



## Documentation

# Advanced Template Development Guide

## Jahia EE v6.1

Jahia delivers the first **Web Content Integration Software** by combining Enterprise Web Content Management with Document and Portal Management features.

### Jahia

9 route des Jeunes,  
CH-1227 Les acacias  
Geneva, Switzerland

[www.jahia.com](http://www.jahia.com)  
The Company website

[www.jahia.org](http://www.jahia.org)  
The Community website

April 2010

# Summary

1	Overview .....	5
1.1	Introduction .....	5
1.2	What's in this documentation? .....	5
2	Externalized Content definitions .....	6
2.1	CND Basics .....	7
2.2	CND Files .....	8
2.3	Field types .....	9
2.4	Field Selectors .....	10
2.5	Container definition .....	11
2.6	Page definitions .....	13
2.7	Mixin types .....	14
2.8	Metadata definitions .....	15
2.9	Default values .....	16
2.10	Constraints and validators .....	17
2.11	Read only properties .....	17
2.12	Indexation configuration .....	18
2.13	Titles and labels .....	20
3	Tag Libraries .....	21
3.1	Standard Tag Libraries .....	21
3.2	Jahia Tag Libraries .....	23
3.2.1	Jahia Template Tag Library .....	24
3.2.2	Jahia UI Component Tag Library .....	26
3.2.3	Jahia Utility Tag Library .....	28
3.2.4	Jahia Functions Tag Library .....	29
3.2.5	Jahia Query Tag Library .....	30
3.2.6	Jahia Search Tag Library .....	32
3.2.7	Jahia JCR Tag Library .....	33
3.2.8	Jahia JCR Tag Library 2 (EE) .....	34
3.2.9	Jahia Internal Tag Library .....	35
3.3	Named Jahia Variables For JSP Expression Language .....	36
3.3.1	Constants .....	39

3.4	Other Third Party Tag Libraries .....	40
3.5	Custom Tag Libraries .....	41
4	Creating and Accessing Content.....	42
4.1	Accessing Container Lists .....	42
4.1.1	Container List Addressing.....	43
4.1.2	Expose ContainerListBean in Page Context.....	46
4.1.3	Pagination and Other Attributes.....	46
4.2	Accessing Containers .....	47
4.2.1	Iterating over Containers in a List .....	47
4.2.2	Retrieving a Specific Container .....	47
4.2.3	Retrieving Random Containers.....	48
4.2.4	Accessing Containers by Category.....	48
4.2.5	Accessing Containers with Expressions .....	48
4.2.6	Exposing ContainerBean in Page Context .....	50
4.3	Accessing Fields.....	51
4.3.1	Retrieving Fields of a Container .....	51
4.3.2	Iterating over Fields in a Container.....	51
4.3.3	Retrieving a Specific Field .....	52
5	Navigation Features .....	53
5.1	Menus .....	53
5.1.1	Caching.....	55
5.2	Sitemap.....	55
5.2.1	Caching.....	57
5.3	Navigation history .....	58
6	Search and query.....	59
6.1	Search features .....	59
6.1.1	Detailed features.....	59
6.1.2	Used frameworks.....	62
6.2	Search vs Query .....	64
6.2.1	Comparison .....	64
6.3	Building Search forms.....	66
6.3.1	Simple search form.....	66
6.3.2	Advanced site search with metadata .....	66
6.3.3	Advanced document search with metadata.....	68
6.3.4	Display search results.....	69

6.4	Advanced search customization .....	70
6.4.1	Configure local specific indexing .....	70
6.4.2	"Did you mean" - spellchecker suggestion facility .....	71
6.4.3	Autocompletion - suggested typing.....	76
6.4.4	Indexing policies .....	76
6.4.5	Cluster setup.....	77
6.4.6	Search result highlighting .....	77
6.4.7	Cached Lucene filters / faceting .....	77
7	Web templates quick customization .....	79
7.1	CSS overview .....	79
8	Cache Configuration .....	80
8.1	Inside Jahia caches .....	80
8.2	Cache configuration.....	81
8.3	HTML Fragment Cache .....	83
8.3.1	What is the HTML Fragmented cache? .....	83
8.3.2	How to integrate it?.....	83
9	Event Handling and Catching.....	87

# 1 Overview

## 1.1 Introduction

Jahia delivers the first Web Content Integration software by combining Enterprise Web Content Management with Document Management and Portal features.

By leveraging state of the art Open Source frameworks and libraries, Jahia offers a complete solution for developing, integrating, delivering, and managing content across intranets, extranets, and internets with a much lower total cost of ownership than proprietary systems.

## 1.2 What's in this documentation?

This documentation aims to describe how to define the structure of a Jahia site with the new content definition mechanism from the basics to the advanced capabilities of the Jahia EE v6.1 templating API.

Should you have questions, please do not hesitate to contact us as mentioned on our website (<http://www.jahia.com>) or Community website (<http://www.jahia.org>).

## 2 Externalized Content definitions

A Jahia site can be seen as an organized set of content object, which structure has to be predefined.

Jahia model defines 4 base types of content objects:

- **Fields:** The smallest content object, is an atomic content like text, link... some fields are localized and can have a different value in each language. A field must be declared inside a container. A field must have a type; it has a title and a description, a default value, a selector that defines the way the value is entered by the author, and other properties.
- **Containers:** The base type of structured content. It has a set of named fields and sub container lists. The container is always inside a container list. It can
- **Container lists:** Simply a list of containers, with all containers of the same type. Container lists can be defined to contain one single container.
- **Pages:** Represent a web page. Its structure defines which container lists are available in the page. It can contain only containers lists.

The structure of the fields, containers, lists and pages of a template set must be defined before creating content.

Compared to Jahia 5, content definitions have been separated from the templates, and have to be declared prior to the JSPs. This mechanism replaces old tags `<declareContainerList>`, `<declareContainer>`, and `<declareField>`, which cannot be used anymore in the templates.

Jahia EE v6.1 introduces a new file format for declaring all type of content that will be stored in the Jahia web content management system. This file will describe what containers and fields are present in a specific template, the type of the fields, properties of the containers, ... This content definition mechanism is an extension of the CND file defined by the JCR specification (<http://jcp.org/en/jsr/detail?id=283> )

## 2.1 CND Basics

The CND file will contain two types of structured content, containers and pages. Each type has a unique name and belongs to one domain. Generic type provided by Jahia is part of the "Jahia nodetypes" domain (jnt). Each template set has its own domain.

Types are described by the list of properties (fields) and sub nodes (container lists). The notation gives the possibility to declare type inheritance, constraints on fields, default values, versioning options, and other special attributes.

The Jahia main content container previously defined is described in compact nodetype notation as:

```
[jnt:mainContentContainer] > jnt:container
  smallText mainContentTitle primary
  bigText mainContentBody
  file mainContentImage
  sharedSmallText mainContentAlign (choicelist) = resourceKey(left) indexed='no' <
  resourceKey(left), resourceKey(right), resourceKey(default)
```

Which says: mainContent is a container definition (inherit from type container), from the jnt domain. It has 4 properties, mainContentTitle, mainContentBody, mainContentImage, and mainContentAlign.

mainContentTitle is a smalltext, mainContentBody is a bigtext. mainContentImage is a reference to a file node. mainContentAlign is a sharedSmallText, using a choicelist (dropdown) selector, that must be either « left », « right » or « default ».

The "full" template from the web templates, which declares 3 containers, is defined as:

```
[web_templates:full] > jnt:page, web_templates:header, web_templates:footer
  containerList maincontent (jnt:mainContentContainer)
  containerList portlet (jnt:portletContainer [addMixin="jmix:skinnable"])
  containerList columnA_box (jnt:box [addMixin="jmix:skinnable",
  availableTypes="jnt:filesBox,jnt:textsBox,jnt:linksBox,jnt:rssBox,jnt:IframeBox,
```

The "full" type is a page definition (inherits from page), from the web\_templates domain. It has 3 container lists, mainContent, portlet, and columnA\_box.

## 2.2 CND Files

Jahia comes with a set of predefined types that can be used for templates, but each template set also needs to add its own types. All these types are gathered in different .cnd files. System types are stored in /etc/repository/nodetypes. Template set can define their own .cnd file by adding in the template.xml file the following statement: `<definitions-file>definitions.cnd</definitions-file>`

The cnd file has a very simple structure:

- The list of namespaces and prefixes that will be used in the definitions. Namespaces are similar to XML namespaces: a prefix is associated to a full domain name. For example, `<nt = 'http://www.jcp.org/jcr/nt/1.0'>` maps the "nt" prefix to the standard jcr namespace for nodetypes.
- The list of container, page, and mixin types, as described in the following sections.

Each cnd file can contain any number of definitions. A definition can use definitions from another file, only if the other definition has already been loaded. A template file can reference any system file, as system files are loaded prior to the templates file. A template file can reference a file from another template set if it inherits from it.



## 2.3 Field types

All Jahia fields must have one of the following types:

- `smallText`: Simple unformatted text, with one value per language. Can be edited with a simple text field, multiline text field or taken from a list of predefined values.
- `sharedSmallText`: Same as small text, but shared in all languages.
- `bigText`: Formatted text, language dependant. Author uses a full text editor to enter the value.
- `date`: Contains a date, with or without time.
- `page`: Link to another page, internal to jahia or external link.
- `file`: Link to a file node inside jahia file management system.
- `portlet`: Displays a portlet or a mashup element.
- `integer`: Simple integer number.
- `float`: Float number.
- `boolean`: True or false field, usually edited with a checkbox.
- `category`: Value is chosen from the categories tree.
- `color`: Simple color value, chosen with a color picker.

## 2.4 Field Selectors

A selector is a tool used by the author to enter the value of a field. A selector can have optional parameters. Each field type has a default selector, but the template developer can change for each field the selector he wants to use and its options.

- Text: is a simple text field used for entering unformatted text. By default, the field is displayed on one single line.
- Richtext: fck editor, mainly used for bigtexts
- Choicelist: a drop down menu with the list of constraints is built. Can take the "sort" option to sort the entries of the drop down menu.
- Datetimepicker: date and time selector, can be used with date type only. "format" option specifies the format of the date.
- Datepicker: date only selector, can be used with date type only. "format" option specifies the format of the date.
- Category: displays the category picker. Can take the "root" option to specify the root category.
- File: displays the file picker. Can take the "mime" option to specify a mime type, and the "filters" option for name filtering.
- Portlet: displays the portlet picker
- Page: jahia page selector is an engine that allows to create a subpage, link to an existing page or create an external link. This selector is only used for the Jahia page type. The "type" option with "direct", "external", and "internal" values can limit the type of link to create. The "templates" option can define the list of templates the user can use for this field (in case of "direct" or "internal" links)
- color - simple color picker
- checkbox - checkbox field

## 2.5 Container definition

Container definitions are definitions that inherit from the `jnt:container` base type. Once the definitions have been created, they can be reused in other container definitions and page definitions.

The definitions starts with the domain and name of the container in brackets, followed by a `>` sign and the type it inherits from. This can be the base `jnt:container` or another container previously defined:

```
[jnt:comment] > jnt:container
```

This can be followed by a list of options. "abstract" tells that the definition cannot be used directly, and can only be extended ( for example, `jnt:box` ). The validator option can defines a struts validator class (see "Constraints and validation" below).

A list of items is then declared. A container can contains either field items or container lists items.

A field declaration consists at least of a field type and a name. The field type can be one of the following:

`smallText`, `sharedSmallText`, `bigText`, `date`, `page`, `file`, `portlet`, `integer`, `float`, `boolean`, `category`, `color`

This is a followed by the field name. Field name must be an alphanumeric identifier, and can contain underscore (`'_'`), but no other character.

```
bigText newsDesc
```

The template developer can then choose an optional selector for the field. The name of the selector is put between parentheses, and can have a list of option between brackets. Note that not all selectors can be used on all type of fields - it's impossible to define a page selector for a `smallText` field. Selector options depend of the selector, as defined in the "Selectors" section.

```
page newslink (page[type="external,internal"])
```

An optional default value can then be specified here, with an `'='` symbol. See the "Default values" section for more details.

A list of options can then be specified:

- Mandatory: field must have a value (see "Constraints and validators")
- Protected: field is read-only (see "Read only properties")
- Hidden: field is hidden in the engine
- Indexed: scoreboost and analyzer: set the index configuration (see "Indexation configuration")

Finally, field constraints can be specified here, with a < symbol followed by a list of constraints (see "Constraints and validators")

A sub container list declaration consists of the keyword "containerList" followed by the name of the container list, or "singleContainer" for a container list containing one single element. It must then be followed by the type of containers that will be contained in the list, between parenthesis characters. This has to be a valid name of a container type previously defined.

```
containerList pageContainer (jnt:pageContainer)
```

The type name can also be followed by a list of options that will be applied on the container.

- availableTypes
- addMixin

Example:

```
containerList columnB_box (jnt:box [addMixin="jmix:skinnable",
availableTypes="jnt:filesBox,jnt:textsBox,jnt:linksBox"])
```

A list of options can then be specified:

- Mandatory: at least one container is always present
- workflow: specifies the type of workflow that will be applied on each new container. Can be "none", "inherited", or "nstep-<workflow name>"

Examples:

```
singleContainer rssContainer (jnt:rssContainer) mandatory
containerList comment (jnt:comment) workflow='none'
```

It is also possible to use the standard CND notation, which starts with a '-' for a property definition, and a '+' for a node definition, instead of the "type + name" jahia syntax. Check the JCR specification for a complete description of this format.

## 2.6 Page definitions

Page definitions are definitions that inherit from the `jnt:page` definition. A page cannot have any field. Only containers can be used here. Here's an example of a simple « full » page type that defines 3 containers:

```
[web_templates:full] > jnt:page
containerList maincontent (jnt:mainContentContainer)
containerList portlet (jnt:portletContainer [addMixin="jmix:skinnable"])
containerList columnA_box (jnt:box [addMixin="jmix:skinnable",
availableTypes="jnt:filesBox,jnt:textsBox,jnt:linksBox,jnt:rssBox,jnt:IframeBox
```

Page definitions are very similar to container definitions - the only difference is that they inherit from the type " `jnt:page` " or another page type instead of the `jnt:container` type, and they cannot have field definitions. Container lists are defined exactly the same way as sub container lists in a container.

