

# Jahia CMS AND Portal Version 5.0

## Template developer guide for Jahia CMS and Portal Version 5.0



*English v0.1 - DRAFT*



Jahia Ltd  
Switzerland  
9, route des jeunes  
1227 Carouge

---

# TERMS AND CONDITIONS

Copyright 2003/2004 Jahia Ltd. (<http://www.jahia.org>). All rights reserved.

THIS DOCUMENTATION IS PART OF THE JAHIA SOFTWARE. INSTALLING THE JAHIA SOFTWARE INDICATES YOUR ACCEPTANCE OF THE FOLLOWING TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THESE TERMS AND CONDITIONS, DO NOT INSTALL THE JAHIA SOFTWARE ON YOUR COMPUTER.

## 1. LICENSE TO USE.

This is protected software (Jahia). The Jahia software is furnished under license and may only be used or copied in accordance with the terms of such license. Check the Jahia license ([www.jahia.org/license](http://www.jahia.org/license)) to know your rights.

## 2. NAMES.

The names JAHIA Ltd., JAHIA Solutions Sàrl, Jahia and any of its possible derivatives may not be used to endorse or promote products derived from this software without prior written permission of JAHIA Ltd. To obtain written permission to use these names and/or derivatives, please contact [license@jahia.org](mailto:license@jahia.org).

## 3. DECLARATIONS AND NOTICES.

This product includes software developed by the Apache Software Foundation ([http:// www.apache.org/](http://www.apache.org/)).

Windows and Windows NT are registered trademarks of the Microsoft Corporation. For more information on these products and/or licenses, please refer to their websites ([http:// www.microsoft.com](http://www.microsoft.com)).

Java is a trademark of Sun Microsystems, Inc. For more information on these products and/or licenses, please refer to their website (<http://www.sun.com>).

Other trademarks and registered trademarks are the property of their respective owners.

## 4. DISCLAIMER OF WARRANTY.

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.

## 5. LIMITATION OF LIABILITY.

THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS UP TO YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT SHALL JAHIA LTD. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY

OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

#### 6. GOVERNING LAW.

Any action derived from or related to this agreement will be governed by the Swiss Law and shall be of the competence of the judicial authorities of the Canton of Vaud, Switzerland.

#### 7. TERMINATION.

This agreement is effective until terminated.

END OF TERMS AND CONDITION

---

# PREFACE

Jahia is an enterprise-level Internet portal and content management system (CMS and Enterprise Portal). Differently than the other content management systems, Jahia supports in-context editing, so the same templates are used for editing and for browsing content. Like in the previous versions, also in Jahia 5 the template mechanism is based on Java Server Pages.

The principles and most of the taglibs and APIs are the same as in Jahia 4, so this documentation will present nearly the same information as before, just in a restructured way. All the functionality, which is new or has been changed in Jahia 5, will be highlighted, so if you already used Jahia 4.x, you can quickly find the new functionality.

**With Jahia version 5.1 we are planning to refactor our tag libraries and deliver a new, clean and easy-to-reuse set of templates. So this developer guide will have to be adapted regarding this new development, that is why it is still a DRAFT version. You may find some [TODO] tags in some chapters, which will be finished later.**

This guide is structured in the following way:

## [Chapter 1:Introduction to Template Development](#)

This chapter covers definitions and important concepts for developing Jahia templates.

## [Chapter 2:Defining Content](#)

This chapter describes the way, how content is defined and managed in Jahia.

## [Chapter 3:Creating Content](#)

Here you will find how to create the content, e.g. how to set the icons to call the edit popups, what possibilities you have to customize the view of the edit popups, and alternatively how to create content via input forms and via the Jahia API.

## [Chapter 4:Accessing Content](#)

This chapter covers the taglibs and API to access and display the content created with Jahia, also the sorting, filtering and paging of large container lists.

## [Chapter 5:Special Engines](#)

Here a template developer can learn how to make use of existing Jahia engines, e.g. for searching, displaying the site map and how to use the API to perform the publishing of Jahia content.

## [Chapter 6:Changing the Style](#)

This chapter is only about the possibilities to change the look&feel and styles of pages and fragments generated by Jahia.

## [Chapter 7:Extending Jahia](#)

This chapter describes ways to correctly extend (or fork) Jahia in order to ease migration and upgrades afterwards.

## [Chapter 8:Advanced Topics](#)

This chapter will throw some light on topics for advanced Jahia template developers, like the usage of event listeners, cache control, schedulers,...

## [Chapter 9:Appendix](#)

Here you will find entire template code examples, with detailed descriptions.

If you are a web designer just interested to take an existing set of template and to not change anything in the existing templates and definitions and to just want to know how to modify the CSS skins you should read [Chapter 6:Changing the Style](#) first. With the new set of corporate\_portal\_templates, which will be delivered with Jahia 5.1 we will also provide an easier way to change the "skin". So watch out.

Basic JSP scripters will either be interested in specific chapters picked from the table of contents or they may want to first go through an entire template to understand what the different code blocks do. In the previous version of the guide (still available on the Jahia homepage) there is a chapter presenting a simple template once written with taglibs and once with scriptlets and describing the different code blocks. We are planning to add such a description of a clean template in the Appendix of this new guide. Along with this guide there are also test templates, which should test and demonstrate all different possibilities described in the chapters. You can access them in our SVN repository: [http://svn.jahia.org/svn/doc\\_templates/trunk](http://svn.jahia.org/svn/doc_templates/trunk)

For a Java developer interested by adding new complex topics to an existing template set (e.g. adding some "user bookmarks" by accessing to the user properties, making some dynamic menus filtered by categories, filtering and personalizing content according to some user or group profiles and so on) the whole template guide should be interesting and also the available JavaDoc ([http://www.jahia.org/javadoc-dev\\_5/](http://www.jahia.org/javadoc-dev_5/)) and our different demo templates.

#### CONVENTIONS USED IN THIS DOCUMENT

The following conventions are used in this document:

*Italic* formatting is used for:

- comments
- emphasized words

Fixed width formatting is used for:

- JAVA, HTML and JSP code examples, literals and classes.

The following icons are used:



*Tips and Tricks*



*Notes*



*Warnings*

**jahia** **jahia** **jahia** **jahia**  
>4.1 >4.2 >5.0 >5.1

*Feature available since Jahia 4.1, 4.2, 5.0 or 5.1*

This document has been written with the goal of being as accurate as possible. However as Jahia is a product, which evolves rapidly, it may not cover new features available in the most recent versions of Jahia.

# TABLE OF CONTENTS

<i>Chapter 1:Introduction to Template Development.....</i>	<i>10</i>
<b>1 Definitions.....</b>	<b>10</b>
1.1 Fields.....	10
1.2 Containers.....	12
1.3 Container Lists.....	12
1.4 Pages.....	14
1.4.1 Child Pages.....	15
1.5 Site.....	16
1.6 Metadata.....	16
1.7 Content Model.....	16
1.8 Content Object's Functionalities.....	17
1.9 Engines.....	17
<b>2 Tag Libraries versus Scriptlets.....</b>	<b>18</b>
2.1 Tag Libraries.....	18
2.2 Scriptlets.....	18
2.3 Using Java Beans.....	19
2.4 Expression Language and JSTL tags.....	19
2.5 Conclusion.....	20
<b>3 Common Variables and Jahia JavaBeans .....</b>	<b>20</b>
<i>Chapter 2:Defining Content.....</i>	<i>21</i>
<b>1 About Content Definition.....</b>	<b>21</b>
1.1 Content Object's and Definition's Life-cycle.....	21
1.2 Declaration Scope.....	21
<b>2 Container Lists and Containers .....</b>	<b>22</b>
2.1 Sub-containers.....	26
2.2 Boxes.....	27
<b>3 Fields and Field Types.....</b>	<b>27</b>
3.1 Field Types.....	29
3.2 Default Values.....	30
3.2.1 Single Selection Choice List Markers.....	30
3.2.2 Multiple Selection Choice List Markers.....	31
3.2.3 Resource Bundle Marker.....	31
3.2.4 Expression Language Marker.....	31
3.2.5 Date / Time Fields.....	31
3.2.6 Page Fields.....	32
3.2.7 Colors.....	32
3.3 Metadata.....	32
3.4 Portlets.....	33
<b>4 Site Structure.....</b>	<b>33</b>
4.1 Limiting Available Templates.....	33
4.2 Limiting Pages to be Linked or Moved.....	33
4.3 General Pages.....	33
<i>Chapter 3:Creating Content.....</i>	<i>34</i>
<b>1 Action Menu.....</b>	<b>34</b>
<b>2 Engines and Edit-views.....</b>	<b>34</b>
2.1 Controlling the Order and Grouping of Fields.....	34

