used together with the **expression** and **field** attributes, you can not sort it by date. It is only possible to specify if the sort will be ascending or descending.

---

view
a view of sections we will use in the search for the latest articles

expression
Is used to get a limited search. This attribute is always used together with the field attribute.

E.g. if you specify the title in field attribute and **a*** in expression you will only get articles that has a title that start with a or A. If you want articles that start with the letters a, b or c you use **a-d**.

---

d-f will include all article with title that starts with d and e. It will not include f.

If the field you want to use the expression in is of type **enumeration** you must wrap the expression with the following: **<ecs_selection>expression</ecs_selection>**.

---

field
Is used when you want the articles sorted by the specified field. And it is used together with the **expression** attribute.

`all`
Attribute to override the default limit of max age on articles. If this is set to **`true`**, then all article will be included in the list.

This attribute should be used with caution. It may cause large search results and slow jsp pages. It should only be used when you limit the search with for instance **`articleType`** or a **`section`** that do not contain many articles.

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.util.List`**.

2.19 article:published

This tag is deprecated. Use <article:date date="published" /> instead.

2.20 article:relatedArticles

This tag is deprecated. Use **`<relation:articles>`** instead.

2.21 article:sections

This tag is deprecated. Replace with jstl.

2.22 article:use

Sets the **current article** for the body of this tag. All article tags that use the current article in the body of this tag will use the article specified here rather than the current article set outside this tag.

All the other tags in this library expect a current article to be set (although some allow you to optionally specify the article to operate on).

To specify the new current article you must set one of the following attributes or attribute combinations:

- **`article`**
- **`articleId`**
- **`name`**
- **`name`** and **`property`**
- **`source`** and **`sourceId`**

Syntax

```
<article:use
   article="..."?
   articleId="..."?
   name="..."?
   property="..."?
   source="..."?
   sourceId="..."?>
   ...
</article:use>
```

Attributes

`article`
The article to be set as current article in the body of this tag. The supplied bean must either be a **`neo.xredsys.api.Article`** or a **`neo.xredsys.presentation.PresentationArticle`** bean.

Using this attribute excludes the use of the attributes **`articleId`**, **`name/property`** and **`source/sourceId`**.

`articleId`
An article id identifying the article to be set as current article in the body of this tag. The value supplied can be an **`int`**, **`Integer`** or **`String`**.

Using **`articleId`** excludes the use of the attributes **`article`**, **`name/property`** and **`source/sourceId`**.

`name`
The name (key) of a bean that is to be used to locate an article that will be used as current article in the body of this tag.

There are two ways of using this bean:

Using this attribute excludes the use of the attributes **`article`**, **`articleId`** and **`source/sourceId`**.

`property`
The name of a bean property. This attribute is used together with the **`name`** attribute to locate an article that will be used as current article in the body of this tag. If name is specified but this attribute is not specified, then the bean identified by the **`name`** attribute will be used as current article.

This attribute cannot be used without the **`name`** attribute.

`source`
The source of an article to be used as current article in the body of this tag. This attribute must be used together with **`sourceId`**.

Using this attribute excludes the use of the attributes **`article`**, **`articleId`** and **`name/property`**.

`sourceId`
The source ID of an article to be used as current article in the body of this tag. This attribute must be used together with **`source`**.

Using this attribute excludes the use of the attributes **article**, **articleId** and **name/property**.

2.23 article:renderField

This tag makes it possible for template developers to access inline objects and specify style according to their types. The body of the tag will be executed once for each inline object in a particular field. The mime-type of the field must be 'application/xhtml+xml'. The inline object will be made available as an attribute in either page, request, session or application scope, as specified by the template developer. The default is page scope. The type of the attribute will be **neo.xredsys.presentation.PresentationElement**

Syntax

```
<article:renderField
   article="..."?
   articleId="..."?
   field="..."
   name="..."?
   output="..."?
   property="..."?
   source="..."?
   sourceId="..."?
   toScope="..."?
   var="..."
   varBody="..."?>
   ...
</article:renderField>
```

Attributes

article
The article to be set as current article in the body of this tag. The supplied bean must either be a **neo.xredsys.api.Article** or a **neo.xredsys.presentation.PresentationArticle** bean.

Using this attribute excludes the use of the attributes **articleId**, **name/property** and **source/sourceId**.

articleId
An article id identifying the article to be set as current article in the body of this tag. The value supplied can be an **int**, **Integer** or **String**.

Using **articleId** excludes the use of the attributes **article**, **name/property** and **source/sourceId**.

If both **article** and **articleId** attributes are used together, then **articleId** takes precedence.

name
The name (key) of a bean that is to be used to locate an article that will be used as current article in the body of this tag.

There are two ways of using this bean:

Using this attribute excludes the use of the attributes **`article`**, **`articleId`** and **`source/sourceId`**.

`property`
The name of a bean property. This attribute is used together with the **`name`** attribute to locate an article that will be used as current article in the body of this tag. If name is specified but this attribute is not specified, then the bean identified by the **`name`** attribute will be used as current article.

This attribute cannot be used without the **`name`** attribute.

`source`
The source of an article to be used as current article in the body of this tag. This attribute must be used together with **`sourceId`**.

Using this attribute excludes the use of the attributes **`article`**, **`articleId`** and **`name/property`**.

`sourceId`
The source ID of an article to be used as current article in the body of this tag. This attribute must be used together with **`source`**.

Using this attribute excludes the use of the attributes **`article`**, **`articleId`** and **`name/property`**.

`var`, mandatory, no runtime expressions
The name of the attribute in which an inline item will be stored

`varBody`, no runtime expressions
The name of the attribute in which the body text of the inline item will be stored, if any.

`field`, mandatory
The name of the article field

`toScope`, no runtime expressions
The scope in which an inline item will be stored as an attribute

`output`
The value **`html`** or **`xml`**.

The value **`html`** causes the output of the markup in the field to be rendered as HTML compliant markup. For example, **`<br>`** tags will not be closed. The value **`xml`** causes the output to be rendered as XML. **`<br>`** tags will in this case be rendered with a closing tag: **`<br />`**.

The value **`html`** is the default.

3 collection Tag Library

This tag library contains tag that are usefull to create new Collections, or manipulate existing Collections.

When we are talking about Collections this is equivalent to the java interface **`java.util.Collection`**. Except that we have included **`java.util.Map`** into our tags.

The essential tags in this taglib are:

- add
- createList
- createMap
- createSet
- get
- pageByPage

```
<%@ taglib uri="http://www.escenic.com/taglib/escenic-collection" prefix="collection" %>
```

We usually use **`collection`** as prefix to this tag library.