The mapping between template type and page definitions is done in the `template.xml` file. Each `<template>` entry must have a "page-type" attribute, which contains the name of the corresponding type:

```
<template name="full" display-name="Full" filename="full.jsp" page-type="web_templates:full" />
```

## 2.7 Mixin types

Sometimes, different container and/or page definitions need to share the same set of item definitions. Instead of repeating the same list of items in all definitions, it is possible to define "mixin" types, that will contain these item definitions, and that can be reused in all other definitions.

A type can be defined as mixin by specifying the "mixin" keyword just after the list of inherited types. Any page or container definition can reuse any number of mixin types. The list of mixin types has to be added just after the primary inherited type.

For example, web templates define two mixin types:

```
[web_templates:header] mixin
containerList  basicLinkHeader (web_templates:basicLinkContainer)
containerList  navLink (jnt:navLink)
singleContainer logo (web_templates:logoContainer)

[web_templates:footer] mixin
singleContainer footerContainerList (web_templates:footerContentContainer)
containerList  basicLinkFooter (web_templates:basicLinkContainer)
singleContainer logoFooter (web_templates:logoContainer)
containerList  bottomLinks (web_templates:bottomLinksContainer)
```

Which are reused in all page types of the template set:

```
[web_templates:full] > jnt:page, web_templates:header, web_templates:footer
```

## 2.8 Metadata definitions

Metadata for containers and pages are also defined in the CND files. The standard file contains a base metadata type, based on a list of standard mixin types:

```
[jmix:contentmetadata] > mix:created, mix:createdBy, mix:lastModified, jmix:lastPublished,  
jmix:categorized, jmix:description mixin
```

If the standard set of metadata needs to be modified for one container type, a new mixin type inheriting from the `jmix:contentmetadata` can be created, and added to the container or page type definition. All metadata mixin types should inherit from the `jmix:contentmetadata` type – this will allow the system to make the distinction between normal data and metadata.

For example, the following new metadata type is created:

```
[jmix:complexMetadata] > jmix:contentmetadata mixin  
categoryField j:mycategory
```

And can be assigned to a specific container definition:

```
[jnt:mainContent] > jnt:container, jmix:complexMetadata  
smallText title
```

This will add the « `myCategory` » metadata to the `mainContent` container only, keeping the normal set of metadata for other containers.

## 2.9 Default values

Default values can be specified in the definitions file.

Jahia extends the syntax by allowing the use of functions in the default value that will preset the field with dynamic values. Functions are identified by the use of parenthesis. For example, the following property will be set to the current time by default:

```
date myDate = now()
```

To get the value from a resource bundle, the following syntax can be used:

```
smallText title = resourceKey(mykey)
```

If the expected value has to be computed for a specific need, an initializer class can be created and used here. The class must implement a specific interface, and will return a default values. The name of the function will actually be the name of the class - for example, to use a function " myInitializer " , a class `org.jahia.services.content.nodetypes.initializers.MyInitializer` has to be defined and should implement the `ValueInitializer` interface.

```
smallText test = myInitializer()
```

The following functions are understood and parsed:

Function name	Result
<code>now()</code>	Get the current time
<code>eval(expr)</code> (not yet implemented)	Execute the JEXL expression
<code>query(query)</code> (not yet implemented)	Execute the xpath query to get the default value
<code>currentUser()</code>	Return the current user name
<code>resourceKey(key)</code>	Takes the value from a resource bundle
<code>skins()</code>	Return the list of available skins

Note: in order to use parenthesis in a default value, the string has to be placed between quotes.



## 2.10 Constraints and validators

JCR definitions give the ability to define mandatory fields, and to add regex constraints. For example, the following field is mandatory and must only contains letters:

```
smallText name mandatory < '[a-zA-Z]*'
```

If needed, multiple regexp constraints can be specified. The value should match at least one of them. Constraints can also be plain text values, as in the following example:

```
smallText name mandatory < 'one', 'two', 'three'
```

The value must then match one the three possible values.

As for the default values, it is also possible to use functions instead of literal values:

```
[jmix:skinnable] > jmix:layout mixin  
sharedSmallText skin (choicelist[image]) < skins()
```

Jahia also allows validation based on the struts validator framework for more complex validation needs. A bean must be created for containers that need to be validated, and a validation rule has to be added in the configuration files. The validator is then declared in the container definition with:

```
[web_templates:navigationContainer] < jnt:container  
validator='org.jahia.beans.navigationValidationBean'
```

## 2.11 Read only properties

A property with the « protected » option will be set as read only and cannot be edited through the engine. The default value cannot be changed.

```
[jmix:lastPublished] mixin  
- j:lastPublishingDate (date) protected  
- j:lastPublisher (string) protected
```

## 2.12 Indexation configuration

It is possible to define if a field should be indexed, if it is sortable, facetable, searchable in a plain fulltext search or not, if the field has to be tokenized, by which analyzer, and an optional scoreboost on that field.

The "indexed" option can have the following values: no, tokenized, untokenized, which means, respectively, that the field won't be indexed, indexed as tokens, or indexed untokenized. If you do not specify the indexed option, Jahia will treat all date, Boolean, integers and float types as untokenized and the other field types as tokenized.

The "scoreboost" option is used to modify the boost factor of a specified field. If you do not specify the option, the default scoreboost of 1.0 will be taken.

The "analyzer" option is used to change the default analyzer (as a key to the compass configuration, which is defined in applicationcontext-compass.xml). If you do not specify the analyzer option, Jahia will use for all date, Boolean, integer and float types the keyword analyzer and for the other field types the standard analyzer.

With the "fulltextsearchable" option you can control whether the field should be included when doing fulltextsearch and part of the search hit excerpts. On default all date, boolean, float, integer, color, application fields are not fulltextsearchable, while the other field types are. The option can have the values: yes or no in order to change the default behavior.

The "sortable" option is here to make text fields, which are on default tokenized, also usable for sorting, where the field should be untokenized. So in such a case there will be two fields in the Lucene document, where the untokenized version will not be stored, but only available for sorting.

The "facetable" option is similar to the sortable option, as it will also create a second version of the field in the Lucene document in order to store an otherwise tokenized field untokenized, which is required to make faceted search and browsing (with the ability to see how many hits there are per facet value). The difference to the sortable option is, that the sortable is resolving any multilingual values to the language of the Lucene document, whereas the facetable option will use the key (e.g. resource bundle marker) of the value to be indexed.

Example:

```
[web_templates:jobContainer] > jnt:container, web_templates:title
smallText      reference
smallText      businessUnit facetable sortable
sharedSmallText contract (choicelist) facetable sortable <
    resourceKey(contract1),
    resourceKey(contract2),
    resourceKey(contract3),
    resourceKey(contract4)
smallText      town sortable
smallText      country (choicelist) facetable sortable < country()
smallText      educationLevel analyzer='keyword' sortable
bigText        description
bigText        skills
containerList   jobAnswers (web_templates:answerJobContainer)
```

Remaining indexation configuration is done in applicationcontext-compass.xml and jahiaresource.cpm.xml files.

## 2.13 Titles and labels

Titles and labels of the fields and containers are taken from a resource bundle file that comes along with the cnd file.

Standard jahia types use the « JahiaTypesResources » bundle file. Types defined in template packages use the bundle provided within the template package.

The bundle key used for types is simply the fully qualified type name, with colon ( : ) replaced by underscore ( \_ ).

```
jnt_mainContentContainer = Main content
```

The bundle keys for fields are built from the type and property names. For example, for the mainContent type and its title field:

```
jnt_mainContentContainer.mainContentTitle = Title
```

If the field use the resourceKey() function, the key will actually be concatenated with the field bundle key. For example, the mainContentAlign field of the mainContentContainer uses three resourceKey functions, resourceKey(left), resourceKey(right) and resourceKey(default):

```
jnt_mainContentContainer.mainContentAlign.left = Left  
jnt_mainContentContainer.mainContentAlign.right = Right  
jnt_mainContentContainer.mainContentAlign.default = Default
```

In the case a field is not correctly entered by the author, and does not match the constraint, the error message can also be customized using the following resource bundle key:

```
jnt_mainContentContainer.mainContentTitle.invalidConstraint = Title must contains only az  
chars !
```

## 3 Tag Libraries

Jahia has very much improved its tag libraries in order to simplify template development. Our goal was to remove Java code scriptlets within the JSP templates for Jahia, in order to keep them more readable, ease template re-use and migration to newer Jahia releases and result in a steeper learning curve for Jahia newbies.

If you have complex requirements, which cannot be solved with the current taglibs, you could still use the Jahia API in Java scriptlets, but we recommend to either write custom tags (can be done with JSP syntax in files with a .tag extension), perhaps contribute them to Jahia or ask the Jahia community to enhance the provided taglibs.

### 3.1 Standard Tag Libraries

We encourage Jahia template developers to use the JSP Expression Language (EL) and the JavaServer Pages Standard Tag Library (JSTL). Documentation and tutorials can be found here:

<http://java.sun.com/products/jsp/jstl/reference/docs/index.html> and here:

<http://java.sun.com/products/jsp/jstl/>

Jahia EE v6.1 is based on the Servlet 2.4 and JSP 2.0 specification (J2EE 1.4) and can be deployed on application servers, which are at least supporting these standards. The JSTL tag library version in Jahia is 1.1.2.

We have removed the taglib references in the web.xml, so it is important that you use the correct standard URI in the taglib directives of your JSPs. For JSTL these are:

Library	Recommended Prefix	URI
Core	c	<a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a>
XML	x	<a href="http://java.sun.com/jsp/jstl/xml">http://java.sun.com/jsp/jstl/xml</a>
Internationalization	fmt	<a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a>
SQL	sql	<a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a>
Functions	fn	<a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a>

## 3.2 Jahia Tag Libraries

Jahia EE v6.1 offers eight tag libraries packaged in the jahia-taglib\*.jar. Notice that we have completely redesigned our tag libraries compared to Jahia 5. There is also no support for jesi tags in Jahia EE v6.1 anymore.

Library	Short name	URI
Template	template	<a href="http://www.jahia.org/tags/templateLib">http://www.jahia.org/tags/templateLib</a>
UI components	ui	<a href="http://www.jahia.org/tags/uiComponentsLib">http://www.jahia.org/tags/uiComponentsLib</a>
Query	query	<a href="http://www.jahia.org/tags/queryLib">http://www.jahia.org/tags/queryLib</a>
Utility	utility	<a href="http://www.jahia.org/tags/utilityLib">http://www.jahia.org/tags/utilityLib</a>
Functions	functions	<a href="http://www.jahia.org/tags/functions">http://www.jahia.org/tags/functions</a>
Search	s	<a href="http://www.jahia.org/tags/search">http://www.jahia.org/tags/search</a>
JCR	jcr	<a href="http://www.jahia.org/tags/jcr">http://www.jahia.org/tags/jcr</a>
Internal	internal	<a href="http://www.jahia.org/tags/internalLib">http://www.jahia.org/tags/internalLib</a>

### 3.2.1 Jahia Template Tag Library

The Template Jahia tag library is the one, which will be used in every Jahia template as it has tags which are mandatory for templates and tags to retrieve Jahia content.

Area	Function	Tags
Template	Declaring a Jahia template, its header and body part. All templates must use these tags.	template templateHead templateBody
	Generates a „meta“ HTML tag retrieved from Jahia meta data to put in the <head> part of the template	meta
	Tags to display a list of boxes or the box itself. Very similar to the well-known containerList and container tags, but for boxes only	boxList box
	Include resources considering Jahia template set inheritance	include param executeSuper
	Tags to use inside JSP files in order to render skins	includeContent includeExtension
	Displays a random container from a container list (tag must be surrounded by *containerList tags)	randomContainer
	Used to create or display container lists	containerList absoluteContainerList relativeContainerList
	Used to iterate over containers in a list (tag must be surrounded by *containerList tags)	container
	Display or get Jahia content fields within containers (should be surrounded by a container tag, or the container bean can be passed as argument)	field
	This tag will display a metadata value or you can get the value as a date object for formatting purpose	metadata
	Get container or field by ID	getContainer getContainerField getField
	Form handling tags	jahiaPageForm formFromDefinition



Category utility tags	category categoriesByProperties getContentObjectCategories
If the default display / rendering offered by the field tag does not suit the template developer needs, he/she can use these tags to display, in a custom way, images, links and files.	image link file
Tags to control caching	cache
Add dependencies to a ContentObjectKey inside a set	addDependency
Generate place holder for a JahiaModule.	gwtJahiaModule
Create html structure for layout manager (My Portal)	layoutManagerArea layoutManagerBox layoutManagerMode
Generates URL links to general Jahia engines, pages or popup.	composeUrl
Displays a theme selector drop-down and the selected theme CSS file(s).	themeDisplay

### 3.2.2 Jahia UI Component Tag Library

The Jahia UI component tag library is an optional taglib to create/display some convenient user interface components, but should most probably be used by every template, because it has often used components like navigationMenu, currentPagePath, login components etc.

Area	Function	Tags
UI Component	Display and handling of a tree control	tree
	Display the portlet's supported modes (e.g. Edit, View, Preview,...) or window states (Maximizing, Minimizing, Closing)	portletModes windowStates
	Generates the HTML listing of Jahia user-groups or users with certain query options.	groupList userList
	Displays/creates a collection of images thumbnails from a WebDAV path	thumbView mediaGallery displayRandomImage
	Optional action menu tag (use if default disabled)	actionMenu
	Navigation tags to create/display menus, page-path, breadcrumbs, sitemaps...	navigationMenu currentPagePath sitemap
	Tags to define and create a login form directly inside a template.	loginArea loginUsername loginPassword loginRememberMe loginButton loginErrorMessage
	Generate links in order to switch languages when browsing the site with possibility to use own styles.	languageSwitchingLinks declareLanguageLinkDisplay displayLanguageSwitchLink displayLanguageFlag displayLanguageState
	Display a combo box allowing to change the window size of a list.	displayWindowSizeComboBox
	Displays a theme selector drop-down and the selected theme CSS file.	themeSelector

	Tag allowing to display date picker and a input field or a calendar based on GXT.	<code>dateSelector</code> <code>calendar</code>
	Displays a category selection drop-down in order to e.g. allow filtering by categories. Another tag displays category titles of a category list.	<code>categorySelector</code> <code>displayCategoryTitle</code>
	Renders the link to the file/path selection engine.	<code>fileSelector</code> <code>folderSelector</code>
	Renders the link to the page selection engine.	<code>pageSelector</code>
	Display a button to subscribe to a RSS feed.	<code>rssButton</code>
	Renders subscribe/unsubscribe link for change notifications of a content object.	<code>subscribeButton</code>

### 3.2.3 Jahia Utility Tag Library

The Jahia Utility tag library is an optional tag library, but should also be used by every template as it has important tags like the useConstants to expose Jahia constants to JSTL for not having to hardcode some values.