2.2 Using Custom JSPs to Edit Fields.....	35
2.3 Field Input Validation.....	35
2.3.1 Apache Commons Validator.....	35
2.3.2 Adding Validations in the Container Declaration.....	35
2.3.3 Providing a JavaBean According to the Container Declaration.....	35
2.3.4 Defining Validation Rules.....	37
2.3.5 Configuring and Enabling the Validator Plug-in.....	38
2.3.6 Displaying Validation Error Messages.....	39
2.3.7 Advanced Validation Example.....	39
2.4 FCK Editor Templates.....	41
<b>3 Managing Forms with Jahia.....</b>	<b>41</b>
3.1 Form Handlers.....	41
<b>4 Creating Content with Jahia API.....</b>	<b>44</b>
4.1 Setting ACL Rights.....	44
4.2 Setting Categories.....	44
4.3 Publishing content.....	44
<i>Chapter 4: Accessing Content.....</i>	<i>45</i>
<b>1 Accessing Container Lists and Containers.....</b>	<b>45</b>
1.1 Accessing Container Lists.....	45
1.1.1 Absolute Addressing (aka Absolute Container Lists).....	45
1.1.2 Relative Addressing (aka Relative Container Lists).....	45
1.1.3 Page Level Addressing (Tags only).....	46
1.1.4 Restrictions.....	46
1.2 JavaBean API.....	46
1.3 Re-using Content using Container Filters.....	47
1.4 Categories.....	47
1.4.1 Browsing Categories.....	47
1.4.2 Finding Object Categories.....	49
1.4.3 Front-end Cache Issues.....	49
1.4.4 Container Filters and Categories.....	49
1.5 Expressions.....	50
1.5.1 Expression Language in Field Declarations.....	50
1.5.2 Expression Language in Templates.....	52
1.5.3 Expression Performance.....	53
<b>2 Container List Sort, Search and Pagination.....</b>	<b>53</b>
2.1 Search, Filter and Sort.....	53
2.1.1 Jahia Page Form.....	53
2.1.2 Search Options.....	54
2.1.3 Filter Options.....	54
2.1.4 Sort Options.....	56
2.2 Container List Pagination.....	58
2.2.1 General Considerations.....	58
2.2.2 Next, Previous Buttons.....	59
2.2.3 Quick Page Access Buttons.....	60
2.2.4 Container List Pagination Information Tags.....	62
2.2.5 Customizable Items per Page Option.....	63
2.3 Dealing with Multiple Languages.....	63
2.4 Examples.....	63
2.5 FAQ.....	64
<b>3 Navigation.....</b>	<b>64</b>
3.1 Page Path.....	64
3.2 Basic Children Navigation.....	65

3.3 Advanced Navigation (with Recursive Display).....	66
<b>4 Filtering.....</b>	<b>71</b>
<b>5 Other Template Issues.....</b>	<b>72</b>
5.1 References to Static Resources.....	72
<i>Chapter 5:Special Engines.....</i>	<i>73</i>
<b>1 Search and Advanced Search.....</b>	<b>73</b>
1.1 Search Architecture Overview.....	73
1.2 Using the Search Functionality.....	73
1.2.1 Inserting a Search Link.....	73
1.2.2 Inserting a Search Form.....	74
1.3 Further Advanced Search Customization.....	76
1.3.1 Advanced Search Architecture.....	76
1.3.2 Global versus Site Searching.....	76
1.3.3 Page Subset Searching.....	77
1.3.4 Searching Specific Languages.....	77
1.3.5 Container and Field Searching.....	78
1.4 Customizing the Results View.....	79
<b>2 Sitemap.....</b>	<b>79</b>
2.1 Inserting a Sitemap.....	79
2.2 Customizing the Sitemap Engine.....	82
<b>3 XML Import/Export.....</b>	<b>88</b>
3.1 XML Export Engine Documentation.....	88
<b>4 Workflows.....</b>	<b>89</b>
<i>Chapter 6:Changing the Style.....</i>	<i>90</i>
<b>1 Engine Pop-up Customization.....</b>	<b>90</b>
1.1 Localizing the Portal with Resource Bundles.....	90
1.2 Customize the Portal's Appearance.....	93
1.3 Customize Add and Update popups with ACL.....	95
1.4 User Registration, Login and Settings Customization.....	96
1.4.1 Registration.....	96
1.4.2 Login.....	98
1.4.3 Logout.....	99
1.4.4 Settings.....	99
1.5 Summary.....	102
<b>2 Action Menu Customization.....</b>	<b>102</b>
<b>3 General Admin Toolbar.....</b>	<b>102</b>
<i>Chapter 7:Advanced Topics.....</i>	<i>103</i>
<b>1 Internationalization.....</b>	<b>103</b>
1.1 What is Internationalization?.....	103
1.2 Resource Bundles.....	103
1.3 Bundle Editor.....	105
1.4 Usage.....	105
1.5 Resource Bundle Tags.....	105
1.6 Tags with Native Support for Internationalization.....	106
1.7 Resource Markers.....	106
1.8 Script-based Internationalization.....	107
1.9 Changing Current Language On-the-fly.....	108
1.10 Engine Internationalization.....	109
<b>2 Event Listeners.....</b>	<b>109</b>



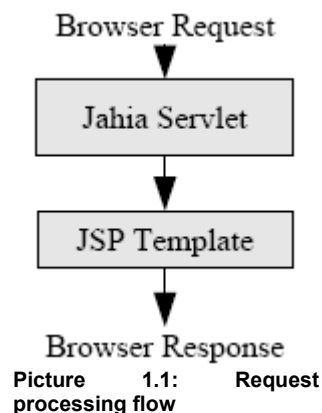
2.1 Supported Jahia Events.....	110
2.2 Usage.....	111
<b>3 Cache Issues.....</b>	<b>112</b>
3.1 Request Processing Flow.....	113
3.2 Controlling Cache Expiration.....	113
3.3 Detail Page Instead of Overview.....	113
<b>4 Cross-link System.....</b>	<b>113</b>
4.1 Introduction.....	113
4.2 Adding Persistence.....	114
4.3 Generic Links.....	114
4.4 Link Examples.....	115
4.5 Database Table Format.....	116
<b>5 Scheduler.....</b>	<b>116</b>
<b>6 Integrate own Struts Actions.....</b>	<b>116</b>
<b>7 Packaging your Templates.....</b>	<b>116</b>
7.1 Package Constituents.....	116
7.2 Detailed Example.....	117
<i>Chapter 8:Appendix.....</i>	<i>120</i>
<b>1 Taglib tutorial.....</b>	<b>120</b>
<b>2 References.....</b>	<b>120</b>

# CHAPTER 1: INTRODUCTION TO TEMPLATE DEVELOPMENT

*This chapter covers various definitions and concepts for developing Jahia templates.*

## 1 DEFINITIONS

One simplified way of defining the processing architecture of Jahia would be the following :  
[TODO BETTER PICTURE]



After some internal processing, Jahia forwards to a JSP template, which handles the display of the currently selected page. Actually a template is able to do a lot more than simply display content, since JSP technology allows users to include custom tags, use Java beans to implement and call business logic or directly use Java code scriptlets within the JSP page.

Jahia's content management system allows users to declare the type of content they want to manage, in order to structure the data in a meaningful way. Many web-based content management systems simply allow users to edit free text on a page, without guiding them into some kind of structure. This last approach has its advantages, as well as its drawbacks. The best way to illustrate how structured content may be useful to users is to illustrate this with a simple example : an address list.

### 1.1 FIELDS

Let's say that on an intranet site you want to build a list of addresses. This list would be a very simple list of people, with a few predefined fields that are common to all users. Such fields could be :

- first name
- last name
- street
- zip code
- city
- country
- phone number

If this was to be stored in a free text form, there would be no way for the users of the content management system to use a uniform structure that would help them input the data in a structured way. In HTML free

text for example the temptation would be great to use presentation tags such as tables or lists, but it would lack the semantic information.

Jahia introduces the basic elements of *Jahia Fields*, to allow users of the CMS (content management system) to structure their data. Fields may have different types, such as :

- Limited text
- Unlimited text
- List of user-selectable values
- Date
- Number
- Color
- File (image, binary file)
- Portlet application

The above list is not exhaustive, as the basic field types may be used for all kind of different scenarios. Also, the list we have presented represents end-user styles, meaning that they represent the type of data that the end-user will be able to input, but the actual content object used to store the types are a little different as we will see. The first thing we have to do is to choose the field type(s) we will use to store the address data. In order to keep things simple, we will use a limited text field to store each address field. In Jahia the limited text field is called `SmallText` field, and is able to store up to 255 characters. For a detailed description of the available field types, consult section Field Types of this guide.

Fields are defined in Jahia using an identifier called the *definition name*. This name must be unique within the current context, which may be the current container or even the current page template (we will go into more details on that restrictions in section Declaration Scope). A field definition name may also be seen as the *sense* we give the field value, meaning we give the value semantic information by giving it a name<sup>1</sup>. So for our address fields, we can now manage them using field types and definition names. Note that it is recommended to use definition names that do not use spaces, and generally we recommended following the Java naming conventions (first letter in lower case, and every new word starts with an upper case character).