3.1 collection:add

Adds a value to a Collection. When adding a value to a Map or List you must only use value/valueName/valueProperty. But you are adding to a Map you must use both value/valueName/valueProperty and key/keyName/keyProperty.

Syntax

```
<collection:add
   collection="..."
   key="..."?
   keyName="..."?
   keyProperty="..."?
   value="..."?
   valueName="..."?
   valueProperty="..."?/>
```

Attributes

`collection`, mandatory
The collection to add a object to. This can either be a List, Set or Map.

`value`
This will be the object to add to the specified Collection. It is not necessary to specify this attribute if valueName is used. They are mutual exclusively. If both are specified the value will be used.

`valueName`
Specifies the attribute name of the bean that we will use as value. If valueProperty also is provided we will get that property from the bean

specified here. This attribute is required unless you specify a value attribute.

valueProperty
Will get the property from the bean specified by the valueName attribute. To use this attribute valueName must also must be defined.

key
This will be the object to add to the specified Collection. It is not necessary to specify this attribute if valueName is used. They are mutual exclusively. If both are specified the value will be used.

keyName
Specifies the attribute name of the bean that we will use as key. If keyProperty also is provided we will get that property from the bean specified here. This attribute is required unless you specify a key attribute.

keyProperty
Will get the property from the bean specified by the keyName attribute. To use this attribute keyName must also must be defined.

3.2 collection:addAll

Adds a Collection to a existing Collection. When adding a Collection you can specify it with the attributes: **add** or **addName**/**addProperty**.

Syntax

```
<collection:addAll
   add="..."?
   addName="..."?
   addProperty="..."?
   collection="..."/>
```

Attributes

collection, mandatory
The Collection to add another Collection into. This can either be a **List**, **Set** or **Map**.

add

addName

addProperty

3.3 collection:contains

Will render body if Collection contains specified Object. Else it will skip to body of this tag.

Syntax

```
<collection:contains
   collection="..."
   name="..."?
   property="..."?
   value="..."?>
   ...
</collection:contains>
```

Attributes

`collection`, mandatory
The collection we will check if it contains the specified object.

`value`
This will be the object to check if it is added to the specified Collection. It is not necessary to specify this attribute if name is used. They are mutual exclusively. If both are specified the value will be used.

`name`
Specifies the attribute name of the bean that we check if it is added to the specified collection. If property also is provided we will get that property from the bean specified here. This attribute is required unless you specify a value attribute.

`property`
Will get the property from the bean specified by the name attribute. To use this attribute name must also must be defined.

3.4 collection:createList

Creates an instance of a **`java.util.List`** object. It will default create a **`java.util.ArrayList`**. If other types of Lists are desired, this must be specified in the type attribute.

The list will be defined as an attribute accessible to the remainder of the current page.

Syntax

```
<collection:createList
   id="..."
   toScope="..."?
   type="..."?/>
```

Attributes

`id`, mandatory, no runtime expressions
Specifies the name of the scripting variable (and associated page scope attribute) that will be made available with the value of the specified property.

`toScope`
Identify the JSP scope within which the List will be created. If not specified, then the bean is created in page scope.

Allowed values are:

type
The type of list to create. If not specified there will be created an **java.util.ArrayList**.

3.5 collection:createMap

Creates an instance of a **java.util.Map** object. It will default create a **java.util.HashMap**. If other types of Lists are desired, this must be specified in the type attribute.

The Map will be defined as an attribute accessible to the remainder of the current page.

Syntax

```
<collection:createMap
   id="..."
   toScope="..."?
   type="..."?/>
```

Attributes

id, mandatory, no runtime expressions
Specifies the name of the scripting variable (and associated page scope attribute) that will be made available with the value of the specified property.

toScope
Identify the JSP scope within which the List will be created. If not specified, then the bean is created in page scope.

Allowed values are:

type
Which type of map to create. If this is not specified a **java.util.HashMap** will be created.

3.6 collection:createSet

Creates an instance of a **java.util.Set** object. It will default create a **java.util.HashSet**. If other types of Lists are desired, this must be specified in the type attribute.

The Set will be defined as an attribute accessible to the remainder of the current page.

Syntax

```
<collection:createSet
   id="..."
   toScope="..."?
   type="..."?/>
```

Attributes

`id`, mandatory, no runtime expressions
Specifies the name of the scripting variable (and associated page scope attribute) that will be made available with the Set.

`toScope`
Identify the JSP scope within which the List will be created. If not specified, then the bean is created in page scope.

Allowed values are:

`type`
Which type of **`java.util.Set`** to create. If not specified a **`java.util.HashSet`** will be created.

3.7 collection:get

Will get an element from the **`Collection`** specified. This tag will only work if the Collection is either a **`List`** or a **`Map`**.

`Set` do not have a get method.

If your Collection is a **`Map`** the **`key`** and **`name/property`** attributes must be the key value of the wanted element. But if the Collection is a **`List`** the attributes must be the index of the wanted element.

This tag might return null, if the specified item does not exist or that null acctually are added in the Collection.

Syntax

```
<collection:get
   collection="..."
   id="..."?
   key="..."?
   name="..."?
   property="..."?
   type="..."?/>
```

Attributes

`collection`, mandatory
The collection to get element from. This must be either a **`List`** or a **`Map`**.

`Set` do not have a get method.

`id`, no runtime expressions
If the id is set the object will be made available as a scripting variable. If id is not specified the object fetched from the collection will be printed directly to the page.

`type`
Type of object that will be return from this tag. Either as a scripting variable or be written directly out. See the id-attribute.

key
Key of the object we will get from the **Map** or the index to get from the **List**.

name
Name to a bean to be used as key of the object we will get from the **Map** or the index to get from the **List**.

property
Specifies the name of the property to be used as key in the **Map** or the **List** (on the bean specified by name).

3.8 collection:isEmpty

Will render body only if the collection is empty.

Syntax

```
<collection:isEmpty
   name="..."?
   property="..."?>
   ...
</collection:isEmpty>
```

Attributes

name
Name of the collection to check if its empty.

property
Property to get the wanted collection from the bean specified by the name attribute.

3.9 collection:isNotEmpty

Will only render body if the collection contains 1 or more elements.

Syntax

```
<collection:isNotEmpty
   name="..."?
   property="..."?>
   ...
</collection:isNotEmpty>
```

Attributes

name
Name of the collection to check if it contains elements.

property
Property to get the wanted collection from the bean specified by the name attribute.