Area	Function	Tags
Utility	Utility tag to use a log4j logger in a template	logger
	Utility tag to be able to use static constants in JSTL	useConstants
	Resource bundle tags to be used by a GWT JahiaModule	gwtResourceBundleDictionary gwtResourceBundle gwtAdminResourceBundle gwtEngineResourceBundle applicationResources
	Display all attributes of the current session	sessionViewer
	Resource bundle utility tag	resourceBundle
	Displays the properties of the current site or page.	displaySiteProperties displayPageProperties
	Utility date tag in order to convert dates and add days, hours, etc.	dateUtil
	This tag will display the username if none property is passed otherwise it will display the requested property. This tag is designed to work well with the container cache.	userProperty
	Resolves the Web path to the specified resource considering template set inheritance	resolvePath
	Display a picto image depending on a file's MIME type.	getPicto
	Retrieves a value of a given request parameter and provides a default if the parameter does not exist.	getRequestParameter

### 3.2.4 Jahia Functions Tag Library

The Jahia Functions tag library is an optional tag library and consists of functions, which can be used directly in JSTL expression language.

Area	Function	Tags
Functions	Prints out the attribute values for a tag, based on the provided Map	<code>attributes</code>
	Returns the requested value if it is not empty, otherwise returns provided default value	<code>default</code>
	Get page ID from url key	<code>getPidFromUrlKey</code>
	Utility to remove HTML tags from a String	<code>removeHtmlTags</code>
	Tags for checking content subscriptions	<code>isSubscribed</code> <code>isNotSubscribed</code>
	Check if the current user is or is not member of a group and a tag to display the current user's name.	<code>memberOf</code> <code>notMemberOf</code>
	Appends 3 strings together and returns the concatenated value	<code>stringConcatenation</code>

### 3.2.5 Jahia Query Tag Library

The Jahia Query tag library is an optional tag library to create filters, searchers, sorter for container lists. In future these tags will also be used for page and file search, which for now are handled by the Jahia Search tag library.

Area	Function	Tags
Query	This tag is used to declare an abstract Query Object Model.	definition
	This tag is used to build a Concrete ContainerQueryBean from a query model.	containerQuery createContainerQueryBean
	The Selector tag is used to define the Node Type Name of Nodes that will appear in the query result.	selector
	This tags are used to create query constraints within an abstract or concrete query definition.	comparison childNodes descendantNode equalTo greaterThan greaterThanOrEqualTo lessThan lessThanOrEqualTo like notEqualTo fullTextSearch
	This tags are used to create logical connections on query constraints within an abstract or concrete query definition.	and or not
	This tag is used to declare ordering of query results.	sortBy
	This tag is used to pass properties to the query object model	setProperty propertyValue
	This tag is used to declare a Journal Logs Query.	logsQuery
	These tags are used to define constraints on Journal Logs queries.	contentDefinitionNamesConstraint usernamesConstraint
	These tags are used to execute a query on Journal logs and create a bean of the result.	executeLogsQuery logEntryBean

	This tag is used to create a ContainerSorterBean from a given LogsQuery object.	<code>createContainerSorterFromLogsQuery</code>
	Display a category selection drop-down, or other, in order to allow filtering by categories.	<code>categoryFilter</code>

### 3.2.6 Jahia Search Tag Library

The Jahia Search tag library is used for defining page or file repository search forms. The tags can also be used to display results of hardcoded searches.

Area	Function	Tags
Search	Tags to create a form for entering search input to create simple or advanced search forms on Jahia pages or the file repository.	<div>form</div> <div>created</div> <div>createdBy</div> <div>date</div> <div>documentProperty</div> <div>documentType</div> <div>fileLocation</div> <div>fileType</div> <div>language</div> <div>lastModified</div> <div>lastModifiedBy</div> <div>pagePath</div> <div>rawQuery</div> <div>term</div> <div>itemsPerPage</div>
	Tag to define the URL of the results page	<div>resultsPageUrl</div>
	Tags to execute the search defined in the form and to iterate over the results.	<div>results</div> <div>resultIterator</div>
	Used to display the results in a table (e.g. for saved search)	<div>resultTableSettings</div> <div>resultTable</div>
	Exposes a descriptor for the specified document property into the page scope.	<div>documentPropertyDescriptor</div>
	Create html structure for an open search.	<div>opensearchBox</div>



### 3.2.7 Jahia JCR Tag Library

The Jahia JCR tag library is experimental and exposes Jahia content in a JCR way.

Area	Function	Tags
JCR	Check if node is of a certain type	isType
	Iterate over file node types and retrieve files	fileNodeTypes file
	Iterate over node types and retrieve node type	nodeTypes nodeType
	Get label or name of node types.	nodeTypeLabel nodeTypeName
	Iterate over properties and retrieve property.	properties property
	Get label, value and input for a property	propertyLabel propertyValue propertyInput

### 3.2.8 Jahia JCR Tag Library 2 (EE)

The Jahia JCR tag library is experimental and exposes Jahia content in a JCR way.

Area	Function	Tags
JCR	Expose a JCR node using a given path and an optional scope	node
	Display a relative or absolute link to a JCR node (behaves like <a> tag)	Link
	Allow acces to a property of a node	nodeProperty
	Allow access to a node using an XPath expression and expose it	xpath

### 3.2.9 Jahia Internal Tag Library

The Jahia Internal tag library is supposed to be used only internally by Jahia engines or administration JSPs.

Area	Function	Tags
Internal	Utility tags used in JSP header	JSTools i18n
	Resource bundle utility tags	adminResourceBundle engineResourceBundle setUsrEngineResourceBundle message
	Display an image flag or code corresponding to a language	displayLanguageFlag displayLanguageCode
	Display an icon	displayIcon
	Create buttons or links	jahiaButton
	Init javascript object that contains some jahia parameters as pid required to use Google Web Toolkit.	gwtInit
	Import GWT by its module name.	gwtImport
	Create html structure for display and job report.	processDisplay jobReport
	Displays the corresponding Jahia engine	fileManager workflowManager categoryManager categorySelector

### 3.3 Named Jahia Variables For JSP Expression Language

Jahia puts the following JavaBean objects as attribute in each request object, so they can be accessed with the expression language, some taglibs or even scriptlets:

Attribute name	Class	Description
currentPage	<code>org.jahia.data.beans.PageBean</code>	Provides access to an object that contains information relative to the current page, as well as all the objects in the page
currentSite	<code>org.jahia.data.beans.SiteBean</code>	Provides access to an object that contains information relative to the current site, as well as all the pages in the site.
currentJahia	<code>org.jahia.data.beans.JahiaBean</code>	Provides access to an object that contains information relative to the current Jahia installation, as well as all the sites
currentRequest	<code>org.jahia.data.beans.RequestBean</code>	Provides access to an object that contains information about the current request. You can think of this object as a complement to the standard Servlet API object since it provides method such as <code>isEditMode()</code> , <code>isLogged()</code> , etc..
currentUser	<code>org.jahia.services.usermanager.JahiaUser</code>	This object is the user currently logged in JahiaUser (would be guest if the user is not logged in), and provides access to user properties
dateBean	<code>org.jahia.services.expressions.DateBean</code>	Contains some formatted dates based on currenttdate

Here are some expressions you may find useful:

<pre> \${currentRequest.normalMode} \${currentRequest.compareMode} \${currentRequest.previewMode} \${currentRequest.editMode} </pre>	Returns <code>true/false</code> depending on the mode of the current request
<pre> \${currentRequest.IE} \${currentRequest.IE4} \${currentRequest.IE5} \${currentRequest.IE6} \${currentRequest.IE7} \${currentRequest.NS} \${currentRequest.NS4} \${currentRequest.NS6} </pre>	Returns <code>true/false</code> depending on the user agent of the current request (Internet Explorer, Mozilla)
<pre> \${currentRequest.mac} \${currentRequest.unix} \${currentRequest.windows} </pre>	Returns <code>true/false</code> depending on the user agent's operating system
<pre> \${currentRequest.logged} </pre>	Returns <code>true/false</code> whether user for current request is logged on or not
<pre> \${currentRequest.admin} \${currentRequest.root} </pre>	Returns <code>true/false</code> if user for current request is administrator or even root user
<pre> \${currentRequest.adminAccess} \${currentRequest.writeAccess} </pre>	Returns <code>true/false</code> if user for current request has admin or write access
<pre> \${currentUser.name} </pre>	Returns name of the current user
<pre> \${currentUser.userProperties['somePropName']} </pre>	Returns a property of current user
<pre> \${currentSite.homepageID} </pre>	Returns homepage ID of current site
<pre> \${currentSite.homePage} </pre>	Returns <code>PageBean</code> of homepage of current site
<pre> \${currentSite.id} </pre>	Returns ID of current site
<pre> \${currentSite.templatePackageName} </pre>	Returns template set name of current site
<pre> \${currentSite.siteName} </pre>	Returns current site name
<pre> \${currentPage.level} </pre>	Returns current page level
<pre> \${currentPage.pageID} </pre>	Returns current page ID

<code>\${currentPage.parentPage}</code>	Returns <code>PageBean</code> of the current page's parent page
<code>\${currentPage.title}</code>	Returns title of the current page in the current language
<code>\${currentPage.urlKey}</code>	Returns URL key of the current page
<code>\${pageContext.request.contextPath}</code>	Returns the URL of the current context
<code>\${currentRequest.parmBean.settings}</code>	Returns the <code>SettingsBean</code> to access several Jahia instance configurations
<code>\${currentRequest.parmBean.settings.jsHttpPath}</code>	Returns the URL of Jahia's Javascript files path

If you anyway need to call the Jahia API from scriptlets you will often need the `JahiaData` or `ProcessingContext` object to be passed into the Jahia-API. They can be retrieved like this:

```
<%
JahiaData jData = (JahiaData)
request.getAttribute("org.jahia.data.JahiaData");
ProcessingContext jParams = jData.getProcessingContext();
%>
```

### 3.3.1 Constants

For some taglibs you will need to pass constants as attributes, which are defined in some Jahia classes. If you just hardcode the constant in your template, it may be for instance just a meaningless magic number. So it would be better to directly use the defined constant variable name. You can use the

`<utility:useConstants>` tag to be able to access such static variables from JSTL.

For instance with this definition:

```
<content:useConstants var="queryConstants"
className="org.jahia.query.qom.JahiaQueryObjectModelConstants" scope="application"/>

<content:useConstants var="fieldTypeConstants" className="org.jahia.data.fields.FieldTypes"
scope="application"/>
```

you could then access the static variables like this:

```
${queryConstants.CATEGORY_LINKS}
```

or

```
${fieldTypeConstants.SMALLTEXT}
```

### 3.4 Other Third Party Tag Libraries

Jahia EE v6.1 is packaged with other open source tag libraries, which you can use in your template development. The packages are:

Library	Description	URI
Display tag library (1.1)	High level web-presentation patterns	<a href="http://displaytag.sf.net">http://displaytag.sf.net</a>
Pager Tag Library (2.0)	Paging of large datasets	<a href="http://jsptags.com/tags/navigation/pager">http://jsptags.com/tags/navigation/pager</a>
Struts Tag Libraries (1.2.9)	Struts offers a wide variety of tags, but most of the features are available in JSTL. The Apache Struts group encourages the use of the standard tags over the Struts specific tags if possible.	<a href="http://struts.apache.org/tags-bean">http://struts.apache.org/tags-bean</a> <a href="http://struts.apache.org/tags-html">struts.apache.org/tags-html</a> <a href="http://struts.apache.org/tags-html">http://struts.apache.org/tags-html</a> <a href="http://struts.apache.org/tags-logic">http://struts.apache.org/tags-logic</a> <a href="http://struts.apache.org/tags-nested">http://struts.apache.org/tags-nested</a> <a href="http://struts.apache.org/tags-tiles">http://struts.apache.org/tags-tiles</a>
Unstandard taglib	Availability of ideas before JSTL responds to user demands	<a href="http://jakarta.apache.org/taglibs/unstandard-1.0">http://jakarta.apache.org/taglibs/unstandard-1.0</a>

If you would like to use newer taglib package versions or other tag libraries, which are not packaged with Jahia, you may try exchanging or adding them.

Regarding any AJAX tag library we recommend using GWT or GXT as Jahia is integrating many components with this framework and adding any other AJAX technology will just make the pages bigger and slower to download.



## 3.5 Custom Tag Libraries

You may also develop your own tags, which can even simply be done in a JSP style with using tag files (.tag extension). You can learn more about that here:

<http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSPTags5.html>

## 4 Creating and Accessing Content

In Jahia EE v6.1 the way of defining content and the display of content edit controls has changed.

Instead of doing content definitions in the JSP we have externalized content definitions to a CND file.

From Jahia EE v6.1 onwards action menus' for creating and editing content will already automatically be created when accessing a template page in edit mode. The tags for accessing content described in this chapter are implicitly creating those menus. Action menu tags are still provided for some advanced non-default usage only.

Also UI controls for paging through large container lists are already automatically created and you do not have to use any pagination tags like in Jahia 5.

### 4.1 Accessing Container Lists

Most of the time you will access Jahia content with addressing a container list. Each field is located in a container and each container is located in a container list, even if it is a single container.

Once you have the container list, you can use the `<template:container>` tag in the body of the different container list tags, to iterate over the containers. In the body of the `<template:container>` tag you can then use the `<template:field>` tag to access the different fields in the container.

Jahia container list instances are always local to a page and are accessed by its name. That means that if we have two container lists called `newsList` on two different pages, they are indeed different, even if they share the same definition (which is local to a template).

There are multiple ways to access a container list:

- page local container lists
- absolutely addressed container lists
- relatively addressed container lists
- sub container lists

There is a Test Unit template, which shows the usage of the absolute and relative addressed container list.