You have to be very careful with the container and field definitions in Jahia. It is possible to use the same field definition name in different containers, but only under limited conditions, which are described in section Declaration Scope.

Address field	Jahia Field Type	Field definition name
first name	SmallText	firstName
last name	SmallText	lastName
street	SmallText	street
zip code	SmallText	zipCode
city	SmallText	city
country	SmallText	country
phone number	SmallText	phoneNumber

**Table 1: Address fields to Jahia fields mapping**

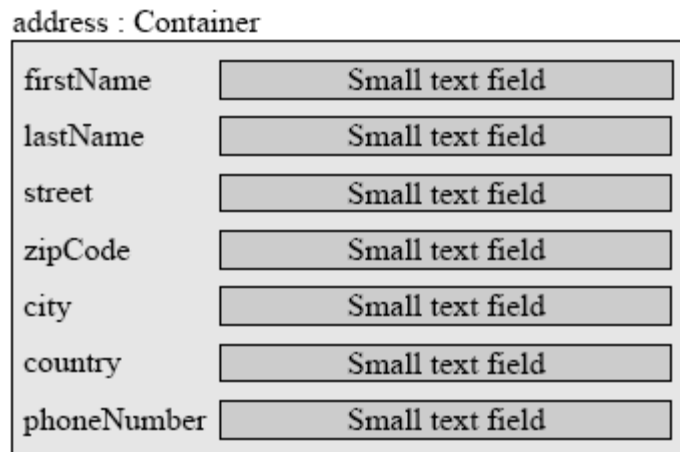
<sup>1</sup>An important field in the evolution of the World Wide Web is the Semantic Web. For more information consult the semantic web pages at : <http://www.w3.org/2001/sw/>

## 1.2 CONTAINERS

Jahia allows fields to be grouped into a logical entity called a *container*. A container may include any number of fields, as well as sub container lists (which we will define later on). As example we will use a container to represent an address. The grouping of fields in containers also defines the ordering (or ranking) of the fields within the address, which will be used by Jahia's edition tools to present the content to the user. This way when a new address must be entered, the order of the fields presented will match the order of the fields when declaring the address container. There is also a possibility to define the order and grouping of fields, by using the `ContainerEditView` objects, which are described in section Controlling the Order and Grouping of Fields.

Just as we were able to identify various fields by a definition name, a container definition may also include a name. In this case the most obvious name for the container would be `address`, since this is what our logical grouping of fields will represent. So in effect we have a container definition that looks something like this :

[TODO BETTER PICTURE]



Picture 1.2: Address container

The container structure we have defined will be used to present to the user for data input. Containers are really a way to define structured grouping of fields, as well as being entries in container lists.



Jahia allowed to use field definitions without a container definition, so the fields were directly located under the page and not in the container. This is still possible in Jahia 5, but only to support legacy applications. Please consider using fields without a container as a deprecated functionality. In future Jahia versions this will not be allowed anymore.

## 1.3 CONTAINER LISTS

A container list is simply a list of containers. It also defines an order, which by default is the order in which the containers were added to the container list. Container lists may only be composed of containers of a specific definition. In our example, the container list would be an address list, and only our current definition would be accepted as valid entries for the container list.

[TODO BETTER PICTURE]

Container List

address : Container	
firstName	Small text field
lastName	Small text field
street	Small text field
zipCode	Small text field
city	Small text field
country	Small text field
phoneNumber	Small text field

address : Container	
firstName	Small text field
lastName	Small text field
street	Small text field
zipCode	Small text field
city	Small text field
country	Small text field
phoneNumber	Small text field

**Picture 1.3: Container List containing two containers**

Contrary to containers and fields, container lists do not have definition names, they are simply referred to using the container definition name, which is fine since a container list may only contain one type of container.

Container lists cannot be created or deleted by end-users in Jahia, they are objects whose life cycle is automatically managed by Jahia. The system will automatically create the container list when the first container is created, and container lists are deleted once the parent object is deleted.

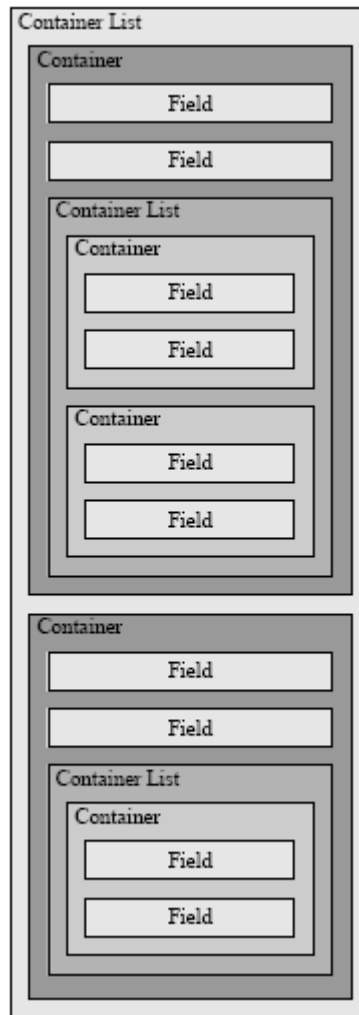
The parent object of a container list is usually a Jahia page, but may also be a container. The possibility to embed a container list within a container is a powerful mechanism to create complex structures.

In order to embed container lists within a container, we need to define a container structure that includes another container structure<sup>2</sup>. As the above example illustrates, we can mix fields and sub container lists within a container structure. It is possible to include several sub container lists of different structure within a container, provided we have declared the container to allow for such a sub-container structure. One advanced way of using such a structure would be to use a field to define the sub-container list to display what would look like a varying structure. This technique is used within Jahia's default corporate portal templates for the `box` container structure, which uses sub-container structures called `files`, `applications`, etc...

[TODO BETTER PICTURE]

---

<sup>2</sup> The same definition cannot be used for the embedded container structure, otherwise a recursive definition would occur, and Jahia doesn't support recursive definitions.



Picture 1.4: Container Lists embedded in containers

It is perfectly allowed to define more than one level of sub-container structures.

On the end-user side, the user will be presented with the interface to add containers in the top level container list. Once he has added a container, he will usually (depending on how the template designer manages the content editing GUI) be presented with the possibility of adding sub-containers for the container he has just added.

Most of the time container lists will have as a parent object simply a Jahia page, which will be the next object we will introduce.

## 1.4 PAGES

A page in Jahia can be viewed as something similar to an HTML page. When a page is created, it is associated to a template, which will define the structure of the content in the page, as well as how it should be displayed. The page template will generate the full HTML for the page display, and pages may be added to a web site dynamically, to easily extend the information presented. A page template has two responsibilities :

1. declaring the content structure for the page

## 2. rendering the instances of the content objects that comply to the structure

When a new page is created in Jahia, the content structure is used to present all the corresponding user interfaces for input of new data. The next step is to load all the existing instances of content objects (of course there won't be any for a brand new page), and to display them. There are some cases where objects will be directly created upon first access of the container.

The rendering part of the template basically queries Jahia for any existing container lists, and then iterates through the containers, retrieving the field values and displaying them. In the case of containers that include container lists, the same process is repeated.

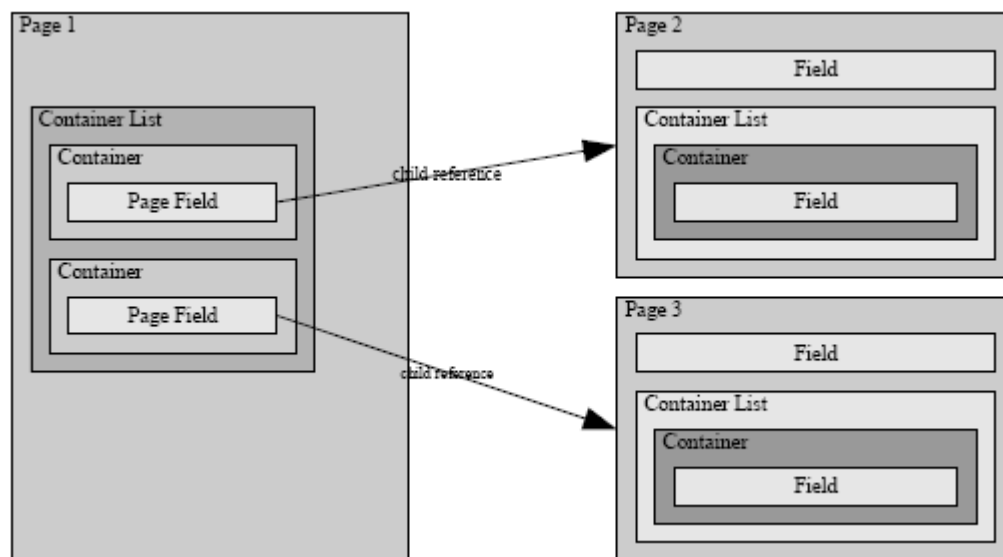
One important thing about content definition in templates is, that it is only used the first time the page is accessed and whenever there are modifications to the definitions. The content definitions are also stored in Jahia's database. It is possible to add new field definitions at any time, but it is not allowed to change the data type of an existing definition once data has already been captured. When removing a field definition from a container definition, the already captured data will not be completely deleted, but will only be deactivated. If you again add the removed field definition to the container definition, the previously captured data will be accessible again.