3.10 collection:pageByPage

Will make it possible to display the elements in a collection over several pages. You specify how many objects to be shown on each page, how many pages to be displayed in the navigator.

Two attributes will be made available: "page" and "nav".

page: which will be a **`java.util.List`** of all the elements to be displayed on this page.

nav: which will be a **`neo.taglib.collection.PagesList`** of all the pages to be displayed in the navigator.

Syntax

```
<collection:pageByPage
   name="..."?
   navSize="..."?
   pageNumber="..."?
   pageSize="..."?
   property="..."?/>
```

Attributes

`name`
Name of bean to create the list from.

`property`
Will get the property from the bean specified by the name attribute. To use this attribute name must also must be defined.

`navSize`
Size of the navigator.

`pageNumber`
Number of the page to be shown.

`pageSize`
Size of the pages to be shown. Number of element to show on each page.

3.11 collection:remove

Removes a object from the specified Collection.

If the collection is of type Map we will remove the key from the map. If the value specified is not equal to any keys in the Map, nothing will happened.

Syntax

```
<collection:remove
   collection="..."
   name="..."?
   property="..."?
   value="..."?/>
```

Attributes

`collection`, mandatory
The collection we are about to remove an object from.

`value`
Value to be removed from the specified Collection.

`name`
Name of an bean to be removed from the specified collection. If the property attribute also is set, we will get that property from the bean specified here.

`property`
Specifies the name of the property to be removed from the collection (on the bean specified by name).

3.12 collection:string

This is a usefull tag if you wan to convert a **`Collection`** eiter to or from a **`String`**. It can do both. This is decided by the **`method`** attribute.

If the operation is **`split`** you can choose the type of object to be returned. This is done by the **`type`** attribute.

Syntax

```
<collection:string
   delimiter="..."?
   id="..."?
   keyValueDelimiter="..."?
   method="..."?
   name="..."?
   property="..."?
   type="..."?
   value="..."?/>
```

Attributes

`id`
Name of the **`String`** or **`Collection`** that is returned from this tag. This is determined by the **`method`** attribute.

`value`
The object to convert. This must be of the correct type acording to current method.

`name`
The key value to look up an existing bean. There is two ways of using this bean.

Either the bean must be of type **`collection`** or **`String`**. This depends on which type of operation you intend to do.

Or the bean must have a property to get the wanted object. Then the attribute **`property`** must be used.

Using this attribute excludes the use of the **value** attribute.

property
Identifies the JavaBeans property (of the bean identified by the **name** attribute) to get the Object to be used by the tag. If not specified, the bean identified by the **name** attribute itself will be used as the value.

method
This attribute will define the usage of this tag.

Valid values:

If set to **split** it will take a **String**, split it and then add all items into the **Collection**. If it is set to **merge** the tag will take the specified **Collection** and merge all itemes into a **String**.

type
Will define what type of **Collection** the tag will return. You can either specify the class or just type either **collection** or **map**.

If you specify a class is must be an instance of **java.util.Collection** or **java.util.Map**

This attribute is only in use when **method** is set to **split**.

delimiter
What delimiter to be used to separate to elements when converting a **collection** to a **String**.

If not specified the default value will be used. The default value is ,

keyValueDelimiter
This delimiter will only be used when converting a **String** to a **Map**. It is used to separate the **key/value** pair.

If not specified will the default value be used. The default value is =

4 publication Tag Library

This tag library contains tags that are working with the **Publication** object.

Syntax to import the publication tag library:

```
<%@ taglib uri="http://www.escenic.com/taglib/escenic-publication" prefix="publication" %>
```

We usually use **publication** as prefix to this tag library.

4.1 publication:use

Sets the "current" publication to a different publication within this tag. Normally the default publication is set in the **index.jsp** and **article.jsp** files.

Almost all the other tags expect that it is within context of a publication.

You must use either the **id** or the **name** attribute to specify the publication.

Syntax

```
<publication:use
   publicationId="..."?
   publicationName="..."?>
   ...
</publication:use>
```

Attributes

publicationId
Id of the publication to be the current publication. You can not this together with publicationName.

publicationName
Name of the publication to be the current publication. You can not this together with publicationId.

5 profile Tag Library

All tags works with profiles in one way or another.

The essential tags in this taglib are:

- attribute
- present
- use

```
<%@ taglib uri="http://www.escenic.com/taglib/escenic-profile" prefix="profile" %>
```

We usually use **`profile`** as prefix to this tag library.

Common Attributes

To specify the PresentationProfile to be used, you must use one of the following attributes: **`profileId`** or **`name/property`**.

In most of the tags it is mandatory to specify either **`profileId`** or **`name/property`**.

`profileId`
The **`profileId`** attribute will get the PresentationProfile with that name.

Using **`profileId`** excludes the use of the attributes:**`name/property`**

`name`
The key value to look up an existing bean. There is two ways of using this bean.

Either the bean must be of type **`com.escenic.profile.presentation.PresentationProfile`**.

Or the bean must have a property to get the **`PresentationProfile`**. Then the attribute **`property`** must be used.

Using this attribute excludes the use of the attribute: **`profileId`**.

`property`
Identifies the JavaBeans property (of the bean identified by the **`name`** attribute) to get the PresentationProfile to be used by the tag. If not specified, the bean identified by the **`name`** attribute itself will be used as the value.

This attribute can not be used without the **`name`** attribute.

5.1 profile:define

Will create both an scripting variable and a page scoped bean from the specified profile. Therefore the tag way look a bit like **<profile:use>**, except that it does not set the default profile.

Syntax

```
<profile:define
   id="..."
   name="..."?
   profileId="..."?
   property="..."?
   toScope="..."?/>
```

Attributes

id, mandatory, no runtime expressions
Name of the scripting-variable we make.

toScope
Identify the JSP scope within which the Presentationprofile-bean will be created. If not specified, then the PresentaionProfile-bean is created in page scope.

Allowed values are:

name
Name of the bean to define.

property
Property to get the **PresentationProfile**.

profileId
Name of the profile to be used.

5.2 profile:exist

Checks if a profile, specified by the profileName attribute, exist. If the profile exist the body if this tag will be rendered.

Syntax

```
<profile:exist
   title="...">
   ...
</profile:exist>
```

Attributes

title, mandatory
Name of the profile to check.

5.3 profile:notExist

Checks if a profile, specified by the profileName attribute, exist. If the profile exist the body if this tag will be rendered.

Syntax

```
<profile:notExist
   title="...">
   ...
</profile:notExist>
```

Attributes

`title`, mandatory
Title of the profile to check.