#### 4.1.1 Container List Addressing

##### 4.1.1.1 Page Local Addressing

This may be the most often used way to access a container list, as for simply accessing content of the current page you do not have to use absolute or relative addressing. To access the container list located on the current page and named newsList (has to be defined in the externalized content definition), you should use:

Taglib Version:

```
<template:containerList name="newsList">
  <!-- tags to display container list here -->
</template:containerList>
```

Java API Version:

```
JahiaContainerList containerList =
    jData.containers().getContainerList("newsList");
```

##### 4.1.1.2 Absolute Addressing (aka Absolute Container Lists)

In order to reference a container list in an absolute way, we must specify both the name of the list, as well as the page identifier on which the list lives. Here is an example of accessing a container list using absolute addressing:

Taglib Version:

```
<template:absoluteContainerList name="newsList" pageId="news">
  <!-- tags to display container list here -->
</template:absoluteContainerList>
```

Java API Version:

```
JahiaContainerList containerList =
    jData.containers().getAbsoluteContainerList("newsList", "news");
```

The above code will retrieve the container list named newsList on the page with the URL key news.

With using absolute addressing of container lists, it becomes possible to access container lists from multiple templates, effectively sharing content, without sharing the view (we can display the container list as

we want in the target template). Since this addressing is completely dynamic, any change in the source container list will be reflected on all pages that access it through absolute or relative addressing.

You can also alternatively use the pageId instead of the pageKey, as pages may not have URL keys. Be aware that it is not a good practice though to hardcode page IDs in the templates, because page IDs may change when doing export/import. However you can perfectly use page IDs returned from an API, or when passing the page ID via URL query parameter.

There is another way to access container lists using a technique similar to the relative addressing, called page level addressing. Page level addressing is only supported with tags, and allows to access container lists in the current page ancestors by specifying a page level offset. The important thing with this page level offset is that it operates from the top of the tree, not from the current node up. So if we have the current page path that looks like this:

```
page1 -> page2 -> page3 -> page4
```

and we are currently on page4, the following code will access the container list called newsList located on page1:

```
<template:absoluteContainerList name="newsList" pageLevel="1">
  <!-- tags to display container list here -->
</template:absoluteContainerList>
```

#### 4.1.1.3 Relative Addressing (aka Relative Container Lists)

The same way we introduced absolute addressing, it is also possible to access container lists in a relative way, by specifying an offset to the current page in the ancestors. So basically if we have the following page ancestors for page4:

```
page1 -> page2 -> page3 -> page4
```

The following call will return the container list called newsList located on page3:

Taglib Version:

```
<template:relativeContainerList name="newsList" levelNb="1">
  <!-- tags to display container list here -->
</template:relativeContainerList>
```

Java API Version:

```
JahiaContainerList newsList =
    jData.containers().getRelativeContainerList("newsList", 1);
```

Relative access can be very practical, when displaying navigation elements. For example, to retrieve the siblings of a page. To do that you would retrieve the child container list of the parent page, and you could then display all the pages that are at the same level as the current page.

#### 4.1.1.4 Sub Container List Addressing

A container within a container list can itself hold zero or more sub container lists for containers of a different type, which again may have sub container lists and so on.

Due to internal restrictions of the Jahia legacy back-end, all the above addressing (page local, absolute and relative addressing) does not give direct access to sub container lists. In order to access them you will have to retrieve their parent container first and then inside the `<template:container>` tag you will be able to use the `<template:containerList>` to access the sub container list.

For instance:

```
<template:containerList name="newsList">
  <template:container>
    <template:containerList name="newsImagesList">
      <!-- tags to display sub container list here -->
    </template:containerList>
  </template:container>
</template:absoluteContainerList>
```

Alternatively if you know the ID of the parent container holding the sub container list, you could also get the sub container list like this:

```
<template:getContainer containerID="{param.containerID}" valueID="myCont"/>
<template:container srcBeanId="myCont">
  <template:containerList name="newsImagesList">
    <!-- tags to display sub container list here -->
  </template:containerList>
</template:container>
```

#### 4.1.2 Expose ContainerListBean in Page Context

The container list tags allows you to expose the container list as a ContainerListBean object into the page context. With the attribute id you can define the name under which the bean can be retrieved from the context (e.g. id="aName"). This way you could access some interesting methods via JSTL tags or expression language.

For instance:

```
<template:absoluteContainerList name="newsList" pageId="news" id="newsBean">
  Size: ${newsBean.fullSize}
</template:absoluteContainerList>
```

#### 4.1.3 Pagination and Other Attributes

With some of the attributes you can control how many list items should be fetched and displayed at all and per page. Jahia EE v6 now implicitly creates the UI controls to navigate through a list. In Jahia 5 you had to explicitly use some pagination tags to do that.

maxSize	Maximum number of containers retrieved from the persistent store. Default setting is Integer.MAX_VALUE
displayPagination	If <code>false</code> is set, then pagination controls are not displayed on top of the list even if the number of containers exceeds the <code>windowSize</code> . <code>true</code> is the default value.
displayPaginationAtEndOfList	If <code>false</code> is set, the pagination controls are not displayed at the end of the list. <code>true</code> is the default.
windowSize	Number of containers displayed in a page. On default all containers are displayed in one page.
windowOffset	With this setting you can start displaying containers in a list from an offset onwards and not from the beginning, like on default.
nbStepPerPage	Specifies the number of clickable steps (URL links) per page. On default all the pages will be numbered and listed with a clickable link.

## 4.2 Accessing Containers

### 4.2.1 Iterating over Containers in a List

Inside the different container list tags, you can access the containers, simply by using the

```
<template:container> tag.
<template:containerList name="newsList">
  <template:container>
    ...
  </template:container>
</template:absoluteContainerList>
```

With that tag you will iterate over containers of the list. The number of containers iterated through depends on the settings in the parent containerList tag, where you can define the number of containers displayed on a page.

Furthermore the returned containers depend on optionally used query tags, which are used to filter and sort containers by certain criteria or to fetch containers from different container lists. Please refer to Container Query tags – sort, search, pagination for more information on how to use these powerful tools to retrieve content from other pages and to see the tags in action.

The container tag will implicitly handle the action menu display in edit mode and will control the container HTML cache and the display of containers in the right skin.

You can see the Java Doc or the taglib description (on [jahia.org](http://jahia.org)) to get more information about the possible attributes of the tag.

### 4.2.2 Retrieving a Specific Container

Sometimes you may want to access only a specific container and not the whole list of containers. For instance you may have a template, which displays a list of containers, but when clicking on a container, you want to display only the details of that container on the next page. You could for instance pass a query parameter in the URL with the container ID and access the container like this:

```
<template:getContainer containerID="{param.containerID}" valueID="myCont"/>
<template:container srcBeanId="myCont">
  <!-- tags to display container fields here -->
</template:container>
```

### 4.2.3 Retrieving Random Containers

Sometimes you may want to display containers randomly, for instance banners or advertisements on a page. You could use the `<template:randomContainer>` tag for that, which has to surround one of the `containerList` tags. In the `displayedContainer` attribute you have to set the number of random containers you want to display.

```
<template:randomContainer displayedContainer="3">
  <template:containerList name="newsList">
    <template:container>
      ...
    </template:container>
  </template:containerList>
</template:randomContainer>
```

### 4.2.4 Accessing Containers by Category

Another possibility for re-using content is through the use of categories. Categories allow users to classify any content object, making it possible to access them using a different hierarchy than the page navigation hierarchy. You can for instance retrieve all containers of a certain category.

### 4.2.5 Accessing Containers with Expressions

Expression Language is a compliment to JSP tags and Java code. It gives access to JavaBean-compliant objects in a succinct manner.

The three most popular EL for Java Web containers are JSTL EL, Jakarta Struts EL and JEXL. JEXL is supported in field declaration, while the slightly less functional JSTL EL and Struts EL can be used anywhere within the template.

#### 4.2.5.1 Expression Language in Field Declarations

Jahia is capable of using an expression instead of static values for field default values. This is especially useful when building dynamic multi-valued drop-down lists. You can then populate the field's default value with expressions, which will be evaluated to retrieve the value from another content object.

See here for more info about field value initializers.

#### 4.2.5.2 Expression Language in Templates

Jahia is also capable of using JSTL EL and Jakarta Struts EL directly within your templates. EL expression start with the `${` marker and end with the associated `}` marker.



Before every EL expression is executed, Jahia sets up the context for it. All the objects loaded in the context are subsequently accessible to EL. Jahia loads the following objects in the context: `PageBean`, `SiteBean`, `RequestBean`, `JahiaBean` and `JahiaUser`. These correspond respectively to the following names: `currentPage`, `currentSite`, `currentRequest`, `currentJahia` and `currentUser` (see Named Jahia Variables for JSP Expression Language.)

This means that all JavaBean-compliant getter/setter methods of these objects are accessible from EL. Please refer to the Javadoc for an extensive list of available methods and the JSP 2.0's EL specification for usage information.

The listing below illustrates how EL can be used to access various fields and container parameters:

```
<div>
  <fmt:message key="username"/>: ${currentUser.username}<br/>
  <fmt:message key="label.siteKey"/>: ${currentSite.siteKey}<br/>
  <fmt:message key="label.homepage.title"/>: ${currentSite.homePage.title}<br/>
  <fmt:message key="label.homepage.id"/>: ${currentSite.homePage.pageID}<br/>
  <fmt:message key="label.currentPage.id"/>: ${currentPage.pageID}<br/>
  <fmt:message key="label.currentPage.languageStates"/>: ${currentPage.languageStates}<br/>
  <fmt:message key="label.containerList.size"/>:
  ${currentPage.containerLists.maincontent.size}<br/>
  <fmt:message key="label.containerList.size.alt"/>:
  ${currentPage.containerLists['maincontent'].size}<br/>
  <fmt:message key="label.container.name"/>:
  ${currentPage.containerLists.maincontent.name}<br/>

  <c:set var="firstContainer"
value="${currentPage.containerLists.maincontent.containers['0']}" />
  <fmt:message key="label.firstContainer.id"/>: ${firstContainer.ID}<br/>
  <fmt:message key="label.firstContainer.title"/>:
  ${firstContainer.fields['maincontent_mainContentTitle'].value}<br/>

  <fmt:message key="label.container.name"/>:
  ${currentPage.containerLists.maincontent.name}<br/>
  <c:forEach var="maincontentContainer" items="${currentPage.containerLists.maincontent"
varStatus="iterationStatus">
    <fmt:message key="label.container">
      <fmt:param value="${iterationStatus.count}" />
    </fmt:message><fmt:message key="label.title"/>:
    ${maincontentContainer.fields.maincontent_mainContentTitle.value}<br/>
  </c:forEach>
</div>
```

Note that, unlike JEXL, the current the JSP 2.0's EL implementation doesn't support method parameters in method calls. So for example `SiteBean.getLightContainerLists(String name)` method isn't supported. However, the elements of a Map object returned by a parameterless method (e.g. `ContainerBean.getFields()` method) are accessible by specifying the key in the `Map` of the targeted object. For example if title is the key in the `Map` to the targeted object it looks like this:

```
${currentPage.containerLists.testCnt.containers[0].fields['title'].value}
```

This also stands for methods, which return `List`, or arrays where the next EL argument is assumed to be an Integer index into the array. An example is present above in the `containers[0]` term, which returns element 0 of the `ArrayList`, itself returned by `ContainerListBean.getContainers()` method.

WARNING: When available we recommend using the Jahia tags for accessing containers and fields, as they are better optimized for performance and implicitly caring about optimal usage of caches, creating of skins, action menus, pagination controls and others. Use the JSTL tags only if you have a good reason.

#### 4.2.6 Exposing ContainerBean in Page Context

Like you can expose a container list as a Java bean in the page context, you can also expose a container as a `ContainerBean` with setting the name in the id attribute.

For instance:

```
<template:containerList name="newsList">
  <template:container id="newsContainer">
    Container ID: ${newsContainer.ID}
    JCR path: ${newsContainer.JCRPath}
    Source page ID: ${newsContainer.pageID}
  </template:container>
</template:absoluteContainerList>
```

## 4.3 Accessing Fields

### 4.3.1 Retrieving Fields of a Container

Inside the `<template:container>` tag, you can access the fields, simply by using the `<template:field>` tag and setting the name attribute with the field name defined in the CND. Jahia does not support automatic type conversion yet.

The tag is used to set the field's bean or value into a variable in the page context for further processing. When setting the display attribute to true, it will already write the field's value to the response output stream. You can specify a defaultValue to be used in case the field has not been edited yet.

Some attributes control the size of an excerpted output and the characters to use at the end of the excerpt (maxChar, maxWord, continueString).

Other attributes are used to control the editing of the field or the display of field action menus in EDIT mode

- inlineEditingActivated sets whether the field can be edited directly in the page
- cssClassName sets a CSS class name to be used in the rich text editor to display the content using the configured CSS class name

You can see the Java Doc or the taglib description to get more information about all the possible attributes of the tag.

### 4.3.2 Iterating over Fields in a Container

As the field's name is a required attribute in the `field` tag. You could iterate over all fields in a container in the following way:

```
<template:containerListname="containerListName" id="processedContainer">
  <c:forEach items="{processedContainer.fields}" var="entry">
    <c:setvar="field"value="{entry.value}"/>
    <div class="title">${field.title}</div>
    <div class="value"><template:fieldname='${field.name}'/></div>
  </c:forEach>
</template:containerList>
```

### 4.3.3 Retrieving a Specific Field

Sometimes you may want to display a single field without having to access the container list and the container first. There are two possibilities to do that.

If you know the ID of the field, you can use `<template:getField>`. Beware that it is not a good practice to hardcode field IDs in the template, because field IDs may change, when using import/export or replications.

```
<template:getField fieldID="{param.fieldID}" valueID="myField"/>
${myField.title}: ${fn:escapeXml(myField.value)}
```

The other alternative is to not use the ID of the field, but if you have a `ContainerBean` you can access the field by its name:

```
<template:getContainerField containerBean="{containerBean}" fieldName="newsBody"
valueID="myField"/>
${myField.title}: ${fn:escapeXml(myField.value)}
```

## 5 Navigation Features

Easy navigation is the key for a good website. Menus are the backbone of any navigation system, completed by a relevant sitemap and tools such as current page path or breadcrumb.

In Jahia 5 a template developers had to use many lines of code and most probably scriptlets to provide the navigational controls. Furthermore the different solutions very often had performance issues and were the source of a slow responding website.

Therefore Jahia EE v6.1 now provides a set of tags, simple but flexible, to create the different navigational elements.