Finally we will see later in this guide, how to access content in absolute or relative ways, in order to share content between different Jahia pages.

### 1.4.1 CHILD PAGES

One of the particularities of Jahia pages is that child pages are not direct child objects of their parent page. Rather child pages are children of a field type called the *Page Field*. The logic behind this design is that navigation can be constructed using containers, and so it is possible to have children pages in different container lists on a page, making for a very flexible hierarchy model.

[TODO BETTER PICTURE (remove the standalone field)]



Picture 1.5: Child pages are referenced by page field

This is the standard way for building navigation in Jahia templates. One of the nice side effects of this model is that the ranking (ordering) of containers can be used to set the ranking of the child pages, since they are simply referenced through a page field. In this example we have presented the page field as capable of storing a *child* reference (also called *direct page* type) to a page, but the field is also capable of storing *link* references, which behave as a link to an existing Jahia page anywhere on the web site. Finally the page field is also capable of storing an URL to reference an external web site.

## 1.5 SITE

A Jahia site is a logical entity that regroups the following data :

- a set of templates
- a set of users
- a set of groups
- a set of portlet applications
- a site key
- an optional site host name (such as `demo.jahia.org`, where `demo` would be the site key on the server serving all the `*.jahia.org` Jahia virtual sites)
- a home page (the entry page to the site)
- a search engine index
- a set of languages in which the content may be input

The administration of a Jahia virtual site is covered in the Jahia Administration Guide so we will not cover it here. The important points relating to template development are the set of templates that are made available for a Jahia site and the site home page. The set of templates is simply a list of templates that the end-users will be able to use when creating new pages. It is possible to deploy new templates in a site while the system is running. The site's home page is simply the entry point into the virtual site. It is automatically created by Jahia, using the default template when creating a new site.

## 1.6 METADATA AND CATEGORIES



Jahia 5 has now a new support for metadata. Template developers are now able to assign default metadata for all content objects at a server level. Jahia is pre-configured by default with the standard [DublinCore](#) set of metadata such as the original author, the creation date, the last modifier, the last modification date, the description, etc. Of course you can perfectly add your own new custom metadata. Of course all the options such as filters, etc... are present and can be used on this new metadata fields.

Categories have also been modified. We added support for a new category field type. This one is directly mapped to the category tree. So you now can have several different fields linking to a tree of default values (e.g. classification, category, etc... fields). Template developers are now also capable of defining the entry point in the category tree for each field. So you are not forced any more to start at the category root level. We also introduced permissions on each node of the category tree in order to restrict the use of certain sub-tree to certain users only.

## 1.7 CONTENT MODEL

Now that we have presented the basic content structure that is available in Jahia, we will present a summary of the organization of content on a typical Jahia site.

[TODO PICTURE like in the previous template guide version just without standalone fields ]

In the above drawing, the content object types are indicated in between parenthesis, while the name before it varies on the object type:

- page object: the name in the graph is the actual display name of the page.
- container list object: we have no name for this object, so that is why only the type is indicated
- container object: the name is the container definition name
- field object: the name is the field definition name



The above object tree includes only fields grouped in containers. Fields directly attached to a page are deprecated now. Also on the right part of the tree we see a navigation hierarchy, which contains *child page reference* fields that point to the *About Us* and *Product* sub-pages. The only feature not illustrated in this object model are container lists contained in containers, but they were illustrated previously. Another way of defining the format is to present embedding possibilities :

- pages may contain container lists
- container lists may only contain containers
- containers may contain fields and/or container lists

## 1.8 CONTENT OBJECT'S FUNCTIONALITIES

Jahia's content management system primarily deals with content objects, these include :

- pages
- container lists
- containers
- fields

As we have seen these content objects may be combined together to build complex content structures using templates that contain their structure definitions.

Content objects also share common functionalities, that the end-user can interact with, that cover a wide range of functions such as access control, versioning, and so on. Here is the list of features available on all content objects:

Feature name	Description
ACL (Access control lists)	Access controls lists allow users to define a list of users and/or groups that may be given/refused permission to either read, write or administer (delegate rights) a content object. For example by removing read access to certain containers for certain users, we can personalize access to the container. The rights on fields are stored at the container list level, that means that fields based on the same field definition have the same right in all containers of the container list.
Staging (workflow states)	Content objects may go through intermediate stages of workflow before being published live. Usually a new content object will first go to a "staging" state, then a "waiting for validation" state, and finally to a "live" state if it was accepted, or back to "staging" state if it was refused. With Jahia 5 you can also define different workflows on up to a per-object basis, so you can for instance define custom N-step workflows or "No workflow" for immediate publishing or even an "External workflow".
Versioning	Every time a content object is validated, the previous version of the content object (including all it's sub-objects) is versioned, making it possible to later retrieve the state of the content object by restoring the state at a specified date.
Locking	Content objects are locked to prevent editing either if another user is currently already modifying the same object, or if the object or one of it's ancestors is in the "waiting for validation" workflow state.
Multi-language	Content objects may either have different values for each language or shared their content in all the languages (for example container lists and containers share their content in all the languages, but not all fields do).
DublinCore Metadata support	Since Jahia 5 every container list, container and page has a set of

Feature name	Description
In-context Copy/Paste option	expandable metadata fields attached, which are based on the DublinCore standard ( <a href="http://en.wikipedia.org/wiki/Dublin_Core">http://en.wikipedia.org/wiki/Dublin_Core</a> ), such as the Author name, the Validator name or the Creation Date. Every content modification or validation is then automatically stored and indexed in the Jahia CMS.
Content Picker (dynamic and static copies)	You can now easily copy and paste content throughout a site, be it a single container, a container list or a whole section of a site. Jahia will check for "compatibility" between the source content and the target, to only allow you to paste it when similar definitions are used.
XML Import/Export feature	A content picker has been integrated on top of the XML import/export feature. The content picker allows any author to easily find and reuse existing content on another page. The author has the choice to make a static copy of the existing content (no link) or a dynamic copy (each change on the source will impact the destination).
Indexation and score boost	A new XML import/export feature will allow any user to export some Jahia content (a container, a full list, a page or a whole virtual site) and to import it elsewhere (including on another Jahia server). The XML import/export tool will also manage "diffs" and only import changes from a certain date.
	The content in Jahia is automatically indexed (unless you mark a field as not indexable) and since Jahia 5 you can also set a search score boost factor to content objects.

**Table 2: Content object's common functionalities**

[TODO DESCRIPTION of more new functions available since Jahia 5]

## 1.9 ENGINES

One of Jahia's core components is the *engine*. An engine is a functional unit, which performs specific tasks usually requiring user interactivity. Engines are formed by a set of one or more classes and/or JSP files. For readers familiar with Struts, engines are similar to Struts Actions.

In Jahia every operation carried out by the users utilizes an engine. For example, the login operation, the user registration, searching, adding or updating content, updating page properties and so on. A simple way to spot which engine is being used for each operation is to check the current URL. It will typically contain the following identifier `/engineName/` and a following keyword which names the engine. Jahia parses the URL and uses the engine identifier to dispatch incoming user requests to the appropriate engine.

The point of entry of an engine is usually its `*_Engine.class` file. This class carries out all the flow control for that particular engine. Typical operations include loading/saving data into the database, verifying submitted form data, loading data into the session, carrying out a search and filtering the results, and carrying out most of the control logic. An engine can also call other sub-engines (i.e. update container engine calls field-specific engines). The engine class will then usually use an external JSP to render the user response. For example, for the user login, the `org.jahia.engines.login.Login_Engine.class` checks the current user status and forwards the request to the `login.jsp` page. From there, the user submits his login details which are processed by `Login_Engine.class`. If an error is detected, the user is forwarded to `bad_login.jsp`, if the login is successful, the user is forwarded to `login_close.jsp` which closes the login popup and forwards him to his main page.

For a visual representation of an Engine workflow, please refer to the section User Registration, Login and Settings Customization.

## 2 TAG LIBRARIES VERSUS SCRIPTLETS

Sometimes a template developer may need to solve complex requirements. With Jahia you have at least 3 and soon 4 different possibilities to write complex templates, which access Jahia content. These are Jahia tag libraries, scriptlets, Java beans and soon you will be able to use the new expression language and the latest available JSTL tags.