5.4 profile:field

This tag is deprecated. Replace with jstl.

5.5 profile:hasValue

This tag is deprecated. Replace with jstl.

5.6 profile:hasNoValue

This tag is deprecated. Replace with jstl.

5.7 profile:present

Checks if the request contains a profile, and evalutates the nested body content of this tag only if a profile is present.

Syntax

```
<profile:present>
   ...
</profile:present>
```

5.8 profile:notPresent

Checks if the request contains a profile, and evalutates the nested body content of this tag only if a profile is not present.

Syntax

```
<profile:notPresent>
   ...
</profile:notPresent>
```

5.9 profile:use

Sets the **`current`** profile within this tag. All **`profile`** tags that relate to the current profile in the body of this tag will use the specified profile. This tag will make any tags within its body use this profile.

Syntax

```
<profile:use
   name="..."?
   profileId="..."?
   property="..."?
   title="...">
   ...
</profile:use>
```

Attributes

`title`, mandatory
Title of the profile to use.

`name`
Name of an bean to specify profile to be set as default profile.

`property`
Property used to find the profile to be set as default profile.

`profileId`
Name of the profile used to to be set as default profile.

6 relation Tag Library

Use the tags in this tag library to display relations to articles. All tags will iterate over the related objects.

The essential tags in this taglib are:

- articles
- images
- links
- media

```
<%@ taglib uri="http://www.escenic.com/taglib/escenic-relation" prefix="relation" %>
```

We usually use **`relation`** as prefix to this tag library.

6.1 relation:articles

This tag is deprecated. Replace with jstl.

6.2 relation:images

This tag is deprecated. Replace with jstl.

6.3 relation:persons

This tag is deprecated. Replace with jstl.

7 section Tag Library

This tag library contains tags that work on the current or specified section. The tags can get information from the section, expire the surrounding jsp-cache, set a new default section or get different **Views**. For more information about jsp-cache see **chapter 9**, about View see **chapter 10**.

The essential tags in this taglib are:

- use
- parameter

All tags are working with a section-object, except **<section:parameter>**. The default behavior is that they will use the current section. This can be overridden by the common attributes discussed below.

Syntax to import the section tag library:

```
<%@ taglib uri="http://www.escenic.com/taglib/escenic-section" prefix="section" %>
```

We usually use **section** as prefix to this tag library.

The following tags has been removed because they either did not work or was not finished: <section:hasAgreement>, <section:validateAgreement> and <section:checkVirtual>. They may be reintroduced later on.

Common Attributes

If you want to specify which section to work with instead of using the default section, you can do so by using one our common attributes. This is the possible attributes: **section**, **sectionId**, **uniquename** and **name/property**. They are mutually exclusive(except name/property).

When using **name/property** you must specify **name**, but **property** is not mandatory.

section
Must be a **neo.xredsys.api.Section** object.

Using **section** excludes the use of the attributes: **uniqueName**, **sectionId** and **name/property**.

sectionId
The **sectionId** as an **String** in this attribute.

Using **sectionId** excludes the use of the attributes: **section**, **uniqueName** and **name/property**.

uniqueName
The unique name of the section.

Using **uniqueName** excludes the use of the attributes: **section**, **sectionId** and **name/property**.

name
The key value to look up an existing bean. There is two ways of using this bean.

Either the bean must be of type **neo.xredsys.api.Section**.

Or the bean must have a property to get the **Section**. Then the attribute **property** must be used.

Using **name** excludes the use of the attributes: **section**, **uniqueName** and **sectionId**.

property
Identifies the JavaBeans property (of the bean identified by the **name** attribute) to get the **Section** to be used by the tag. If not specified, the bean identified by the **name** attribute itself will be used as the value.

This attribute can not be used without the **name** attribute.

7.1 section:ancestorView

Creates an ancestor view for the given section.

Syntax

```
<section:ancestorView
   height="..."?
   id="..."
   includeRoot="..."?
   name="..."?
   property="..."?
   section="..."?
   sectionId="..."?
   uniqueName="..."?/>
```

Attributes

id, mandatory, no runtime expressions
The name of the page-scoped attribute to create with this view.

section
The section to be used to get the ancestor view.

sectionId
Id of the section to be used to get the ancestor view.

uniqueName
Unique name of the section to be used to get the ancestor view.

name
Name of bean to get section to be used to get the ancestor view.

property
Property to get the section to be used to get the ancestor view. This attribute can not be used without the **name** attribute.

height

includeRoot
Optional boolean value specifying if the root section is to be included in the view. The default is 'true'.

7.2 section:lastChanged

This tag is deprecated.

7.3 section:expiresCache

Expires the nearest cache element if the specified section is updated.

This tag will locate the enclosing **<util:cache>** tag and expire that cache. If there is no enclosing **<util:cache>** nothing will happened.

If you repeat the tag, the cache will be expired if either one of the specified sections are updated.

See also

util:cache (section 9.1)

Syntax

```
<section:expiresCache
   name="..."?
   property="..."?
   section="..."?
   sectionId="..."?
   uniqueName="..."?/>
```

Attributes

section
The section to be used to expire the cache.

sectionId
Id of the section to be used to expire the cache.

uniqueName
Use the section with this unique name instead of using the "default" section.

name
Name of bean to get section to be used to expire the cache.

property
Property to get the section to be used to expire the cache. This attribute can not be used without the **name** attribute.

7.4 section:link

This tag is deprecated. We recommend that you ask the section-object about the url, and generate the html-tag by yourself. This is a more flexible solution. Example:

```
<a href="<util:valueof param="section.url"/> ">name</a>
```

7.5 section:parameter

This tag is deprecated. Use jstl instead.

7.6 section:name

This tag is deprecated. We recommend that you ask the section-object about the name. Example:

```
<util:valueof param="section.name"/>
```

7.7 section:recursiveView

Create a view that includes the specified section, and all of its sub-sections, optionally limited to a level. If no section is specified, the "default" section will be used as the base section.

Syntax

```
<section:recursiveView
   depth="..."?
   id="..."
   includeRoot="..."?
   name="..."?
   property="..."?
   section="..."?
   sectionId="..."?
   uniqueName="..."?/>
```

Attributes

id, mandatory, no runtime expressions
The name of the page-scoped attribute to create with this view.

depth
Include this number of levels in the view. Setting depth to 1 limits the view to the root element, while setting it to 2 includes the immediate children.

includeRoot
Optionally, specify whether the base section (specified by the section or sectionName attributes) is to become part of the view. By default, this value is "true", and can be set to either "true" or "false".

section
The section to be used to get the recursive view.

sectionId
Id of the section to be used to get the recursive view.

uniqueName
Unique name of the section to be used to get the recursive view.

name
Name of bean to get section to be used to get the recursive view.

property
Property to get the section to be used to get the recursive view. This attribute can not be used without the **name** attribute.