### 5.1 Menus

Many different menus exist. Here are the most common:

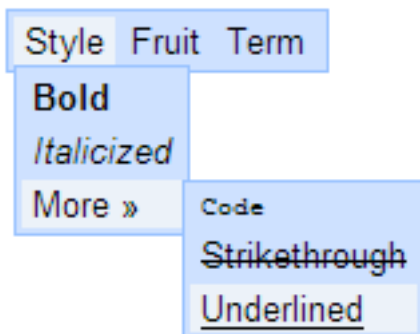
- standard left (or right) vertical menu
- top horizontal menu
- menu bar with cascading items

While the standard side menu may have several depth levels, we may want to stick to two or less sublevel for cascading menus, in order not to overwhelm the user.

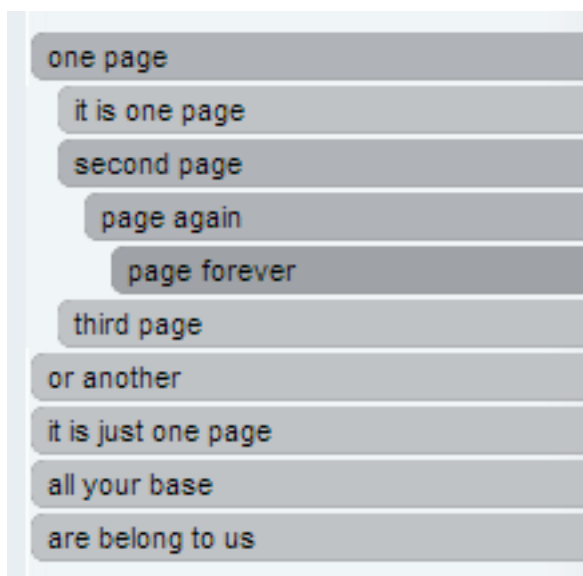
Here is an example of a flat top menu, without sublevels (tabs):



If we add one or two sublevels, we end up with a menu bar like this one:



A side navigation menu with hierarchy would look like one of these:



Including a menu now is as easy as typing this kind of code in a template:

```
<ui:navigationMenu kind="topTabs">
<ui:navigationMenu kind="sideMenu">
<ui:navigationMenu kind="dropDownMenu"> (not implemented in current community edition)
```

This tag will already automatically show action menu icons in EDIT mode, so that editors will be able to add/update/delete pages or reorder them in the menu.

To improve customization there are several parameters, such as depth limit, source container list and field name, conditional display of certain items in case of a deep hierarchy, CSS styles,....

For instance, to add a side menu with no depth limit but with only four levels displayed, one should write:

```
<ui:navigationMenu kind="sideMenu" dispNumber="4">
```

If instead of having a display limit we want a recursion depth limit, use:

```
<ui:navigationMenu kind="sideMenu" maxDepth="4">
```

See the JavaDoc to see the comprehensive list and the meaning of attributes available in this tag.

For the visual customization you should find information in the CSS description document.

### 5.1.1 Caching

This tag uses the container cache, storing information within the properties map to easily format HTML output. No particular parameter is needed, caching is entirely transparent.

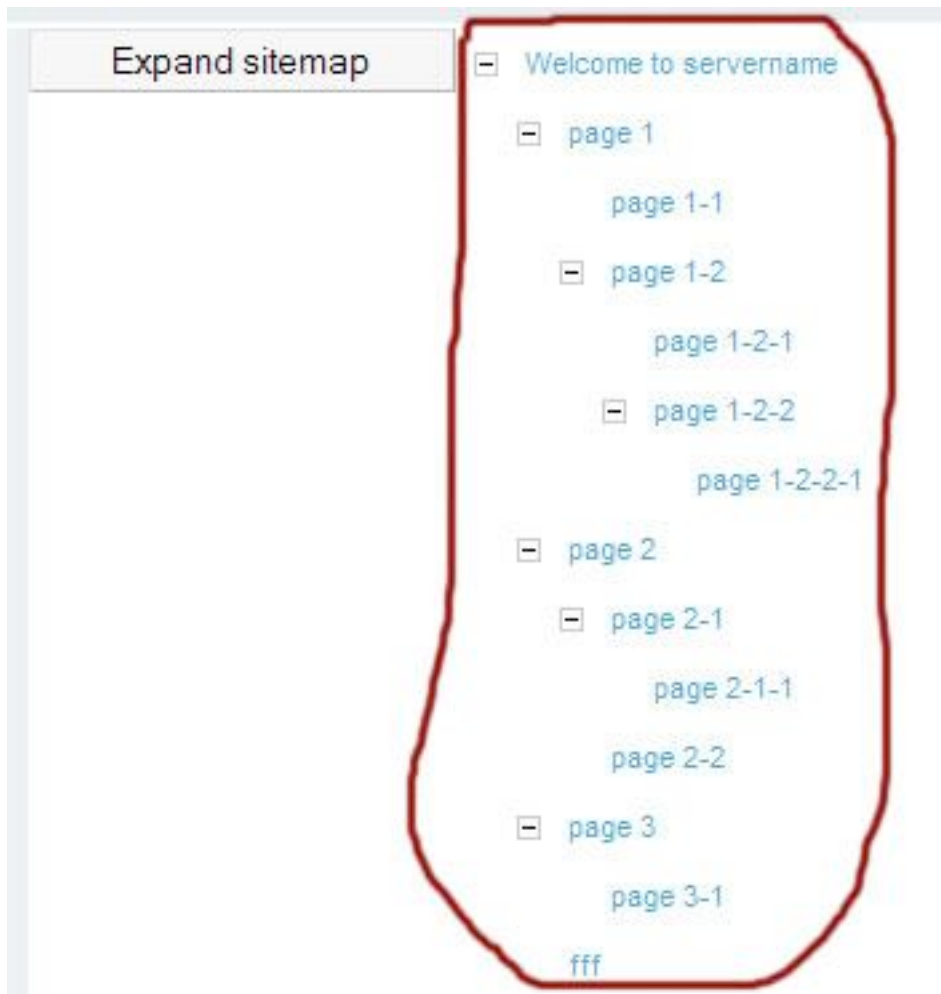
## 5.2 Sitemap

Also for creating a sitemap, Jahia 6 now provides a tag, which is simple to use.

The features for the sitemap are:

- display the sitemap as a dynamic tree, loading only what the user choose using AJAX
- display the entire sitemap tree without AJAX
- display a sitemap without unavailable pages
- display information about each page workflow state in edit mode
- maintain a sitemap.xml file to help indexation by search engines
- use cache when needed/possible
- display the sitemap using only one or a few tags

A created sitemap could look like this:



To create a sitemap, you simply call the following tag:

```
<ui:sitemap>
```

There are a number of attributes, where you can specify,

- whether the sitemap is dynamic (`ajax="true"`) or static.
- The start-page (`startPid="2"`), the maximum depth (`maxDepth="2"`) and whether the root page should be shown or not (`showRoot="false"`)
- If a description (from the content's description metadata field) should be displayed or not (`enableDescription="true"`)



The sitemap creates valid links to the pages, the hierarchy and the rank of a given page have to be consistent, and the workflow state should be displayed in edit mode.

Displaying a sitemap has to follow the rules applied to each and every page. Thus, we won't load any page that a user cannot read or that isn't published yet as well as areas with restricted access. Pages that aren't available in the current language won't be displayed.

#### 5.2.1 Caching

Concerning the dynamic sitemap, performance should not really be an issue, since an AJAX call is triggered only when expanding a given node.

For the static sitemap, there is a call, which will crawl through the whole page hierarchy. This is often the biggest weakness concerning performance, but we are trying to have the server-side logic as optimized as possible and behave well when being called by numerous users.

Nevertheless, since sitemap uses container lists to build its page hierarchy, cache can be used and only the first call might be heavy.

## 5.3 Navigation history

To integrate the display of your navigation history, you can use the following sample code:

```
<ul class="breadcrumbs">
  <c:forEach var="historyBean" items="${sessionScope['org.jahia.toolbar.history']}">
    <li><a href="${historyBean.url}">${historyBean.pageTitle}</a></li>
  </c:forEach>
</ul>
```

This uses a list of beans stored in the session and options defined in the jahia.properties file.

By default, the history beans are page-based, but you can use url-based beans to include queries by setting the historyUrlBased parameter to true.

You can also specify the size of the history queue by setting the historySize parameter to any integer, the default value being 10.

## 6 Search and query

Jahia has integrated a mix of search frameworks and uses a selected set of features from each in order to offer a powerful and flexible search solution.

### 6.1 Search features

This section gives a detailed overview about the search features integrated in Jahia and how Jahia 5 and 6 uses a mix of common frameworks.

#### 6.1.1 Detailed features

##### 6.1.1.1 Indexing

- No need for separate index data schema configuration as Jahia automatically does it according to container/field declarations (CND since Jahia 6 or declarations in templates in Jahia 5)
- Content is immediately indexed after editing or publishing
- Support Solr-based filtering and caching for faceted search with autowarming in order to have the new searcher (after content update) immediately performant
- Allow to set different analyzers per field in copies of the field (e.g. keyword analyzer for sortable/faceted fields)
- Using different score.boosts and text analyzers per field (keyword, standard, simple, whitespace, snowball or any custom analyzer) and language (different tokenizers, stemmers, stopwords)
- Jahia automatically re-configures the index schema on-the-fly if new modules/templates are deployed or updated on a running server not requiring a server restart
- Create and use local indexes per cluster node or access a shared index
- Ability to re-index a site on a Jahia server instance, without the need of restarting the server
- Configure valves for index processing chain with ability to plug-in own valves
- If required, Jahia can create and update a spellcheck dictionary from a site's index in configurable intervals
- Internal document repository is separately indexed and is used for document repository search. If a file is linked to content in a Jahia site, its extracted text and some metadata will also be indexed in

Jahia's index to automatically include linked files in a site search or container queries. The same applies to linked documents from externally mounted repositories via Jahia Unified Content Hub.

- Rich document parsing (text extraction) as supported by Apache Jackrabbit

#### 6.1.1.2 Search

- Structured search based on any number of constraints on any content and metadata field
- Supports defining a selector and multiple constraints and orderings as specified in the JSR-283 (Jahia 5 and 6 support a large subset of constraints, while 6.5 will support all)
- Search in one container list
- Search in multiple container lists with same name or alias
- in current site
- in a different site
- in multiple sites
- in multiple specified sections (descendant, childnode)
- Automatically search content using the current language or workflow state
- Allow search in content for other language(s) or workflow state(s) than current
- Sort by any number of fields (also relevance)
- Support for faceted search and browsing based on multiple values lists in field definition, category (sub)trees, explicit queries (e.g. date range queries)
- Ability to convert queries to either Lucene queries, so that our Lucene index is used or to SQL to use the Jahia database
- Unstructured search (fulltext)
- Supports the Lucene query syntax (optional and mandatory words, phrase search, without words, use of wildcards, ranges,...) (link ???)
- Search through all site content
- including linked documents from internal document repository

- including linked documents from mounted repositories
- including linked documents from external document repositories (Jahia 6EE)
- Search in internal document repository and/or external repositories (via Jahia's UCH)
- Search term highlighting in results
- Sort by relevance
- Parallel search in multiple sites
- Dynamically use same analyzer as used for indexing (also depending on searched language)
- Ability to configure returning one or multiple hits per page
- Ability to exclude fields from being fulltext searched by configuration
- Consider restricted content, workflow and timebased publishing rules to not return search hits to content, which cannot be accessed by the current user at the current time
- Specify limit for number of search results for better performance, to for instance display just the top ten related content objects
- Return total number of hits

#### 6.1.1.3 Easy template development

- Powerful JSP tags to remove the necessity for scriptlets
- Tag libraries for creation of simple and advanced search forms (site and document search) with support for unstructured query and metadata search
- Tag library for structured queries (container queries)
- Tag library for faceted search support (show hits per facet, display already selected facets in the path and allow to remove them again)
- Template developer can create different result snippets per found content type and for instance display the most relevant excerpt of a larger content object (with highlighted search terms) or display other data related to the found content object
- Offer URL to link to the page displaying the found content with rule based query parameter manipulation (e.g. container details view or correct page in pagination)

#### 6.1.1.4 Advanced features

- Use Solr-based Spellchecker, which is better supporting multiple terms per query in the „Did you mean“ support
- Automatically creates OpenSearch tags to be detectable by OpenSearch clients (like Firefox)

#### 6.1.1.5 Administration

- Start re-indexing of a site via the Jahia administration center
- Possibility of writing the created queries into the jahia log
- Luke can be used to view the index and make sample queries
- Configure settings in XML file for Jahia in cluster, for Lucene performance tuning, for Solr-based filter caching and autowarming
- Configure language specific index settings and analyzers, highlighting, spellchecking, index optimization policies via Compass XML
- Configure indexing policies (immediate or delayed indexing) per content type in an XML

#### 6.1.2 Used frameworks

Index/search in Jahia works out-of-the-box with just using the default configuration settings, but you can customize and tune settings in these configurations (mainly based on Compass, Lucene, Jackrabbit, Solr-like settings and some custom configurations).

Jahia has integrated multiple search frameworks and uses a selected set of features from each in order to offer a powerful and flexible search solution.

##### 6.1.2.1 Apache Lucene (integrated library)

- Jahia uses Lucene directly for indexing and searching Jahia's content
- Supports the Lucene query syntax (link ???)
- Parallel search in multiple indexes (multiple Jahia sites)
- Sort by relevance and sort by any number of fields
- Support local index per cluster node or shared index
- Luke can be used to view the index and make sample queries

#### 6.1.2.2 Compass (integrated library)

- Maintain a search index per virtual site
- Allow for using different analyzers per field and language and allow for overriding Jahia's default choice (according to field declaration) by Compass XML configuration files
- Supports dynamic re-configuring of settings without server restart
- Configurable flexible text analysis (support to plug-in Lucene based analyzers, filters, configuring stopwords,...)
- Configurable highlighting of searched terms in result snippets
- Creating and updating spellcheck dictionaries in configurable intervals
- Configurable index optimization policies

#### 6.1.2.3 Apache Jackrabbit (integrated library)

- As the document repository is based on Java Content Repository (JCR), we use its reference implementation Apache Jackrabbit and thus file documents are indexed and searched via Jackrabbit. In Jahia 5.x we still use Apache Slide. From Jahia 6.5 onwards Jackrabbit will be used also for Jahia content indexing, which will phase out the custom Jahia index and Compass integration.
- Rich document parsing (text extraction) and indexing
- Abstract query object model is used as basis for our tags to create container queries, which are then either mapped to Lucene queries or SQL queries (query Jahia content via JDBC)

#### 6.1.2.4 Apache Solr (integrated some classes)

- Filtering and caching for faceted search
- Allow queries to be converted to filters and cached
- Autowarming of recently used filters in the background and populating it to the new searcher
- Using Spellchecker for „Did you mean“ support

#### 6.1.2.5 OpenSearch (integrated library)

- Supports inbound and outbound OpenSearch calls
- Jahia automatically creates OpenSearch tags to be detectable by OpenSearch clients (like Firefox)

- GWT module for parallel searching in any OpenSearch provider

#### 6.1.2.6 EntropySoft (library can be integrated via Jahia Unified Content Hub)

- Allow to search in external repositories mounted in Jahia via the Universal Content Hub
- Transparently convert queries to EntropySoft query language

## 6.2 Search vs Query

Jahia offers two distinct tag libraries. The search tags and the query tags. Here is a detailed comparison between those tag libraries to let you find out, which one is better for a specific use case.