### 2.1 TAG LIBRARIES

One of the purposes of tag libraries is to reduce the need to introduce Java code inside templates. Although very powerful, Java code introduces a lot of dependencies, and often makes underlying changes to the Jahia API very difficult. tag-libs are also usually a little easier to use, but can make the template a lot more verbose.

Jahia's tag library is now at it's second iteration. The first version of the tag library contained a lot of tags, because there were tags for all the logic tests (such as `IfIsHomePage`, `IfIsNotHomePage`), that were mostly very similar, and not very flexible. The second version of the tag library has been completely redesigned to make it compatible with the Standard JSP tag library, as well as the Apache Struts tag library. Both these libraries contain logic tags, bean manipulation tags, and so on. The logic tags have therefore been removed from Jahia (although they still exist for compatibility reasons but are no longer recommended). The advantage of relying on these external libraries is that there is already a lot of excellent documentation on these tools, and people can reuse the skills along with the Jahia-specific libraries.

Pros :

- Simpler to use, no required knowledge of Java
- Doesn't introduce dependencies on the Jahia Java API
- Compatible with the JSP standard tag library and the Apache Struts tag library
- Simpler to read

Cons :

- Limited by what tags offer
- Can be very verbose for simple tests
- Slower to compile than Java code, at least with Tomcat's JSP compiler

### 2.2 SCRIPTLETS

Using scriptlets actually means introducing Java code into the templates, which is usually not a good idea and should be avoided as much as possible. Why? Simply because this code is usually not compiled alongside the rest of the Jahia code. Since this code may contain errors, this makes debugging more complicated. Also, because of the structure of JSP pages, mixing Java code and HTML tags usually makes for poor readability. Along with increased maintenance complexity, a lot more code is necessary to achieve the equivalent functionality of a tag.

Other problems include the fact that usually in a MVC (model-view-controller) design pattern, the JSP, which is the view, should not be able to do any modifications to the underlying model objects. When using Java code within a JSP page, it is impossible to limit the actions of the template designer, so he may do whatever he wants. This might be exactly what is needed, but usually removing code out of a JSP and putting it in a tag-lib or a Java bean is recommended to make the project more maintainable.

Despite all the negative points, there are some cases, where adding Java code makes for simpler pages, especially for example: during prototyping.

Jahia offers a set of APIs designed to be also called from a JSP template, and another set for back-end tasks which should be reserved for advanced users, who know exactly what they are doing. It should be

noted that APIs have a tendency to change due to new feature introduction or to a sub-system refactoring. Therefore, calling the APIs directly might entail extensive template updating to support future versions of Jahia (we will do our best to avoid modifications, but not at the cost of efficient design or simplicity).

Pros :

- Everything is possible, the sky is the limit
- In some cases, makes for less complex templates than with taglibs
- Faster to compile, at least with Tomcat's JSP compiler

Cons :

- Can easily become unreadable
- Usually not compiled with source code, unless the templates are precompiled
- No guarantees on compatibility with future versions of Jahia
- Doesn't guarantee MVC model separation

## 2.3 USING JAVA BEANS

In order to keep the JSP templates readable and to prevent compiler errors, you may decide to use your own Java beans to implement the business logic and to call those beans from your JSPs. This way you can also use all the available Jahia APIs.


Pros :

- Simpler to read
- Cleaner separation of model, view and controller logic
- Not compiled at runtime and thus no hidden compiler errors will occur

Cons :

- No guarantees on API compatibility with future versions of Jahia
- Java knowledge necessary

## 2.4 EXPRESSION LANGUAGE AND JSTL TAGS

 For now we still have to stay compatible with the Java Servlet 2.3 standard due to customers running Jahia on Websphere 5, but with Jahia 5.1 we are going to support at least the Java Servlet 2.4, Java Server Pages 2.0 and the corresponding Java Server Pages Standard Tag Library 1.1 standards.

With using the expression language and the JSTL tags, some more Jahia tags will become obsolete and you will be able to write even more elegant and simple looking JSPs.

If you are using Tomcat 5, you could already use these standards, by migrating the web.xml to the 2.4 standard, and exchanging the `lib\jstl-1.0.2.jar` and `lib\standard-1.0.4.jar` and also the `etc\taglibs\jstl\*.tld` with the newer versions.

## 2.5 CONCLUSION

In this document we will mainly describe the usage of the Jahia tag libraries along with the Struts v1.2.4 and JSTL v1.0.2 tags. As scriptlets are seen as bad practice, we will pay less attention to it in this guide, although for each tag there is also an API counterpart. As we recommend moving complex methods to Java beans and because some functionality cannot be solved with taglibs yet, we will also include some scriptlet examples here and there.

# 3 COMMON VARIABLES AND JAHIA JAVA BEANS

In the scriptlet examples you will often see the reference to the `jData` or `jParams` variable. Jahia passes a few objects via the request object, of which two important ones are :

- `JahiaData`: contains all the content objects for the current page
- `ProcessingContext` (previously `ParamBean`): contains all the Jahia parameters for the current request, plus basic URL generation methods. You can think of this object as an extension on the request and response objects.

They can be retrieved like this:

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

Jahia also puts the following JavaBean objects as attribute in each request object, so they can be accessed via the taglibs, the scriptlets and the expression language:

Attribute name	Class	Description
<code>currentPage</code>	<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
<code>currentSite</code>	<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.
<code>currentJahia</code>	<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
<code>currentRequest</code>	<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..
<code>currentUser</code>	<code>org.jahia.services.usermanager.JahiaUser</code>	This object is the user currently logged in <code>JahiaUser</code> (would be guest if the user is not logged in), and provides access to user properties

**Table 3: Jahia JavaBean objects accessible via the request object**

## CHAPTER 2: DEFINING CONTENT

*This chapter describes the way, how content is defined and managed in Jahia.*

### 1 ABOUT CONTENT DEFINITION

Jahia is different from most other content management systems, because it allows content designers to combine basic working blocks to enforce content structure. This means that like a XML structure is defined using DTDs or schemes, Jahia's content objects must respect the structure defined by the content designer.

Content declarations (or definitions, these terms are used interchangeable and mean the same) are used to define the format of the content objects, which the user will be allowed to add and/or edit. Declaration refers mostly to the actual moment when the content definition is done.

To understand the difference between content definition and content object, we could make a parallel with the Java language: a content definition would be a Java class, whereas the content object would be the associated Java object instance.

#### 1.1 CONTENT OBJECT'S AND DEFINITION'S LIFE-CYCLE

A container list has a lifetime that starts when the first container is inserted or the container list properties are edited, and only disappears when the page is deleted. This is of course only the case for container lists that are directly inserted in the page. Sub-container lists have a lifetime that is tied to the parent container. As soon as the container is deleted, all sub container lists are removed.

The same is not true for content definitions. Content definitions never get removed. Content definitions are attached to a page template, and even if the template changes, the previous definitions stay in Jahia's database. This is done notably to allow template swapping. You can change a page's template, without loosing neither the content definitions, nor the actual content objects.



So to resume we have:

- field definition life cycle = Jahia installation life-cycle
- container structure definition life cycle = Jahia installation life-cycle



Once you have created content objects, you are not allowed to change the field definition name and the field type anymore, while the other attributes of the content definition may be changed.

#### 1.2 DECLARATION SCOPE

Now that we have looked at the life-cycle of content definitions, we will look at the scope of the declarations, in order to understand name clashing issues.

In Jahia, some content declarations have a scope of the page template and some of the entire site.



For example this means that we cannot have on the same template two container or field definitions that have the same name but different attribute values. You can perfectly reuse the same field definition names in different container definitions on the same page, if you can ensure that all attribute values and field definition properties are exactly the same.

As long as you are not defining extended properties for a field (like `aliasName`, `readOnly`, `indexableField`, `scoreBoost`, `field_update_jsp_file_prop` and custom or future ones), you could even use the same field definition name on another template and use different attribute values for e.g. different default value or title.



Warning Once you are setting an extended property for a field, this definition becomes scoped for the entire site, that means the definition must be the same in all templates.

The same is true for container structure definitions. Let's say we have two container structures : `Container_A` and `Container_B`. `Container_A` declares a sub container structure called `RelatedDocuments`. `Container_B` can now not declare a sub container structure definition also called `RelatedDocuments`. It must use a different name. One way around this is to append the parent container structure definition name, such as `RelatedDocuments_A` and `RelatedDocuments_B`.

What is allowed though is to use the same name for a container structure declaration in two different templates. So for example template `Template_A` and `Template_B` may both have a container structure declaration called `News`, that have different structures. This is possible but not recommended since Jahia includes methods for aggregating container lists by name, and if the structures of the containers are different, it will be much more difficult to display them, because for a given name we will have different types of objects. Future versions of Jahia might even require that the scope of container structure definition be local to the current site, in order to ease re-use of structure definitions on a web site.