7.8 section:use

Sets the "current" section to a different section within this tag. All section tags that relate to the current section in the body of this tag will use the specified section instead of any previous current section. Normally, the default-section is set by the template mechanism, but for a portion of JSP it may be desirable to use a different section. This tag will make any tags within its body use this section. One of section or sectionName must be specified. The section will be also be available using the "apisection" request attribute.

The section:use tag does **not** alter the default presentation pool. Tags that operate on the pool, rather than on the section (e.g. **<template:element>**), will not work as expected unless the **<template:defaultPresentationPool>** is used as well.

Syntax

```
<section:use
   name="..."?
   property="..."?
   section="..."?
   sectionId="..."?
   sectionIdInt="..."?
   source="..."?
   sourceId="..."?
   uniqueName="..."?>
   ...
</section:use>
```

Attributes

section
A scriptlet that returns the section object to be used as the new "default" section.

sectionId
Optionally,specify Section ID (a numeric string) of an attribute that contains a Section object to be used as the new "default" section.

uniqueName
The unique name of the section that is to be the new "default" section. This section will be loaded from the current default publication.

name
The name of an attribute that contains a Section object to be used as the new "default" section.

property
Specifies the name of the property to get the wanted Section on the bean specified by name. It makes no sense specifying property if name is not set.

This value may be a simple, indexed, or nested property reference expression. If not specified, we expect that the bean specified by name is a Section.

source
Source of the section to set as "default" section. This attribute can not be used without the sourceId attribute.

sourceId
SourceId used to identify the section to be set as "default" section. This attribute can not be used without the source attribute.

sectionIdInt
The Section ID (an int) of an attribute that contains a Section object to be used as the new "default" section.

7.9 section:view

Create a view that includes this section only. Normally, this tag is only used together with other views, using the add and subtract tags.

Syntax

```
<section:view
   id="..."
   name="..."?
   property="..."?
   section="..."?
   sectionId="..."?
   uniqueName="..."?/>
```

Attributes

`id`, mandatory, no runtime expressions
The name that will be given to this view, in the page scope.

`section`
The section to be used to get the view.

`sectionId`
Id of the section to be used to get the view.

`uniqueName`
Unique name of the section to be used to get the view.

`name`
Name of bean to get section to be used to get the view.

`property`
Property to get the section to be used to get the view. This attribute can not be used without the **`name`** attribute.

8 template Tag Library

This tag library contains tags to include ECE templates and other JSP files. Addition to this there are tags to set different default objects(as PresentationArticle and PresentationPool), and some simple testing.

The tags used to set the default objects are normally only invoked once per request. This is typically done in the **`index.jsp`** or **`article.jsp`** files.

The essential tags in this taglib are:

- call
- insert
- parameter

```
<%@ taglib uri="http://www.escenic.com/taglib/escenic-template" prefix="template" %>
```

We usually use **`template`** as prefix to this tag library.

8.1 template:call

This tag is deprecated. Use **`jsp:includcde`** instead.

8.2 template:defaultPresentationArticle

This tag is deprecated and there is no replacement.

8.3 template:defaultPresentationPool

This tag is deprecated and there is no replacement.

8.4 template:defaultPublication

This tag is deprecated and there is no replacement.

8.5 template:defaultSection

This tag is deprecated and there is no replacement.

8.6 template:element

This tag is deprecated. Please use **<template:insert>** instead.

8.7 template:hasElement

This tag is deprecated and there is no replacement.

8.8 template:hasNoElement

This tag is deprecated.

8.9 template:insert

This tag is deprecated.

8.10 template:parameter

This tag is deprecated.

8.11 template:serviceParameter

This tag is deprecated.

9 util Tag Library

This tag library contains tags that do not fit into any other categories. There are tags to do contitional testing, handle iterations, cache management etc.

The essential tags in this taglib are:

- cache
- switch
- toggle
- valueof

```
<%@ taglib uri="http://www.escenic.com/taglib/escenic-util" prefix="util" %>
```

We usually use **util** as prefix to this tag library.

9.1 util:cache

Use this tag to cache a block of JSP code. Typically, you use this around code that is resource hungry, such as iterating through 200 sections. Use the JSP statistics functionality to guide you in using this tag.

To explicity expire a cached block (before the cache tage's own expire time), you can to use these tags:

- **<article:expiresCache>**
- **<section:expiresCache>**

Be careful **not** to create nested **<util:cache>** elements as this may cause significant performance problems. Beware of templates calling/ including other templates where the caller has a **<util:cache>** around the callee, which itself also has a **<util:cache>** element.

If you get a JSP with **<util:cache>** taking just over 10 seconds to render, typically 10023ms, it is probably because of nested **<util:cache>** elements.

See also

article:expiresCache (section 2.5) , section:expiresCache (section 7.3)

Syntax

```
<util:cache
   blocking="..."?
   blockingTimeout="..."?
   expireTime="..."?
   id="..."
   includeArticle="..."?
   includeSection="..."?>
   ...
```

```
</util:cache>
```

Attributes

`id`, mandatory
Each cached fragment must have an id. The id is used to identify thecached fragment.

The id can be any string. If two or more instances of the cache tag have identical ids, they will share the cached content as well.

Typically you want to create a unique id for every instance of the cache tag. You can do so for example by using section ids,article ids, string constants, or a combination of them.

`includeArticle`
Will make it easier to create a unique id. If set to **`true`** the tag will automaticly include the current article into the key.

The tag will handle if there is no current article, but the id will not be unique.

Default it is set to **`false`**.

`includeSection`
Will make it easier to create a unique id. If set to **`true`** the tag will automaticly include the current section into the key.

The tag will handle if there is no current section, but the id will not be unique.

Default it is set to **`false`**.

`expireTime`
Sets the how long we will wait until the cached items are expired from the cache. The time is by default in minutes. You can changed that to seconds, minutes or hour by adding a s,m or h after wanted time.

e.g. 120s means 120 seconds and 2h means 2 hours

Please note that setting the attribute to 1 second as this is **not** the same as disabling the cache.

`blockingTimeout`
Sets how long we will wait for the cached item to be created. This is only relevant during system startup. The only reason to set this, is if you have got a resource that normally takes more time than the default value to create. The value is in milliseconds. Set to 0 to block forever, but this may potentially lock your system and should be avoided.