### 6.2.1 Comparison

	Query-tags	Search-tags
Main purpose	Retrieving homogeneous objects of one type (e.g. news items, event, etc.) including filtering and sorting by fields	Site and document repository search using full text queries and metadata search
Result objects	Containers (from Jahia 6.5 onwards any content node)	Hit object, which refers to containers, files, folders or pages
Automatic creation of HTML input form	No	Yes
Can do programmatic search without input form	Yes	Yes
Search any content object by fulltext search or constraints on metadata fields	It is possible to search through all containers of any type	Yes
Limit search to certain container definition	Yes	Possible by adding constraint in raw query (but not compatible in Jahia 6.5)
Limit search to multiple container definitions	Only when using same alias (or by adding constraint in fulltext search)	Possible by adding constraint in raw query (but not compatible in Jahia 6.5)
Ability to search in entire site	Yes	Yes
Ability to search in multiple or all sites	Yes	Yes
Ability to limit search to children or descendants of content objects	Yes	Yes, page or descendant page constraint are available by tag; container or container list constraints must be added in raw query



Automatically dereference linked files (from internal or external mounted document repositories) in content objects within the searched scope	Yes	Yes
Search for files in internal or external document repositories (if supported by connector)	No	Yes
Ability to search in current, specific and multiple languages	Yes	Yes
Ability to search in current, specific and multiple workflow states	Yes	Yes
Support of fulltext query syntax	Fulltext query syntax as specified in JSR-283	Lucene query syntax
Ability to build complex queries	Yes	Only when writing a raw query
Highlighting search term in search result excerpts	No (only possible by some custom scriptlets)	Yes
Set limit for number of search results	Yes	Yes
Sort by relevance	Yes	Yes
Sort by fields	Yes	No
Automatically consider access control and timebased publishing to not return hits, which cannot be access by current user at current time	Yes	Yes
Ability to configure returning one or multiple hits per page	No	Yes
Ability to automatically convert queries to run against database	Yes	No
Integrated with faceted search/browsing tags	Yes	No (not yet)
Ability to use backend caches (Lucene filters)	Yes	No
Integrated with frontend HTML cache	Yes	No
OpenSearch integration	No	Yes

## 6.3 Building Search forms

This section shows the usage of Jahia Search JSP Tag Library in building custom search forms for several use cases.

### 6.3.1 Simple search form

Here is a simple search template leaving out internationalization and layout for the sake of simplicity:

```
1: <%@ taglib prefix="s" uri="http://www.jahia.org/tags/search" %>
2:
3: <s:form method="get">
4:   <label for="searchTerm">Start searching:</label>
5:   <s:term id="searchTerm"/>
6:   <input type="submit" value="Search" title="Search"/>
7: </s:form>
```

This code snippet results in an HTML form with an input field and a submit button. All the form-related search tags are nested in a `<s:form/>` tag (lines 3 to 7).

`<s:term/>` tag (line 5) renders a simple entry field for the search term.

Line 6 contains standard HTML input element that renders form submit button.

### 6.3.2 Advanced site search with metadata

The following code is used to render an advanced form for searching site content (we omit resource bundles and layout stuff):

```
1: <s:form name="advancedSearchForm" method="get">
2:   <fieldset>
3:     <legend>Text search</legend>
4:     <label for="searchTerm">Search:</label>
5:     <s:termMatch selectionOptions="all_words,exact_phrase,any_word,as_is"/>
6:     <s:term id="searchTerm"/><br/>
7:     <s:termFields value="pages_content" display="false"/>
8:   </fieldset>
9:
10:  <fieldset>
11:    <legend>Author and date</legend>
12:    <label for="searchCreatedBy">Author:</label>
13:    <s:createdBy id="searchCreatedBy"/><br/>
14:
15:    <label for="searchCreated">Created:</label>
16:    <s:created id="searchCreated"/><br/>
17:
18:    <label for="searchLastModifiedBy">Last editor:</label>
19:    <s:lastModifiedBy id="searchLastModifiedBy"/><br/>
20:
21:    <label for="searchLastModified">Modified:</label>
22:    <s:lastModified id="searchLastModified"/>
23:  </fieldset>
24:
25:  <fieldset>
26:    <legend>More...</legend>
27:    <label for="searchSite">Site:</label>
28:    <s:site id="searchSite"/><br/>
29:
30:    <label for="searchLanguage">Language:</label>
```

```

31:      <s:language id="searchLanguage"/><br/>
32:
33:      <label for="searchPagePath">Page:</label>
34:      <s:pagePath id="searchPagePath"/><br/>
35:    </fieldset>
36:
37:    <input type="submit" name="search" value="Search"/>
38:  </s:form>

```

The code renders the following form as a result:

The first fieldset (lines 2 to 8) contains a hidden field, created by the `<s:termFields/>` tag with the value `pages_content`, which indicates that the fulltext search will take place in the site content. When metadata should be included in the search, then use `pages_all`. Contrary if the search should take place in the document repository you should use `documents_content` or `documents_all`.

Next fieldset (lines 10 to 23) is responsible for rendering metadata related entry fields, like author, creation date, last editor and last modification date.

The last fieldset (lines 25 to 35) displays additional criteria: site selection, content language selection and the page path (restricted search on a particular page or site section).

One note to the site selection. All search tags allow path-through attributes to be specified, i.e. CSS class, style, title, HTML element ID, JavaScript event handlers. For example the site selection tag can be transformed into a multiple selection combo box (allows selection of multiple sites for simultaneous search) by simply adding the `multiple` attribute:

```
<s:site multiple="multiple"/>
```

### 6.3.3 Advanced document search with metadata

The following code is used to render an advanced form for searching through the document repository (we omit resource bundles and layout stuff):

```
1: <s:form name="advancedSearchForm" method="get">
2:   <fieldset>
3:     <legend>Text search</legend>
4:     <label for="searchTerm">Search:</label>
5:     <s:termMatch selectionOptions="all_words,exact_phrase,any_word,as_is"/>
6:     <s:term id="searchTerm"/><br/>
7:     <s:termFields value="documents_content" display="false"/>
8:   </fieldset>
9:
10:  <fieldset>
11:    <legend>Author and date</legend>
12:    <label for="searchCreatedBy">Author:</label>
13:    <s:createdBy id="searchCreatedBy"/><br/>
14:
15:    <label for="searchCreated">Created:</label>
16:    <s:created id="searchCreated"/><br/>
17:
18:    <label for="searchLastModifiedBy">Last editor:</label>
19:    <s:lastModifiedBy id="searchLastModifiedBy"/><br/>
20:
21:    <label for="searchLastModified">Modified:</label>
22:    <s:lastModified id="searchLastModified"/>
23:  </fieldset>
24:
25:  <fieldset>
26:    <legend>More...</legend>
27:    <label for="searchSite">Site:</label>
28:    <s:site id="searchSite"/><br/>
29:
30:    <label for="searchFileType">File type:</label>
31:    <s:fileType id="searchFileType"/><br/>
32:    <s:documentType value="jnt:file" display="false"/>
33:  </fieldset>
34:
35:  <input type="submit" name="search" value="Search"/>
36: </s:form>
```

Following form is displayed as a result:

TEXT SEARCH

Search: all these words

AUTHOR AND DATE

Author:

Created: anytime

Last editor:

Modified: anytime

MORE...

Site: ACME Site

File type: any

Search

any

Adobe Acrobat PDF

Microsoft Word

Rich Text Format

Microsoft Excel

Microsoft PowerPoint

OpenOffice

Archive

The code is similar to the advanced search for site content search. The first fieldset contains in the line 7 the documents\_content value instead of the pages\_content to indicate the search in document content.

And last fieldset (lines 25 to 33) contains additionally the file type selection control and a hidden field, generated by the <s:documentType/> tag, which scopes the search to results with node type jnt:file.

#### 6.3.4 Display search results

For displaying results of the search the <s:results/> and <s:resultIterator/> pair of tags can be used.

The <s:results/> tag parses the search parameters from the current request, executes the search and exposes the results list (java.util.List<org.jahia.engines.search.Hit<?>>) and the total hit count into the page scoped variables with names hits and count (these names can be overridden with the tag attributes: see documentation for Jahia Search tag Library for more details).

The second tag - <s:resultIterator/> - is used to iterate over the exposed list of hits as shown in the following example:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@ taglib prefix="s" uri="http://www.jahia.org/tags/search" %>

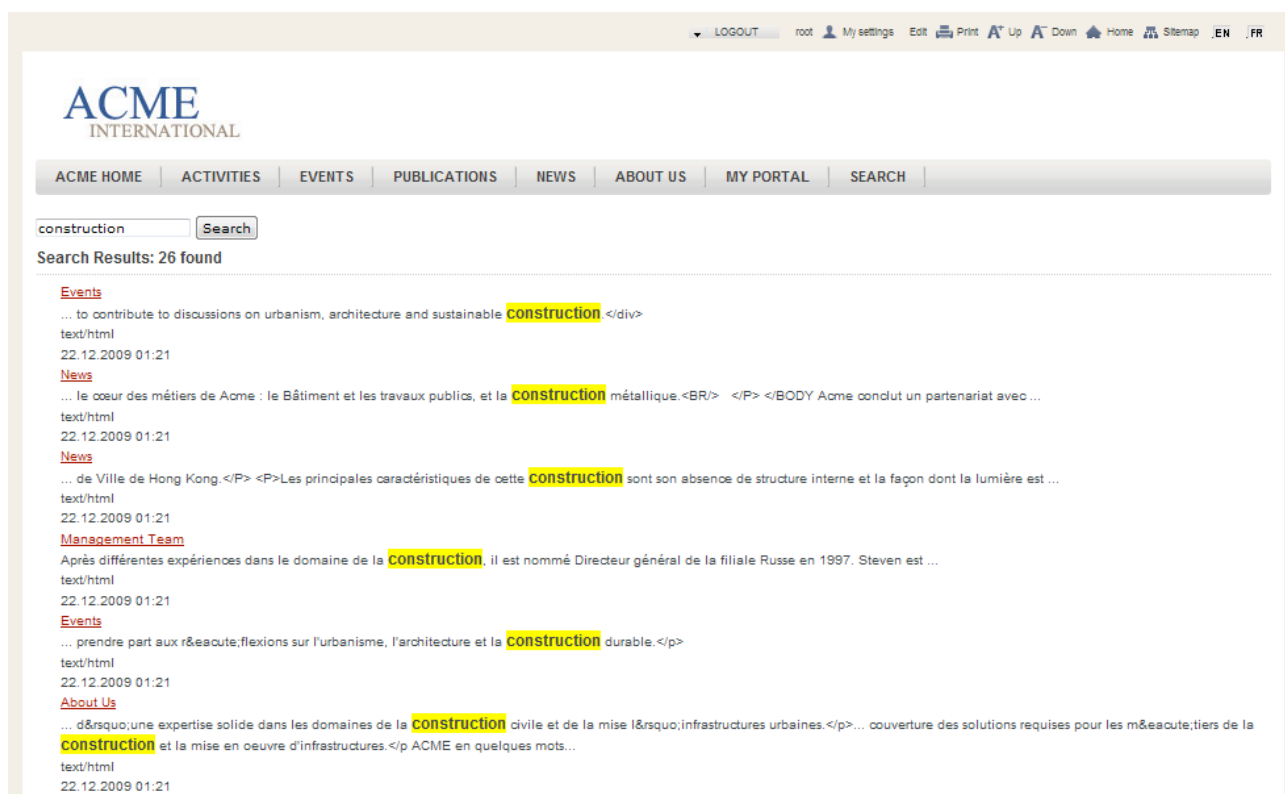
<s:results>
  <c:if test="${count > 0}">
    <h4><fmt:message key="search.results.found"><fmt:param
value="${count}"/></fmt:message></h4>
    <ul>
      <s:resultIterator>
        <li>
          <div><a href="${hit.link}">${fn:escapeXml(hit.title)}</a></div>
          <div>${hit.excerpt}</div>
          <div>${hit.contentType}</div>
        </li>
      </s:resultIterator>
    </ul>
  </c:if>
</s:results>
```

```

        <div><fmt:formatDate value="{hit.lastModified}" pattern="dd.MM.yyyy HH:mm"/></div>
    </li>
</s:resultIterator>
</ul>
</c:if>
<c:if test="{count == 0}">
    <h4><fmt:message key="search.results.no.results"/></h4>
</c:if>
</s:results>

```

The corresponding screenshot is:



## 6.4 Advanced search customization

### 6.4.1 Configure local specific indexing

Jahia is delivered out-of-the-box with no locale specific indexing settings. As Lucene allows for plugging in different locale specific analysis (tokenizing, stemming, stopwords, filtering,...), Jahia 5 and 6 allows to do the same by using the dynamic capabilities of the Compass framework (from 6.5 onwards this configuration will completely be replaced by Jackrabbit's configuration files). In the Administration center you can manage the languages of a Jahia site. This way when you capture content for a specific language or make a search in this language, we will check, whether you have activated language specific analysis in the Compass configuration.

Jahia is delivered with some commented example settings for different languages. For instance adding search support for Thai in Jahia can be added by the following configuration changes:

1. Open `webapps\jahia\WEB-INF\etc\spring\applicationcontext-compass.xml` and you will see that we have some commented lines showing language dependend example configurations for "compass.engine.analyzer.default\_\*" and "compass.engine.analyzer.search\_\*". You can do it for Thai simply by adding or uncommenting the following lines:

```
<prop
key="compass.engine.analyzer.default_th.factory">org.compass.core.lucene.engine.analyzer.DefaultLuceneAnalyzerFactory</prop>
<prop
key="compass.engine.analyzer.default_th.type">org.apache.lucene.analysis.th.ThaiAnalyzer</prop>
>