Therefore the content designer beforehand has to pay much attention to the content and field definitions, to not create conflicts, which will lead to instabilities and are hard to resolve. Renaming a field in a later stage is not possible without losing the already captured data.

Jahia will log any detected changes to container and field definitions to the system log. So whenever you see that a container or a field is being constantly redefined, then you probably have a name conflict. For the same field or container name you have multiple different definitions and you should immediately try to make these differing definitions being exactly the same.



From Jahia 5.1 (exactly from Jahia 5.0.SP3) onwards container and field definitions will only be applied when accessing a page in EDIT mode. So after changing a container or field definition you have to first access in EDIT mode at least one page per template(s), which include this definition.

## 2 CONTAINER LISTS AND CONTAINERS

Container lists are holding one or more containers, but are not allowed to mix containers that have different structures. That is why the container structure declaration could be also called *container declaration* or *container list declaration*.

Container structures may contain two types of content definitions:

- field definitions
- sub-container definitions

Sub-container definitions are like container definitions, except that their declaration happens within the declaration of a container structure definition.

For now, container definitions have to be done in the template JSPs, but this may change in the near future. You do not need to call the container declaration on each page or template, where you access the containers and fields, but only on those page templates, where the content will actually be located and/or edited. For instance the `siteSettings` container is located at the homepage. If you would make the edit

icons for the `siteSettings` only visible on the homepage, so that updating the `siteSettings` could be done only from there, then you may include the `siteSettings` container and field definitions only in the home template or you could put an

```
<c:if test="${currentPage.level == 0}">
</c:if>
```

or

```
<c:if test="${currentPage.id == currentSite.homepageID}">
</c:if>
```

bracket around the declarations, if you use JSTL, otherwise you could alternatively use the Struts tags or scriptlets. This way the definitions will not be created or checked on accessing each single page in your system, while you will still be able to read the `siteSettings` from any page.

We give you an example for the taglib and the Java API version of the same container list declaration, which as example contains two simple fields, called `title` and `content`.

#### TAGLIB VERSION:

```
<content:declareContainerList name='simpleContainerList' title='Simple container list'>
  <content:declareContainer>
    <content:declareField name='title' title='Title' type='SmallText' />
    <content:declareField name='content' title='Content' type='BigText'
      value='Enter text here' />
  </content:declareContainer>
</content:declareContainerList>
```

#### JAVA API VERSION :

```
// fields declaration
jData.containers().declareField( "title", "Title", FieldTypes.SMALLTEXT, "" );
jData.containers().declareField( "content", "Content", FieldTypes.BIGTEXT,
  "Enter text here" );

// vector to store the fields
Vector simpleContainerFields = new Vector();
simpleContainerFields.add( "title" );
simpleContainerFields.add( "content" );

// container list declaration
jData.containers().declareContainer( "simpleContainerList", "Simple container list",
  simpleContainerFields );
```

The declaration of fields will be explained in the next section, for now we only concentrate on the container declaration, although it is to say here that the order of fields is essential and used by Jahia to automatically create the GUI presented to the editors in a popup engine window. According to the order of the field definitions Jahia will group the edition fields for the whole structure. As some of the field widgets take a lot of space, they will be presented in a separate tab, but all the edition for the container will happen in the same popup window. There is a possibility to specify a custom grouping of fields in the engine window by using the `ContainerEditView` object. This is described in section Controlling the Order and Grouping of Fields. It is also possible to use custom JSPs for editing a field. You will find a description in section Using Custom JSPs to Edit Fields.

Consider container definitions as being constant and not changed very often, similar to a definition of a database table. All the containers in the container list should use the same structure, you cannot for instance have one single product container list and depending on the type of product, define different fields. What you can do is define multiple sub-container lists and depending on the type of the product, initialize and use only certain sub-containers. Sub-containers are described in section Sub-containers.



In the simple example above, we only set the name and the title of the container. There are also other possible attributes and properties in the taglib and API, which are described in the following table.

Attribute or property name	Description
name	It is an identifier that must be unique within the page, and usually it is a good idea to make it unique within the set of templates especially if you plan to reuse content later on between several pages (see section Declaration Scope). Avoid using spaces and special characters (like [ ] &) in this name as the import/export utility uses the container names as element names in the XML file.
title	It will be displayed in the update container list window. Unlike the name, it should be as end-user friendly as possible, since it will be displayed in the browser. It is possible to make the title multi-lingual using the <code>titleKey</code> attribute.
titleKey	This key uniquely identifies the locale-specific field title in the associated <code>bundleKey</code> bundle appropriate for the current user's locale. Note that if <code>titleKey</code> 's value cannot be found, the value in the <code>title</code> attribute (if defined) is used instead. If <code>titleKey</code> and <code>title</code> are both defined, then precedence is given to <code>titleKey</code> .
bundleKey	<p>Resource bundles contain key/value pairs for specific locales and are used for internationalization (see section Resource Bundle Tags). Note that Jahia finds the <code>*.properties</code> file associated to a given <code>bundleKey</code> by doing a lookup in <code>resourcebundles_config.xml</code> located in <code>\WEB-INF\etc\config</code>. The <code>resourcebundles_config.xml</code> file is generated automatically during template deployment. For example, given the following <code>resourcebundles_config.xml</code>:</p> <pre data-bbox="462 1039 1437 1722"> &lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;registry&gt;   &lt;!-- core resources --&gt;   &lt;resource-bundle&gt;     &lt;key&gt;JahiaEnginesResources&lt;/key&gt;     &lt;file&gt;JahiaEnginesResources&lt;/file&gt;   &lt;/resource-bundle&gt;   &lt;resource-bundle&gt;     &lt;key&gt;JahiaAdministrationResources&lt;/key&gt;     &lt;file&gt;JahiaAdministrationResources&lt;/file&gt;   &lt;/resource-bundle&gt;   &lt;resource-bundle&gt;     &lt;key&gt;JahiaMessageResources&lt;/key&gt;     &lt;file&gt;JahiaMessageResources&lt;/file&gt;   &lt;/resource-bundle&gt;    &lt;resource-bundle&gt;     &lt;key&gt;CORPORATE_PORTAL_TEMPLATES&lt;/key&gt;     &lt;file&gt;jahiatemplates.Corporate_portal_templates&lt;/file&gt;   &lt;/resource-bundle&gt;   &lt;resource-bundle&gt;     &lt;key&gt;CORPORATE_PORTAL_TEMPLATES_V2&lt;/key&gt;     &lt;file&gt;jahiatemplates.Corporate_portal_templates_v2&lt;/file&gt;   &lt;/resource-bundle&gt; &lt;/registry&gt; </pre> <p>Then for a <code>bundleKey</code> of <code>CORPORATE_PORTAL_TEMPLATES</code>, the key/value lookup will be in <code>Corporate_portal_templates.properties</code> (located in directory <code>\WEB-INF\classes\jahiatemplates</code>) for the default language or for example in <code>Corporate_portal_templates_de.properties</code> if the current user's locale is</p>

Attribute or property name	Description
German.	Both <code>titleKey</code> and <code>bundleKey</code> cannot be found as arguments in the <code>declareContainer</code> API signatures. You have to use the <code>ResourceBundleMarker.drawMarker(bundleKey, titleKey, title)</code> method and pass the result to the <code>title</code> argument of the <code>declareContainer</code> API.
<code>windowSize</code>	It is possible to allow the user to change the number of items per page used in the pagination of a given container list. Say, if a container list contains 1000 containers/fields, you'll want to spread this list across multiple pages. If you don't, then enjoy the wait... The default value is -1 meaning the functionality is deactivated.
<code>windowOffset</code>	This attribute dictates the initial number of pages into the paginated list for this <code>containerList</code> . The default value is 0, which should cover most cases. There is little incentive to set this attribute since you'll want to initially display to the user the first page of the list, and not say the 4th page which seems unintuitive. On the other hand <code>containerList</code> 's <code>windowOffset</code> attribute is helpful since it allows you to specify the pagination offset at display time. Note that if <code>windowOffset</code> is superior to the number of elements in the container list, it defaults back to a zero value.
<code>validatorKey</code>	Name of the rule-set, which need to be applied for validating the container and its fields. As we are using Struts Validator, this key corresponds to the <code>form</code> element's <code>name</code> attribute in the <code>validation.xml</code> file.
<code>containerBeanName</code>	This attribute is used by the validator framework and sets the name of the class or interface, which provide the getter methods for the fields of the container. For further information about field input validation see section Field Input Validation.
<code>aliasNames</code>	This is a comma separated list of alias names. This way containers with different names are defined to be compatible. It can be used by the import/export service or the content copies (content picking, copy/paste,...). If source and destination containers have the same alias, import or content copying will be allowed.
<code>containerListType</code>	The container list types are defined in the <code>JahiaContainerDefinition</code> class: STANDARD_TYPE (0)      default type SINGLE_TYPE (1)      the container list can hold only one container MANDATORY_TYPE (2)    if the container list is empty, a new container is automatically created SINGLE_MANDATORY_TYPE (SINGLE_TYPE   MANDATORY_TYPE)

**Table 4: Attributes and properties for container definition**

Most of these attributes are stored as container definition properties. You could also set them by using the `declareContainerListProp` tag or use the `declareContainer` API with the `containerDefProperties` argument. These properties are simply name/value pairs, which can then be retrieved with the `JahiaContainerDefinition.getProperty` or `JahiaContainerDefinition.getProperties` methods. For your custom properties or for future properties this way will be used, instead of extending the taglib or API signature.