Default is 10 seconds.

blocking
Set this to "false" to avoid waiting for a cached item to finish loading. This will leave the cached part of the page empty. This option is only relevant during startup of the system.

Default is to block waiting for the cached item to finish. Timeout is set with blockingTimeout.

9.2 util:case

This tag is deprecated. Use jstl instead.

9.3 util:catalogs

This tag is deprecated. No replacement.

9.4 util:dateFormatter

This tag is deprecated. Replace with jstl.

9.5 util:equal

This tag is deprecated. Replace with jstl.

9.6 util:evaluate

This tag is deprecated. Use jstl instead.

9.7 util:expiresCache

Expires the cache if there is an event on the specified object.

See also

util:cache (section 9.1)

Syntax

```
<util:expiresCache
   object="..."/>
```

Attributes

`object`, mandatory
The object we want the cache to be depended on

9.8 util:includeExtContent

Retrieves content from a page specified by the given URL. URL's has to be absolute, and can not be to a frame-set. If the HTTP Get times out(see timeout attribute) or failes for any other reason, body of tag will be shown.

The content of the URL can be cached by setting the useCaching attribute=true. Also url's that failes will be cached. this means that this url will be 'marked' as failed until the background job retrieves content succesfully or content is thrown from cache.

- If content has been in cache longer than validTime without beeing accessed, expire content.
- If content has been in cache longer than refreshTime, try to retrieve new version of content.
- if refreshTimeout > timeout this value is used as timeout. This makes it possible to let the background job retrieve content from slow sites that might have timed out on first atempt.

validTime,refreshTime and refreshTimeout is set in the /neo/io/services/ExternalContentManager component

Syntax

```
<util:includeExtContent
   cookie="..."?
   includeAll="..."?
   timeout="..."?
   url="..."
   useCaching="..."?>
   ...
</util:includeExtContent>
```

Attributes

`url`, mandatory
The URL of the page where content is to be retrieved from

`includeAll`
Return all content, without removing <HTML>, <HEAD> and <BODY>-tags. Possible values are true or false. The default value is false.

`useCaching`
Enable caching of the retrieved content. See general description for details.

`timeout`
The number of milliseconds before cancel retrieval of content. Body of tag will be shown if timeout is reached. If not set default value from /neo/io/services/ExternalContentManager component will be used

cookie
use as in following example

9.9 util:image

Image tag. Used to get image height and width.

Syntax

```
<util:image
   file="..."?
   id="..."
   name="..."?
   property="..."?>
   ...
</util:image>
```

Attributes

id, mandatory, no runtime expressions
id attribute

file
file attribute

name
name attribute

property
property attribute

9.10 util:isEmpty

This tag is deprecated. Replace with jstl.

9.11 util:isNotEmpty

This tag is deprecated. Replace with jstl.

9.12 util:logMessage

Will write messages/comments to either the browser log or inline html comments in the the html page. This tag is meant to be used by the template developers when creating the JSP-pages.

The tag can make it easier to locate error in the JSP-pages.

When sending messages to the BROWSER-log use the message attribute, and when creating inline html-comments use the comment attribute.

The messages and comments will only be displayed when the parameter:'debug=true' is added to the url, e.g. www.escenic.com?debug=true

If the **comment** attribute is empty, the body content of the tag will be used instead.

Syntax

```
<util:logMessage
   category="..."?
   comment="..."?
   commentStyle="..."?
   level="..."?
   message="..."?>
   ...
</util:logMessage>
```

Attributes

category
The category all messages will be logged to

This attribute can only be used together with the message attribute.

comment
The comment will be added to the jsp page in the specified comment style.

commentStyle
Style of the comment. The default comment style is html-comments. This attributes allows you to change the comment style.

Allowed styles: html, xml, css, javaScript.

This attribute must be used together with the comment attribute.

level
Debug level. Default value is debug.

This attribute can only be used together with the message attribute.

message
The message to be written to the browser log.

9.13 util:lookup

Looking up a component in Nursery

Syntax

```
<util:lookup
   id="..."
   name="..."
   scope="..."?
   type="..."/>
```

Attributes

`id`, mandatory
Id of the object we are looking up

`name`, mandatory
Name of the component

`type`, mandatory
type of the component

`scope`
Scope of the component

9.14 util:loop

This tag is deprecated. Replace with jstl.

9.15 util:notEqual

This tag is deprecated. Replace with jstl.

9.16 util:pager

Sets startIndex, endIndex, displayStartIndex and totalSize attributes which can be used for paging.

Syntax

```
<util:pager
   length="..."
   name="..."
   startIndex="..."?
   startIndexName="..."?/>
```

Attributes

`name`, mandatory
The name of the attribute that contains object to be used for paging. This can either be an array or a collection

`length`, mandatory
Specifies the length of the pageintervall

`startIndexName`
Specifies the name of the attribute that contains the start index, i.e. the first item to display on the page. If not set, then the "start" request parameter is checked.

`startIndex`
Specifies the start index, i.e. the first item to display on the page. If not set, then startIndexName is checked.

9.17 util:parameter

Defines a parameter to be passed on to the "sendMail" tag. 'name' defines the name of the parameter, and 'value' defines the string-value of the parameter. Valid names are mailTo, mailCc, mailBcc, mailFrom, mailerName, subject, url, plainContent, htmlContent, charset and attachments. Minimum requirements to send a mail are mailTo, MailFrom and content. The charset parameter will override the default charset defined in the /neo/util/PostCardSender component.

See also

util:sendMail (section 9.21)

Syntax

```
<util:parameter
   key="..."
   name="..."?
   property="..."?
   scope="..."?
   value="..."?>
   ...
</util:parameter>
```

Attributes

key, mandatory
The name of the parameter that is to be passed onto the "sendMail" tag. The allowed parameters are: mailTo, mailCc, mailBcc, mailFrom, mailerName, subject, url, plainContent, htmlContent and attachments.

name
The name of the attribute to use as the value of this parameter. If not scope is specified we will look for the specified object in all scopes. This attribute can not be used together with the "value" attribute.

property
Property to get from the specified bean with the name attribute. This attribute can not be used without the name attribute.

scope
The scope in which to search for when using the "name" parameter. This optional parameter defaults to a value which means that all scopes will be searched. The scope attribute may be set to "page", "request", "session", "application", or nothing.

value
The value of the parameter to be passed to the parameter. This attribute can not be used together with the "name" and "property" attributes.