<prop
key="compass.engine.analyzer.search_th.factory">org.compass.core.lucene.engine.analyzer.DefaultLuceneAnalyzerFactory</prop>
<prop
key="compass.engine.analyzer.search_th.type">org.apache.lucene.analysis.th.ThaiAnalyzer</prop>
>
```

2. Open `webapps\jahia\WEB-INF\classes\org\jahia\services\search\compass\jahiaresource.cpm.xml` and you will again see that there are several commented lines with language dependend examples, which are not activated on default. If you want to activate Thai indexing and searching, you need to uncomment or add these lines:

```
<resource alias="jahiaSearcher_th" extends="jahiaFields" sub-index="jahia"
analyzer="search_th" spell-check="exclude"/>

<resource alias="jahiaIndexer_th" extends="jahiaFields" sub-index="jahia"
analyzer="default_th" spell-check="exclude"/>

<resource alias="jahiaHighlighter_th" extends="jahiaIndexer_th" sub-index="jahia" spell-check="exclude"/>
```

3. After modifying the above files, you need to restart Jahia and if you have already captured Thai content, you need to re-index the site. This can be done in Administration -> Site settings (tab) -> Manage search engine.

Wait until content is re-indexed and then you should be able to search in Thai documents and content. For other languages, which cannot work with the default analyzer, you will need to add similar configuration, but exchange the language code (like `_th` for Thai) and use either Lucene packaged analyzers for that language or if not available on default you may have to check if there are available Lucene extension analyzers for that language.

## 6.4.2 "Did you mean" - spellchecker suggestion facility

From Jahia 6.1 onwards Jahia's search result JSP template and tags allow enabling the so called "Did you mean" feature. In this case an eventual typing mistake can be automatically corrected by the search engine based on the language-dependent spell checker dictionary, built for the current virtual site.

### 6.4.2.1 Configuration

The auto-suggestion feature can be activated and configured in the Compass configuration.

#### 6.4.2.1.1 applicationcontext-compass.xml

Open the file: webapps\jahia\WEB-INF\etc\spring\applicationcontext-compass.xml.

The parameters in the following section are used for the spellcheck component

```
<prop key="compass.engine.spellcheck.enable">true</prop>
<prop key="compass.engine.spellcheck.schedule">true</prop>
<prop key="compass.engine.spellcheck.scheduleInterval">10</prop>
<prop key="compass.engine.spellcheck.numberOfSuggestions">1</prop>
<prop key="compass.engine.spellcheck.accuracy">0.5</prop>
<prop
key="compass.engine.spellcheck.class">org.apache.lucene.search.spell.CompositeLucene
SpellCheckManager</prop>
```

More information about the activation, scheduling and other options can be found

here: <http://www.compass-project.org/docs/2.1.4/reference/html/core-searchengine.html#core-searchengine-spellcheck>

With compass.engine.spellcheck.numberOfSuggestions you can set how many possible suggestions should be returned.

You can fine tune the compass.engine.spellcheck.accuracy, to get a better quality of suggestions.

For the compass.engine.spellcheck.class parameter you should use org.apache.lucene.search.spell.CompositeLuceneSpellCheckManager which is a Jahia specific implementation integrated with the Jahia CMS.

#### 6.4.2.1.2 jahiaresource.cpm.xml

In the dictionary Jahia stores the language along with terms. This way for example when someone searches on the English version of the site, he will not be presented with suggested words, which are only used in the French version of the site.

The Lucene field where all terms searchable by a fulltext query is "jahia.all". The terms in this field most probably use stemming, so this field is not really suitable to create a dictionary from it. Furthermore terms of all languages will be saved in this field.

Jahia therefore also creates a field suffixed with the language code ( "jahia.all\_<language-code>" ) so that the spellchecker dictionary can keep the languages apart. By default all other Lucene fields will not be checked when creating the spellcheck dictionary, but if specific fields need to be included in the dictionary you can configure it in jahiaresource.cpm.xml with the use of the spell-check attribute .

The languages English, French and German are configured on default in webapps\jahia\WEB-INF\classes\org\jahia\services\search\compass\jahiaresource.cpm.xml



```
<resource-property name="jahia.all_en" analyzer="simple" store="yes" spell-check="include"/>
<resource-property name="jahia.all_fr" analyzer="simple" store="yes" spell-check="include"/>
<resource-property name="jahia.all_de" analyzer="simple" store="yes" spell-
check="include"/>
```

If you use other languages and want to create spellcheck dictionaries using the content in these languages you need to add additional lines exchanging the <language-code> placeholder with the language code of the used language:

```
<resource-property name="jahia.all_<language-code>" analyzer="simple" store="yes" spell-
check="include"/>
```

#### 6.4.2.2 Usage in template

The following code example shows the implementation we use in web-templates\src\main\webapp\areas\searchResults.jsp of our ACME demo web-templates

```
<s:results>
  ...
  <pg:pager maxPageItems="${itemsPerPage}" url="${jahia.page.url}"
export="currentPageNumber=pageNumber">
  ...
  <s:suggestedQuery var="suggestedQueries" hitsVar="suggestedQueryHits"
countVar="suggestedQueryHitsCount">
    <br/>
    <c:if test="${suggestedQueryHitsCount > count}">
      <div class="didyoumean">
        <fmt:message key="search.results.did.you.mean"/>
        <a href="${suggestedQueries[0].searchUrl}">
          <c:forEach items="${suggestedQueries[0].cleanQueryTerms}"
var="queryTerm">
            <c:if test="${queryTerm.changed}"><b></c:if>
              ${queryTerm.term}
            <c:if test="${queryTerm.changed}"></b></c:if>
          </c:forEach>
        </a>
        <c:choose>
          <c:when test="${suggestedQueryHitsCount > 2}">
            <c:set var="displayNoOfSuggestedHits" value="2"/>
          </c:when>
          <c:otherwise>
            <c:set var="displayNoOfSuggestedHits"
value="${suggestedQueryHitsCount}"/>
          </c:otherwise>
        </c:choose>
        <c:if test="${displayNoOfSuggestedHits > 0}">
          ( <fmt:message key="search.results.number.displayed">
            <fmt:param value="${displayNoOfSuggestedHits}"/>
            <fmt:param
value="${suggestedQueryHitsCount}"/>
          </fmt:message>
        <br/><br/>
        <s:resultIterator var="suggestedHit" begin="0"
```

```
end="${displayNoOfSuggestedHits - 1}">
    <li>
        <h4><a class=<c:choose><c:when
test="${suggestedHit.typeFile}">"${suggestedHit.iconType}"</c:when><c:otherwise>"jah
iapage"</c:otherwise></c:choose>
href="${suggestedHit.link}">${fn:escapeXml(suggestedHit.title)}</a></h4>
        <c:if test="${!empty suggestedHit.referencedHit}"><div
class="resultslistReference">(&nbsp;<fmt:message
key="search.results.referenced.on"/>&nbsp;<a class="jahiapage"
href="${suggestedHit.referencedHit.link}">${fn:escapeXml(suggestedHit.referencedHit.
title)}</a>&nbsp;</div></c:if>

        <div class="resultslistDesc">${suggestedHit.summary}</div>
        <div
class="resultslistFileType">${suggestedHit.contentType}</div>
        <div class="resultslistDateModified"><fmt:formatDate
value="${suggestedHit.lastModified}"

        pattern="dd.MM.yyyy HH:mm"/></div>
        <c:if test="${suggestedHit.typeFile}">
            <div class="resultslistSize">${suggestedHit.sizeKb}k</div>
        </c:if>
    </li>
</s:resultIterator>
</c:if>
</div>
</c:if>
<br/>
</s:suggestedQuery>
...
</pg:pager>
</s:results>
```

The main tag is:

```
<s:suggestedQuery var="suggestedQueries" hitsVar="suggestedQueryHits"
countVar="suggestedQueryHitsCount">
```

All three attributes are names for variables attributes which will be set into the page context.

- `var="<any-name>"` ... will receive the list of suggested queries as beans of type `org.jahia.data.search.SuggestedQuery`
- `hitsVar="<any-name>"` ... will hold the search results by using the first suggested query for immediate display
- `countVar="<any-name>"` ... will hold the total number of hits by using the first suggested query

As we have configured in Compass that only one suggestion is returned, we are only expecting one search suggestion, which we access with `${suggestedQueries[0]}`. The bean has the following methods:

- `${suggestedQueries[0].searchUrl}` ... the URL to be used for executing a search with the suggestion
- `${suggestedQueries[0].cleanQueryTerms}` ... a list of `QueryTerm` beans representing each word of the query cleaned of internally added query modifications
- `${suggestedQueries[0].queryTerms}` ... a list of `QueryTerm` beans representing each word of the query including internally added query modifications

- `${suggestedQueries[0] .suggestedQuery} ..` the suggested query with internally added query modifications
- `${suggestedQueries[0] .oldQuery} ...` the old query with internally added query modifications

The QueryTerm bean has the following methods:


- `${queryTerm.changed}` ... a boolean which is true if the user typed term has been changed or false if it is unchanged
- `${queryTerm.term}` ... the suggested or unchanged term

This way it is possible to highlight to the user, which terms have been changed, because they may have been misspelled.

### 6.4.2.3 Display

Now, when searching for a term with a spelling mistake the search engine tries to find the best suggestion and display the "Did you mean" term with a link to refine the search:

---



[ACME HOME](#) | [ACTIVITIES](#) | [EVENTS](#) | [PUBLICATIONS](#) | [NEWS](#) | [ABOUT US](#) | [MY PORTAL](#) | [SEARCH](#)

Did you mean: [lorem](#). Top 2 results shown


- [News Archives \(2007\)](#)  
**Lorem** ipsum dolor sit amet 2007  
text/html 06.01.2010 17:05
- [2008 Publications](#)  
Marketing Department ACME **Lorem** ipsum dolor sit amet  
text/html 06.01.2010 17:05

---

Results for: **loram**  
Search Results: 1 found

- [News Archives \(2007\)](#)  
2b219b1c-8080-447f-bbcc-1389313862da **Loram** ipsum dolor sit amet 2007 2007-06-18T00:00:00.000+02:00  
text/html 06.01.2010 17:05

---


© Copyright 2002-2009 - ACME International Corp.

### 6.4.2.4 Migration

If you migrate from a previous version to Jahia 6.1 you need to manually delete all files and folders under `webapps\jahia\WEB-INF\var\search-indexes\`. After restarting you need to re-index each site from within Jahia's Administration center: Site settings -> Manage search engine.

### 6.4.3 Autocompletion - suggested typing

While suggested typing will be completely integrated from Jahia 6.5 onwards, Jahia 6.1 also already provides a simple way of retrieving terms matching the already typed characters from the index. There is a servlet, which can be called via AJAX calls with the following URL:

```
http://<your-domain>/<your-jahia-context>/ajaxaction/SearchAutoFullFill?searchString=<already-typed-
characters>&searchField=<fieldname-in-lucene-index>&params=/site/<site-key>
```

Regarding the <fieldname-in-lucene-index> you can use Luke to look at the created index to see which Lucene fields are created by Jahia. It is very important that the fields are not using any stemming analyzer, because that one will not store the full terms, so the suggestion will not really be meaningful. As mentioned above in the "Did you mean" section you could use jahia.all\_<language-code> to get all unstemmed terms for a language.

The autocompletion however makes most sense for terms in a limited scope. Categories, or tags, or if you for example have a field holding relevant city names, then you could for such fields use autosuggestion in input forms.

The result of the above call will be a simple comma-separated list of suggested terms for autocompletion. In Jahia 6.5 we will send back JSON formatted results, which are a better standard and allow for sending more information along.

There are already several jQuery components available which offer auto-completion capabilities.

### 6.4.4 Indexing policies

Jahia allows to configure several indexing policies according to rules. This is done in webapp\WEB-INF\etc\spring\applicationcontext-indexationpolicy.xml .

For instance it is possible to set the certain content should never be indexed or should be indexed delayed or at specific times. In Jahia 5 by default most content is indexed with a delay, which is a problem when you used container queries based on Lucene. For all content objects queried like this, one needed immediate synchronized indexing, which could be set in the mentioned applicationcontext-indexationpolicy.xml .

In Jahia 6 however the synchronized indexing is now the default behaviour. Also the <query:> tags use the Lucene index on default, while in Jahia 5 queries used the database on default. The only exception is file extractions, which can take a lot of resources. That is why in Jahia 6 file content is still indexed delayed.

You can take a look at the provided webapp\WEB-INF\etc\spring\applicationcontext-indexationpolicy.xml , because it is full of examples, how you can set different indexing policies (e.g. by content type, by page path,...). You can also write your own implementations of

org.jahia.services.search.indexingscheduler.RuleCondition if the provided conditions do not meet your requirements.

#### 6.4.5 Cluster setup

Chapter 2.9.7 of the [Configuration and fine tuning guide](#) available through the extranet, explains how to setup search indexes in cluster.

If you decide to use an index shared over an NFS drive, then take care to make the following change mentioned in webapp\WEB-INF\etc\spring\applicationcontext-indexationpolicy.xml :