In the next example we show a container list declaration (using the taglib) that includes two properties named `columns` and `rows` that could then be used to render the container list in for example using a table.

```
<content:declareContainerList name='mainContentContainerList' title='Main Content'>
  <content:declareContainerListProp name='columns' value='3'/>
  <content:declareContainerListProp name='rows' value='2'/>
  <content:declareContainer>
    <content:declareField name='title' title='Title' type='SmallText'/>
    <content:declareField name='content' title='Content' type='BigText'/>
  </content:declareContainer>
</content:declareContainerList>
```

```

        value='Enter text here'/>
    <content:declareField name='image' title='Image' type='File'/>
    <content:declareField name='align' title='Image align' type='SharedSmallText'
        value='<jahia_multivalue[left:right:default]>left'/>
    <content:declareField name='date' title='Date' type='Date'/>
</content:declareContainer>
</content:declareContainerList>

```

## 2.1 SUB-CONTAINERS

We now extend the first example from the previous section by inserting the `fileContainerList` container list declaration inside the `simpleContainerList` container list. `fileContainerList` contains a single `File` field type. This means that each container in `simpleContainerList` will contain two text fields and zero or more files.

### TAGLIB VERSION:

```

<content:declareContainerList name='simpleContainerList' title='Simple container list'>
  <content:declareContainer>
    <content:declareField name='title' title='Title' type='SmallText'/>
    <content:declareField name='content' title='Content' type='BigText'
      value='Enter text here'/>
    <content:declareContainerList name='fileContainerList' title='File container list'>
      <content:declareContainer>
        <content:declareField name='file' title='File' type='File'/>
      </content:declareContainer>
    </content:declareContainerList>
  </content:declareContainer>
</content:declareContainerList>

```

### JAVA API VERSION:

```

// fileContainerList declaration
// fields declaration
jData.containers().declareField( "file", "File", FieldTypes.FILE, "" );

// vector to store the fields
Vector fileContainerFields = new Vector();
fileContainerFields.add( "file" );

// container list declaration
jData.containers().declareContainer( "fileContainerList", "File container list",
    fileContainerFields );

// simpleContainerList declaration
// fields declaration
jData.containers().declareField( "title", "Title", FieldTypes.SMALLTEXT, "" );
jData.containers().declareField( "content", "Content", FieldTypes.BIGTEXT,
    "Enter text here" );
;
// vector to store the fields
Vector simpleContainerFields = new Vector();
simpleContainerFields.add( "title" );
simpleContainerFields.add( "content" );
simpleContainerFields.add( "fileContainerList" ); // <- here we are...

// container list declaration
jData.containers().declareContainer( "simpleContainerList", "Simple container list",
    simpleContainerFields );

```

## 2.2 BOXES

In Jahia's packaged templates, you might have noticed that we have declared container lists called `box`, that seem to be able to contain different types of content. Actually here we use a trick to make it seem that way: we declare a container list with a field of type `SmallText`, that is initialized using a drop down list of box types. We also declare multiple sub container lists, one for each box type, and only display the one that corresponds to the value in the `SmallText` box type field.

## 3 FIELDS AND FIELD TYPES

We have already shown examples of field declarations in the previous sections, but here we will focus on the general syntax. Notice that fields should only be declared within containers. In previous Jahia versions we also allowed to declare fields outside of a container structure. This is now only possible for compatibility reasons, but we have deprecated this and will not allow it anymore in future versions of Jahia.

Declaring a field looks like this:

### TAGLIB VERSION

```
<content:declareField name='FIELDNAME' title='GUITITLE' type='TYPENAME'
  value="DEFAULTVALUE" />
```

### JAVA API VERSION





```
jData.containers().declareField("CONTAINERNAME", "FIELDNAME", "GUITITLE",
  "FIELDTYPE", "DEFAULTVALUE");
```

where :

- `FIELDNAME` is a unique name (see section Declaration Scope). Avoid using spaces and special characters (like [ ] &) in this name as the import/export utility uses the field names as element names in the XML file.
- `GUITITLE` is a string containing the name you want to display to the end user in the edition popup window.
- `TYPENAME` is used only in the taglib and may be one of the following: `Application`, `BigText`, `Boolean`, `Color`, `Date`, `File`, `Float`, `Integer`, `Page`, `SharedSmallText` and `SmallText`
- `FIELDTYPE` is the API counterpart to `TYPENAME` and should be one of the following:
  - `FieldTypes.APPLICATION`, `FieldTypes.BIGTEXT`, `FieldTypes.BOOLEAN`, `FieldTypes.COLOR`, `FieldTypes.DATE`, `FieldTypes.FILE`, `FieldTypes.FLOAT`, `FieldTypes.INTEGER`, `FieldTypes.PAGE`, `FieldTypes.SMALLTEXT_SHARED_LANG`, `FieldTypes.SMALLTEXT`
- `DEFAULTVALUE` is a string that contains the default value the field should have before the user starts editing it. The default value format will depend on the type of field you are declaring. Default values may contain some advanced markers that allow for some powerful customization. Notice that the default value should be a constant value and should rarely change. If you want to use dynamic values, you should either use expression markers (described in the next section Default Values) or set the default value in event listeners (see section Event Listeners)
- `CONTAINERNAME` is used only in the API and should be the name of the container structure definition name we will include it in.

These were again just the basic and most used attributes. There are also other possible attributes and properties in the taglib and API, which are described in the following table:





Attribute or property name	Description
titleKey	<p>This key uniquely identifies the locale-specific field title in the associated <code>bundleKey</code> bundle appropriate for the current user's locale.</p> <p>Note that if <code>titleKey</code>'s value cannot be found, the value in the <code>title</code> attribute (if defined) is used instead. If <code>titleKey</code> and <code>title</code> are both defined, then precedence is given to <code>titleKey</code>.</p>
bundleKey	<p>Resource bundles contain key/value pairs for specific locales and are used for internationalization (see section Internationalization).</p> <p>Note that Jahia finds the <code>*.properties</code> file associated to a given <code>bundleKey</code> by doing a lookup in <code>resourcebundles_config.xml</code> located in <code>\WEB-INF\etc\config</code>. The <code>resourcebundles_config.xml</code> file is generated automatically during template deployment. For example, given the following <code>resourcebundles_config.xml</code>:</p> <pre data-bbox="464 726 1432 1402"> &lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;registry&gt;   &lt;!-- core resources --&gt;   &lt;resource-bundle&gt;     &lt;key&gt;JahiaEnginesResources&lt;/key&gt;     &lt;file&gt;JahiaEnginesResources&lt;/file&gt;   &lt;/resource-bundle&gt;   &lt;resource-bundle&gt;     &lt;key&gt;JahiaAdministrationResources&lt;/key&gt;     &lt;file&gt;JahiaAdministrationResources&lt;/file&gt;   &lt;/resource-bundle&gt;   &lt;resource-bundle&gt;     &lt;key&gt;JahiaMessageResources&lt;/key&gt;     &lt;file&gt;JahiaMessageResources&lt;/file&gt;   &lt;/resource-bundle&gt;    &lt;resource-bundle&gt;     &lt;key&gt;CORPORATE_PORTAL_TEMPLATES&lt;/key&gt;     &lt;file&gt;jahiatemplates.Corporate_portal_templates&lt;/file&gt;   &lt;/resource-bundle&gt;   &lt;resource-bundle&gt;     &lt;key&gt;CORPORATE_PORTAL_TEMPLATES_V2&lt;/key&gt;     &lt;file&gt;jahiatemplates.Corporate_portal_templates_v2&lt;/file&gt;   &lt;/resource-bundle&gt; &lt;/registry&gt; </pre> <p>Then for a <code>bundleKey</code> of <code>CORPORATE_PORTAL_TEMPLATES</code>, the key/value lookup will be in <code>Corporate_portal_templates.properties</code> (located in directory <code>\WEB-INF\classes\jahiatemplates</code>) for the default language or for example in <code>Corporate_portal_templates_de.properties</code> if the current user's locale is German.</p> <p>Both <code>titleKey</code> and <code>bundleKey</code> cannot be found as arguments in the <code>declareField</code> API signatures. You have to use the <code>ResourceBundleMarker.drawMarker(bundleKey, titleKey, title)</code> method and pass the result to the <code>title</code> argument of the <code>declareField</code> API.</p>
valueKey	<p>This is the internationalized default value that is displayed if no value is defined for it in the database.</p> <p>Note that if <code>valueKey</code>'s value cannot be found in the associated <code>bundleKey</code> bundle, the value in the <code>value</code> attribute (if defined) is used instead. If <code>valueKey</code> and <code>value</code> are both defined, then precedence is given to <code>valueKey</code>.</p>

Attribute or property name	Description
aliasNames 	This is a comma separated list of alias names. This way fields with different names are defined to be compatible. It can be used by the import/export service. If source and destination fields have the same alias, import will be allowed. We also use it for setting the validation rules for field input validation. With using alias names you could define the rules just once and use the same definition for several similar containers and fields.
scoreBoost 	The search score boost factor attribute.
indexableField 	<code>false</code> if the field should not be indexed, <code>true</code> by default.
readOnly 	<code>true</code> if the field is just read only, <code>false</code> by default.