9.18 util:pluginResources

Executes the body of the tag for each plugin resource that has a resource matching the specified attributes.

Syntax

```
<util:pluginResources
   area="..."?
   id="..."
   target="..."?
   task="..."?
   type="..."?/>
```

Attributes

`id`, mandatory, no runtime expressions
The name of the nested scripting variable used to hold the resource object that contains the uri and label (or labelKey indicating the key of a label in a resource bundle).

`type`
Used to filter based on the type of resource requested. If type is not specified, all types will be considered. Types are typically "internal-link" or "external-link"

`target`
Used to filter based on the target of resource requested. If target is not specified, all targets will be considered. Target is typically "escenic" or "admin"

`task`
Used to filter based on the task of resource requested. If task is not specified, all tasks will be considered. Tasks are target-specific. Each target typically has certain entry points.

`area`
Used to filter based on the area of resource requested. If area is not specified, all areas will be considered. Areas are target/task specific.

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.util.Collection`**.

9.19 util:profiler

Adds JSP Profiling for a fragment of JSP code. The profiling will only be effective if the profiling has been enabled as described in the page developer's guide.

Profiling should be specified for any page where the request has been forwarded. Typically, this happens to be the wireframe file, which is often the outermost page in a request. The **`path`** attribute should be set in this outermost tag.

```
<util:profiler path="/template/ver1/wireframe/normal.jsp">
  <jsp:include ... />
</util:profiler>
```

For profiling fragments of a page (for example a loop, or a computationally intensive part), use the **`fragment`** attribute:

```
<util:profiler fragment="compute-1">
  <% new Baby(); %>
</util:profiler>
```

Profiling fragments can be useful to track down performance problems in a particular JSP template. It can also be used when the page contains branches (for example **`<util:switch>`** tags) where there are several distinct parts are executed independently of each other.

Syntax

```
<util:profiler
   fragment="..."?
   path="..."?>
   ...
</util:profiler>
```

Attributes

`path`
The fully qualified path of the current JSP template. This attribute should only be used in the outer-most profiling tag. collector.

`path` may not be specified if **`fragment`** is specified

`fragment`
An arbitrary name of a fragment of the JSP template. Fragments are typicaly used to profile portion of the JSP.

`fragment` may not be specified if **`path`** is specified

9.20 util:rewrite

A simple tag to rewrite/calculate the path to specified file. The calculate path will be returned as a **`java.lang.String`**.

There are two different **`types`** of paths you can get from this tag. One is the url to the file, and the other is the file path to the file.

There are several **`roots`** that can be used to calculate the path.

- publication
- template
- multimedia
- section

Syntax

```
<util:rewrite
   file="..."
   id="..."?
   root="..."
   type="..."?/>
```

Attributes

`id`, no runtime expressions
Id is not mandatory. If specified the path is returned as a scripting variable (and associated page scope attribute). If not specified the path will be written.

`file`, mandatory
Name of the file.

`root`, mandatory
The root of the file. Allowed values:

`type`
The type of path. Allowed values:

Default value is url.

9.21 util:sendMail

Sends mail with specified content to specified recipients. Content can be plain text, html or content from a specified URL.

See also

`util:parameter` (section 9.17)

Syntax

```
<util:sendMail
   type="..."?>
   <util:parameter.../>
</util:sendMail>
```

Attributes

`type`
Determines what kind of content a mail can have. Allowed values are:

Content-type of entire message is multipart/alternative.

9.22 util:switch

This tag is deprecated. Use jstl instead.

9.23 util:toggle

Sets up a toggle instance, which allows a template developer to repeatedly switch between two or more strings.

See also

`util:toggleNext` (**section 9.24**)

Syntax

```
<util:toggle
   declare="..."?
   id="..."
   items="..."?
   name="..."?
   toScope="..."?
   value="..."?/>
```

Attributes

`id`, mandatory
Used to identify the toggle that will be made available as a scripting-variable.

NOTE: When declear is set to 'true', it is illegal to use a runtime expression in the 'id' attribute.

`declare`, no runtime expressions
Turn declaration of scripting-variables on/off. If set to 'true' will scripting-variables be created. Else it will not be created.

Default is set to true.

`items`
A java.util.List of values that are to be toggled. The Toggle will be initialized with the first value in the list. If the list does not contain String elements, it will convert them to strings first.

`value`
A comma separated list of values that are to be toggled. The Toggle will be initialized with the first value in the list, and all of the values will be trimmed of leading and trailing whitespace before being used.

`name`
The name of a scoped attribute that will be used as the list of names. The attribute can be a List, or a comma separated string.

`toScope`
To which scope the toggle will be put. Allowed values are: page, request, session and application.

9.24 util:toggleNext

Advances the specified Toggle one step forward in its list. If the Toggle reaches the end, then it automatically restarts at the beginning.

See also

`util:toggle` (**section 9.23**)

Syntax

```
<util:toggleNext
   name="..."?
   scope="..."?
   toggle="..."?/>
```

Attributes

`name`
The name of the attribute that contains the Toggle object. One of name or toggle must be specified.

`toggle`
A scriptlet that evaluates to the Toggle object to advance. One of name or toggle must be specified.

`scope`
On which scope the tag will look for the toggle. Allowed values are: page, request, session and application.

9.25 util:valueof

This tag is deprecated. Replace with jstl.

9.26 util:click

Places a click counter on a link

Syntax

```
<util:click
   categoryName="..."
   categoryProperty="..."?
   id="..."?
   systemName="..."
   systemProperty="..."?
   urlName="..."
   urlProperty="..."?/>
```

Attributes

`id`, no runtime expressions
The id we want to use to look up the scripting variable set by this tag.

The id is not required here. If its not use the tag will simply print out the result.

`urlName`, mandatory
The url to place a click counter on

`urlProperty`
The property

`systemName`, mandatory
The click counter system to use

`systemProperty`
The property

`categoryName`, mandatory
The name to place the link in

`categoryProperty`
The property

10 view Tag Library

The essential tags in this taglib are:

- iterate
- relationships

```
<%@ taglib uri="http://www.escenic.com/taglib/escenic-view" prefix="view" %>
```

We usually use **`view`** as prefix to this tag library.

10.1 view:add

Add two views together, to produce a view that includes the objects in both views. If any items were in both views, the new view will only include one of the objects.

Syntax

```
<view:add
   id="..."
   nameView1="..."?
   nameView2="..."?
   propertyView1="..."?
   propertyView2="..."?
   view1="..."?
   view2="..."?/>
```

Attributes

`id`, mandatory, no runtime expressions
The name of the page scoped attribute to create with the result view.