```
<!--
When storing the index on an NFS file system, we recommend to comment the following line and activate
the next two settings, as autocommit should be deactivated
and a ExpirationTimeDeletionPolicy deletion policy set.
-->
<prop key="org.apache.lucene.indexerAutoCommit">1</prop> <!-- 1 for true -->
<!--
<prop
key="org.apache.lucene.indexDeletionPolicy">org.apache.lucene.index.ExpirationTimeDeletionPolicy</prop>
<prop key="org.apache.lucene.expirationTimeForDeletion">60</prop>
<prop key="org.apache.lucene.indexerAutoCommit">0</prop>
-->
```

#### 6.4.6 Search result highlighting

For search term highlighting we use the Compass framework, which is configured in WEB-INF\etc\spring\applicationcontext-compass.xml . If you want to change the behaviour of highlighting, then you can get more information here: <http://www.compass-project.org/docs/2.1.4/reference/html/core-settings.html#core-configuration-searchenginehighlighters>

#### 6.4.7 Cached Lucene filters / faceting

Jahia uses classes from Apache Solr in order to provide faceted queries. What the feature does is create Lucene filters for example for all documents modified in the last months, year, etc. or for all documents per category. The BitSets of these queries will be stored in cache. This way when the user specifies does

query, you can very display in a very performant way, how hits would be limited if he would have used certain categories or other facets.

As soon as new or updated content is indexed, Jahia will use this Solr technology to re-run all the cached filters before exposing the new IndexReader to the ongoing search requests. This way the performance will stay fast and the hits per facet will always be consistent and up-to-date even after re-indexing.

## 7 Web templates quick customization

The new web-templates package has been build respecting the XHTML / CSS standards allowing a quick revamping by competent web designers. Jahia EE v6.1 also introduced some new mechanism to enhance the webmaster (editors) experience and control upon their pages layout: skins and themes.

### 7.1 CSS overview

There are 4 important stylesheets webdesigners should consider

- web.css is the main stylesheet where 95% of styles are declared
- files.css is used to define styles applied on documents lists, display the right icon for instance
- print.css defines the rendering when users call the print function. The role of this stylesheet is to feet A4 printing width, hiding some unnecessary content areas, making a clear and readable printed page.
- edit.css is used to deal with special layout tricks when editors brows pages in edit-mode

## 8 Cache Configuration

### 8.1 Inside Jahia caches

There are three levels of caches inside JAHIA:

- Database Level handled by hibernate framework
- Service Level handled by JAHIA
- HTML Level handled by JAHIA

For the DB level refers to the hibernate documentation: [http://www.hibernate.org/hib\\_docs/v3/reference/en-US/html/performance.html](http://www.hibernate.org/hib_docs/v3/reference/en-US/html/performance.html)

At the service level JAHIA will cache as many objects as possible using mostly (by default) its own cache implementation based on the reference API from JAVA:

<http://java.sun.com/javase/6/docs/api/java/lang/ref/package-summary.html>

At the HTML level JAHIA will cache two different things, the first one is called the skeleton of the page (it is the HTML code of the page with some space reserved for dynamic insertion of piece of content) and the other parts is this same piece of HTML contents needed to fill in the blank inside the skeleton.

This allows us to only regenerate some specific part of the page when needed.

When a page needs refreshing only the first user hitting the page will regenerate it (for all the users sharing the same view as him based on ACL) while the others will received the content in cache. Once it is updated everybody will see the latest version.



## 8.2 Cache configuration

To configure the hibernate cache, you will have to configure it through the Spring framework configuration files, more specifically the file `"WEB-INF/etc/spring/applicationcontext-hibernate.xml"`

The service and HTML cache are configurable inside the file `"WEB-INF/etc/spring/applicationcontext-service.xml"`

```
<bean id="org.jahia.services.cache.CacheProvider"
class="org.jahia.services.cache.clusterservice.batch.BatchingClusterServiceCacheProvider">
    <property name="clusterService">
        <ref bean="clusterService" />
    </property>
</bean>

<bean id="JahiaCacheService" parent="proxyTemplate">
    <property name="target">
        <bean class="org.jahia.services.cache.CacheFactory" parent="jahiaServiceTemplate"
factory-method="getInstance" >
            <property name="cacheProviders">
                <map>
                    <entry>
                        <key><value>DEFAULT_CACHE</value></key>
                        <ref bean="org.jahia.services.cache.CacheProvider" />
                    </entry>
                    <entry>
                        <key><value>EH_CACHE</value></key>
                        <bean class="org.jahia.services.cache.ehcache.EhCacheProvider">
                            <!-- This property allows to fix a limit for cache entries
dependencies management,
                            if an entry have more than this value of dependencies then
                            when we flush this entry we will flush the whole cache-->
                            <property
name="groupsSizeLimit"><value>100</value></property>
                        </bean>
                    </entry>
                </map>
            </property>
            <property name="cacheProviderForCache">
                <map>
                    <entry>
                        <key><value>SkeletonCache</value></key>
                        <value>EH_CACHE</value>
                    </entry>
                    <entry>
                        <key><value>ContainerHTMLCache</value></key>
                        <value>EH_CACHE</value>
                    </entry>
                    <entry>
                        <key><value>LockAlreadyAcquiredMap</value></key>
                        <value>EH_CACHE</value>
                    </entry>
                    <entry>
                        <key><value>LockPrerequisitesResultMap</value></key>
                        <value>EH_CACHE</value>
                    </entry>
                    <entry>
                        <key><value>WebdavCache</value></key>
                        <value>EH_CACHE</value>
                    </entry>
                </map>
            </property>
        </bean>
    </property>
</bean>
```

```
        </property>
    </bean>
</property>
</bean>
```

We see that inside JAHIA we can define several cache providers and specify which providers we use for each internal cache.

The default provider is not configurable as it is based on reference cache it will try to maximize usage of memory and rely on the garbage collector to flush caches when memory is needed for computing.

For the EHCACHE provider the configuration file is `"WEB-INF/classes/ehcache-jahia.xml"` to have more information on EHCACHE configuration: <http://ehcache.sourceforge.net/documentation/>

All the HTML Level caches used an expiration time for all elements. This default expiration is defined inside the `"WEB-INF/etc/config/jahia.properties"` file:

- the following value is the default expiration date in seconds of container cache entries
- (for example 2592000secs is 30 days or 604800secs is 7 days)
- here the value is 4 hours

```
containerCacheDefaultExpirationDelay = 14400
```

## 8.3 HTML Fragment Cache

### 8.3.1 What is the HTML Fragmented cache?

Jahia has introduced a new HTML fragmented cache system acting at the container-level in Jahia. The HTML fragmented cache replaces the HTML full page cache. Containers are cached (i.e. fields, data) separately than the rest of the page. The main benefits of this new implementation are:

- Cache entries are shared between users with the same ACL
- Absolute containers can share the same cache, on different pages
- A page is rarely completely regenerated, only new or modified containers are.

The HTML of the page is cached under name of skeleton (this is the whole HTML page with some holes defined to be filled at runtime with piece of HTML coming from the containers cache).

This skeleton is flushed when the changes made on the page affect the number or the order of the containers inside this page. For example if my page contains two containers like `mainContent(title,body)`, if I add one more container on my list Jahia will flush the skeleton so that the newly regenerated one is aware that there is three containers to aggregate now. All the other containers will not be regenerated, as they are not impacted by these changes.

### 8.3.2 How to integrate it?

#### 8.3.2.1 Basic functioning

By default, the standard `<content:container>` tag caches all the output and returns the cached HTML when it is reused. The container cache can be disabled for one container, or for all containers of a list. If the container needs to be reused on multiple places with different rendering, the attribute `cacheKey` can be used to set a unique key for each different rendering. A container that is not entirely cached can still cache some of its parts by using the `<content:container-cache>` tag

Container cache is invalidated when one of its fields or sub-container are modified. Other containers in the page are still cached.

#### 8.3.2.2 Disabling the cache

For the container cache to be disabled for all the container in a containerlist, we should set the `cache` attribute to "off"

```
<content:containerList name="webappContainer" id="webappsContainerList"
parentContainerName="boxContainer">
<content:container cache="off">
```

```
...
</content:container>
</content:containerList">
```

### 8.3.2.3 Multiple rendering

Sometimes a container is used on different spots in the same page, and rendered in different ways. For example:

```
<content:container id="qaContainer">
<a href="#"<bean:write name='qaContainer' property='id'/"
class="bold"><content:textField name="question"/></a>
</content:container>
...
<content:container id="qaContainer">
<a name="<bean:write name='qaContainer' property='id'/"
class="bold"><content:textField name="question"/></a><br/>
<content:textField name="answer"/>
<br/>&nbsp;
</content:container>
```

With the container cache activated, the second container tag will use the cache generated by the first one. This can be fixed by adding the `cacheKey` attribute:

```
<content:container id="qaContainer" cacheKey="question">
<a href="#"<bean:write name='qaContainer' property='id'/"
class="bold"><content:textField name="question"/></a>
</content:container>
...
<content:container id="qaContainer" cacheKey="answers">
<a name="<bean:write name='qaContainer' property='id'/"
class="bold"><content:textField name="question"/></a><br/>
<content:textField name="answer"/>
<br/>&nbsp;
</content:container>
```

`cacheKeyName`, `cacheKeyProperty` and `cacheKeyScope` attributes can be used to get the actual value from a bean.

The `cacheKey` parameter could be defined by expression language:

```
<content:containerList name="cachecontent" id="cachecontentList">
<content:container id="cachecontentContainer" cacheKey="{param.selectedColor}"><div
style="background-color:{param.selectedColor}">
<content:textField name="mainContentTitle" diffActive="true"/>
<content:bigTextField name="mainContentBody" diffActive="true"/>
</div>
</content:container>
</content:containerList>
<form method="get">
<select name="selectedColor">
```

```
<option value="red">Red</option>
<option value="green">Green</option>
</select>
<input type="submit"/>
</form>
```

### 8.3.2.4 Partial container caching

As it has been said before, the container cache can be disabled at the container level, but it is still possible to cache parts of the container. Use the following in order to take profit of this feature:

```
<content:container id="topMenuContainer" cache="off">
<content:pageField valueId="topLink" name="topLink" id="topLinkField" />
<content:container-cache>
... display ...
</content:container-cache>
</content:container>
```

The container-cache tag can also use the cacheKey attributes.

### 8.3.2.5 Expiration

If your container need to be regenerated more often that the default value (defined in jahia.properties) then you can add an expiration parameter to your tags:

```
<content:containerList name="cachecontent" id="cachecontentList">
<content:container id="cachecontentContainer" cacheKey="expired" expiration="30">
<jsp:useBean id="now" class="java.util.Date" />
<fmt:formatDate value="{now}" dateStyle="full" type="both"/><br/>
<content:textField name="mainContentTitle" diffActive="true"/>
<content:bigTextField name="mainContentBody" diffActive="true"/>
</content:container>
</content:containerList>
```

The expiration is expressed in seconds.

### 8.3.2.6 Cache non JAHIA content

Imagine you need to access webservices or other channel of data to display information on your page. This is not Jahia content but you do not want to avoid caching the page due to that just to display dynamic data that are changing every 5 minutes for example.

Here an example displaying a RSS, refreshed every 5 minutes:

```
<content:cache cacheKey="rssOutput" expiration="300">
<c:import var="xml" url="http://www.theserverside.com/rss/theserverside-rss2.xml"/>
<x:parse var="rss" xml="{xml}" />
<ul>
  <x:forEach select="$rss//channel/item" var="n">
    <li>
      <a href="{x:out select="$n/link"/}">
        <x:out select="$n/title"/>
      </a><br/>
      <span><x:out select="$n/description" escapeXml="false"/></span>
    </li>
  </x:forEach>
</ul>
<span> Rss updated on: <fmt:formatDate value="{now}" dateStyle="full"
type="both"/><br/></span>
</content:cache>
```

For other examples please explore the Jahia test templates.

## 9 Event Handling and Catching

Jahia allows the template developer to catch events and customize the way Jahia should behave in a particular situation. There is nearly an event fired for each action a user can perform offering lots of freedom. You will find on this page all the information you need to know about event handling and catching.

Below you will find the list of all available events you can easily catch.

Event Name	Description
<code>addContainerEngineAfterInit</code>	Fired once the user has clicked on the "Add" action but the edition window has still not been displayed to the user
<code>addContainerEngineAfterSave</code>	Fired once the user has clicked on the "OK" or "Apply" button of the engine and the container has been added to the database
<code>addContainerEngineBeforeSave</code>	Fired once the user has clicked on the "OK" or "Apply" button of the engine but the container has not yet been saved into the database
<code>afterGroupActivation</code>	
<code>afterServicesLoad</code>	Fired once all the Jahia services are available
<code>aggregatedContentActivation</code>	
<code>aggregatedContentObjectCreated</code>	
<code>aggregatedEventsFlush</code>	
<code>aggregatedObjectChanged</code>	
<code>beforeContainerActivation</code>	
<code>beforeFieldActivation</code>	
<code>beforeServicesLoad</code>	Fired upon Jahia loading, just before loading the Jahia services
<code>beforeStagingContentIsDeleted</code>	

categoryUpdated	Fired once a Category has been updated (database has been updated) and the caches have been updated.
containerAdded	Fired once a Container has been added (database has been updated) and the caches have been updated.
containerDeleted	Fired once a Container has been deleted (database has been updated) and the caches have been updated.
containerListPropertiesSet	Fired once the properties of a list have been set
containerUpdated	Fired once a Container has been updated (database has been updated) and the caches have been updated.
containerValidation	
contentActivation	
contentObjectCreated	
contentObjectDelete	
contentObjectRestoreVersion	
contentObjectUndoStaging	
contentObjectUpdated	
fieldAdded	
fieldDeleted	
fieldUpdated	
fileManagerAclChanged	
groupAdded	Fired once a group of users has been added
groupDeleted	Fired once a group of users has been deleted
groupUpdated	Fired once a group of users has been updated
metadataEngineAfterInit	Fired once the user has clicked on the Metadata tab, but the



	screen has yet not been displayed to the user
metadataEngineAfterSave	Fired once the user has clicked on the "OK" or "Apply" button, and the modifications have been saved into the database
metadataEngineBeforeSave	Fired once the user has clicked on the "OK" or "Apply" button, but the modifications have not yet been saved into the database
objectChanged	
pageAccepted	Fired once a page has been accepted in the workflow process
pageAdded	Fired once a page has been added
pageDeleted	Fired once a page has been deleted
pageLoaded	Fired once a page has been loaded
pagePropertiesSet	Fired once the properties of a page have been set
pageRejected	Fired once a page has been rejected in the workflow process
rightsSet	Fired once the rights on a content object have been set
siteAdded	Fired once a site has been added / created
siteDeleted	Fired once a site has been deleted
templateAdded	Fired once a template has been added
templateDeleted	Fired once a template has been deleted
templateUpdated	Fired once a template has been updated
timeBasedPublishingEvent	Fired once there is a change of state regarding time based publishing
updateContainerEngineAfterInit	Fired once the user has clicked on the "update" action but the edition window has still not been displayed to the user
updateContainerEngineBeforeSave	Fired once the user has clicked on the "OK" or "Apply" button of the engine but the container has not yet been saved into the database

userAdded	Fired once a user has been added / created
userDeleted	Fired once a user has been deleted
userLoggedIn	Fired once a user has logged in into Jahia
userLoggedOut	Fired once a user has logged out of Jahia
userPropertiesSet	Fired once the user properties of a Jahia user have been updated
userUpdated	Fired once the properties of a Jahia user have been updated



9 route des Jeunes,  
CH-1227 Les acacias  
Geneva, Switzerland

[www.jahia.com](http://www.jahia.com)  
The Company website

[www.jahia.org](http://www.jahia.org)  
The Community website