`view1`
The first view to add. This must be a scriptlet that returns a View object.

`nameView1`
Name of bean to be used to get view1.

`propertyView1`
Name of property to get view1 from the bean specified by the nameView1-attribute.

`view2`
The second view to add. This must be a scriptlet that returns a View object.

`nameView2`
Name of bean to be used to get view2.

`propertyView2`
Name of property to get view1 from the bean specified by the nameView2-attribute.

10.2 view:subtract

Subtract a view from another view, to produce a view that includes the difference of two views. Any item in view1 will be included in the result view, except for any item in view1 which is also in view2, which will not be included in the result view.

Syntax

```
<view:subtract
   id="..."
   nameView1="..."?
   nameView2="..."?
   propertyView1="..."?
   propertyView2="..."?
   view1="..."?
   view2="..."?/>
```

Attributes

`id`, mandatory, no runtime expressions
The name of the page scoped attribute to create with the result view.

`view1`
The view to subtract from. This must be a scriptlet that returns a View object.

`nameView1`
Name of bean to be used to get view1.

`propertyView1`
Name of property to get view1 from the bean specified by the nameView1-attribute.

`view2`
The view which will be removed from the first view. This must be a scriptlet that returns a View object.

`nameView2`
Name of bean to be used to get view2.

`propertyView2`
Name of property to get view2 from the bean specified by the nameView2-attribute.

10.3 view:iterate

Iterate over each item in a view. The body of this tag will be executed once for each item in the view.

Syntax

```
<view:iterate
   depth="..."?
   id="..."
   length="..."?
```

```
   name="..."?
   offset="..."?
   property="..."?
   type="..."?
   view="..."?>
   <view:relationships.../>
</view:iterate>
```

Attributes

`id`, mandatory, no runtime expressions
The current item in the view will be exported using a variable with this name. In addition, the page-scoped attribute of this name will also contain the current item.

`depth`
The maximum depth to be iterated through in this tag. Elements that are deeper will not be iterated over. If it is not present there will not be any limit on the depth.

`length`
The maximum number of entries (from the view) to be iterated through in this tag. If not present, there will be no limit on the number of iterations performed.

`offset`
The zero-relative index of the starting point at which entries from the view will be iterated through. If not present, zero is assumed (meaning that the view will be iterated from the beginning).

`type`, no runtime expressions
The variable created will be of the type specified by this attribute. If this variable is not specified, the variable will be exported as a generic object.

`view`
The view to iterate over. It must be specified by a scriptlet that returns a View object.

`name`
Name of bean that contains the view.

`property`
Name of the property to get the view from the bean specified by the name-attribute.

10.4 view:forEachLevel

When iterating over a view of a hierarchy, this tag makes it possible to repeat a certain text once for each "level". If this tag contains the text "*", then this tag will print out one "*" for each level that the iteration is doing.

Syntax

```
<view:forEachLevel
```

```
   startAt="..."?>
   ...
</view:forEachLevel>
```

Attributes

`startAt`
What level should we "start" at? If this is not specified, the root of the hierarchy will be used. If this is specified as a number, the number will be subtracted from the level, before iteration starts.

10.5 view:relationships

When iterating over a view of some hierarchy, this tag makes a Relationship object available in the page context. This Relationship object can tell a page developer if the currently iterated item is a sibling of, a parent or child of, an ancestor or descendant of any other specified item.

The relationships tag must be used from within an "iterate" tag in the same tag library.

The tag allows the page developer to specify what item it is to check up against, by specifying the name/scope attributes or the node attribute.

Syntax

```
<view:relationships
   id="..."
   name="..."?
   node="..."?
   property="..."?
   scope="..."?>
   <view:first.../>   <view:last.../>
</view:relationships>
```

Attributes

`id`, mandatory, no runtime expressions
The name of the page-context attribute to which the newly created Relationships object will be created.

`name`
The name of an attribute that contains the item to check for relationships. All scopes are checked when looking for the node; this behavior can be changed by specifying the "scope" attribute

`property`
Name of the property to get the item to check for relationships.

This attribute can not be used without the the **`name`** attribute.

`scope`
The scope in which to look for the base node.

`node`
The actual item to check for relationships.

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.common.util.tree.Relationships**.

10.6 view:first

Will render body of this tag only if we are first element on a level.

This is an example on how you would use this tag.

```
<%@ taglib uri="/WEB-INF/escenic-view.tld" prefix="view" %>
<%@ taglib uri="/WEB-INF/escenic-section.tld" prefix="section" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>

<section:recursiveView id="aView" uniqueName="ece_frontpage" />

<view:iterate id="current" name="aView">
  <view:relationships id="relation" name="section">
    <view:first>
      <div class="level">
    </view:first>
    <div class="section">
      <bean:write name="current" property="name" />
    </div>
    <view:last>
      </div>
    </view:last>
  </view:relationships>
</view:iterate>
```

Syntax

```
<view:first>
   ...
</view:first>
```

10.7 view:last

Will render the body of this tag only if we are the last element on a level. This tag will iterate the amount of time needed to close the content openend in it's sibling tag.

This is an example on how you would use this tag.

```
<%@ taglib uri="/WEB-INF/escenic-view.tld" prefix="view" %>
<%@ taglib uri="/WEB-INF/escenic-section.tld" prefix="section" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>

<section:recursiveView id="aView" uniqueName="ece_frontpage" />

<view:iterate id="current" name="aView">
  <view:relationships id="relation" name="section">
    <view:first>
      <div class="level">
    </view:first>
    <div class="section">
      <bean:write name="current" property="name" />
    </div>
    <view:last>
      </div>
    </view:last>
```

```
  </view:relationships>
</view:iterate>
```

Syntax

```
<view:last>
   ...
</view:last>
```

11 tag Tag Library

This library contains tags that operate on **`PresentationTag`** beans.

In order to use these tags in a JSP file you must include the following prefix declaration in the file:

```
<%@ taglib uri="http://www.escenic.com/taglib/escenic-classification" prefix="tag" %>
```

`tag` is the prefix normally used for this tag library.

11.1 tag:use

Loads a tag specified with the **`tagid`** attribute and assigns it to the variable specified with the **`var`** attribute.

Syntax

```
<tag:use
   tagId="..."
   var="...">
   ...
</tag:use>
```

Attributes

`tagId`, mandatory
The **scheme** or URI of the tag to be loaded.

`var`, mandatory, no runtime expressions
The name of the scripting variable to which the specified tag is to be loaded.