**Table 5: Attributes and properties for field definition**

Most of these attributes (except the title and default value) are stored as extended properties of field definition. You could also set them by using the `jData.containers().declareFieldDefProp()` API. These properties are simply name/value pairs, which can then be retrieved with the `JahiaFieldDefinition.getProperty()` or `JahiaFieldDefinition.getProperties()` methods. You can use that also for some own custom properties.

There are already some extended properties, which are implemented in Jahia:

Attribute or property name	Description
<code>JahiaFieldDefinitionProperties.FIELD_UPDATE_JSP_FILE_PROP</code> 	Custom JSP file to be used by add/update containers engines in place of default ones to edit / display a particular field.
<code>JahiaFieldDefinitionProperties.FIELD_MULTILINE_SMALLTEXT_PROP</code> 	This property can be used for <code>SmallText</code> and <code>SharedSmallText</code> fields. Set the value to <code>true</code> , if you want to display a multi-line input field instead of a single-line.
<code>JahiaFieldDefinitionProperties.PAGE_SELECTION_FILTER_PROP</code> 	Set a selection filter for page fields, to filter the list of pages displayed or being selectable for linking or moving pages (see section Limiting Pages to be Linked or Moved how to implement page selection filters). The default filter used is the <code>LimitedTemplatesFilter</code> , which considers the list of templates defined in the default value of the page field.
<code>JahiaFieldDefinitionProperties.FIELD_STYLESHEET_ID_PROP</code> 	This property can be used for <code>BigText</code> fields. You can set the ID of a stylesheet and a style definition description configured in <code>htmleditors_config.xml</code>

**Table 6: Extended properties for field definition**

### 3.1 FIELD TYPES

Fields may have different behaviors due to multi-language support in Jahia: some fields may have their values translated into different languages. For example, a menu title `SmallText` field will contain the string *Summary* for the English version and *Sommaire* for the French version. Other fields will use only one value displayed regardless of the language currently selected in the user's browser. Here is a complete list of the field types available in Jahia, including their multi-language behaviors :

Field Type	Description	Multi-lingual
<code>SmallText</code>	Most simple, and probably most useful, type of field. It may contain up to 255 characters <sup>3</sup> . Advanced uses of this field include the possibility to define a drop-down list of elements for the user to choose from, or using resource bundle or expression markers	Yes
<code>SharedSmallText</code>	Exactly the same as a <code>SmallText</code> field, except that the value of the field is shared in all languages.	No
<code>BigText</code>	Same as the <code>SmallText</code> field, except that the value is stored either on the file system or in a separate database table, so that the value may be unlimited in size.	Yes
<code>Integer</code>	<i>This field type will be able to hold positive natural numbers, their negatives and the number zero. The valid range is from <math>-2^{31}</math> to <math>2^{31}-1</math>.</i>	No
<code>Float</code>	<i>This field type can hold floating point numbers. The valid range is from <math>2^{-149}</math> to <math>(2-2^{-23}) \cdot 2^{127}</math>.</i>	No
<code>Boolean</code>	Using this field type, a check box will be presented. When the check box is checked, the value will be <code>true</code> , otherwise <code>false</code> .	No
<code>Color</code>	<i>With this field type a color palette will be displayed, where the editor will be able to pick a color.</i>	No
<code>Date</code>	<i>This field type can hold a date and/or time value. A calendar will be displayed, where the editor will be able to pick a date and enter a time.</i>	No
<code>File</code>	<i>A file manager will be displayed, where the editor will be able to upload files to the repository and connect the field with a file in the repository.</i>	Yes
<code>Page</code>	<i>With this field type you will either be able to create new pages, link to existing internal pages in the system, to external pages via URL. The widgets connected to this field type, may also provide the possibility to move pages.</i>	Yes
<code>Application</code>	<i>Using this field type, registered portlets can be chosen to be integrated.</i>	No

Table 7: Jahia field types

### 3.2 DEFAULT VALUES

The string that content designers can use as default values for fields might include a lot of options, and most of them will be covered in different sections (such as Expressions for expression markers, and Resource Bundle Tags for resource bundle markers), but we will now briefly present each marker.

A marker is very similar to an HTML tag, it is an enclosure for a string, that tells Jahia that it is not handling a regular string default value, but rather that the GUI engine must process the expression.

<sup>3</sup> The 255 character limit actually depends on the encoding used, and the way the database back-end stores the encoding. Some databases will treat this limit as a byte limit instead of a character limit, such as is the case for Oracle, limiting the length of input to a lower value.

### 3.2.1 SINGLE SELECTION CHOICE LIST MARKERS

The single selection choice list marker is used to display a list of items to select from. If such a marker is set as default value, users will be restricted to choose only a value listed in the item list. In the field edition window, the user will typically be presented with a selection widget of the following nature:

 SharedSmallText\_single



Picture 2.1.: Single Selection List

The following notation is used to define such a marker:

- `value="<jahia_multivalue[value1:value2:value3]>"` (displays three choices for this field and allows the user to only pick one)
- `value="<jahia_multivalue_single[value1:value2:value3]>"` (ditto)
- `value="<jahia_multivalue[value1:value2:value3]>value1"` (ditto but with `value1` as the default selection)

The `jahia_multivalue` marker is supported by the `Integer`, `Float`, `SharedSmallText` and `SmallText` field types.

### 3.2.2 MULTIPLE SELECTION CHOICE LIST MARKERS

This marker is the same as the above multi-value marker except that the user can select one or more items from the list as depicted below:

 SharedSmallText\_multiple



Picture 2.2.: Multiple Selection List



The notation therefore becomes:

- `value="<jahia_multivalue_multiple[value1:value2:value3]>"` (displays three choices for this field and allows the user to pick one or more)
- `value="<jahia_multivalue_multiple[value1:value2:value3]>value1"` (ditto but with `value1` as the default selection)

The `jahia_multivalue` marker is supported by the `Integer`, `Float`, `SharedSmallText` and `SmallText` field types.

### 3.2.3 RESOURCE BUNDLE MARKER

This marker is used to retrieve the default field value from a resource bundle. Further details are available in the section Internationalization.

The syntax is as follows:

```
value='<jahia-resource id="MySiteResource" key="product.001" default-value="Crew"/>'
(insets the text from key product.001 of resource bundle MySiteResource and defaults to Crew if it cannot find it).
```

### 3.2.4 EXPRESSION LANGUAGE MARKER

The Expression Language marker is a tiny script which is evaluated to determine the default value of a field. It is compatible with Apache's Java Expression Language (JEXL) specification and more details are available in the section Expressions.

The syntax is as follows:

```
value='<jahia-expression expr="currentUser.username" storeMarker="false"/>'
(retrieves the current user name. If storeMarker was set to true, the expression would not be resolved and stored uninterpreted)
```

### 3.2.5 DATE / TIME FIELDS

In the case of the date field, we use another default value marker to specify the date format used to display the date value in the templates. In the edition popup the date field will be presented in a default format.

For example if you want to display dates in this form: `25 Mar 2003 / 15:34`, you need to define this format: `dd.MMM.yyyy / HH:mm`. In order to do this set the following marker for the default field value:

```
value="<jahia_calendar[dd.MMM.yyyy / HH:mm]>"
```

The format syntax is defined in the `SimpleDateFormat` class in Java. See <http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html>



In previous versions it was not possible to set the default date via the definition, but only in an event listener. Since Jahia 4.2 and Jahia 5.0 SP3 it is possible to set the default date in the definition right after the `<jahia_calendar>` tag.

You can either set a specific date, like this:

```
value="<jahia_calendar[dd.MM.yyyy / HH:mm]>31.12.2999 / 23:59"
```

or you can use a JEXL like expression, where the variable "now" is standing for the current system date, when the user opens and thus edits the field.

```
value="<jahia_calendar[dd.MM.yyyy / HH:mm]>now"
```

You could also use an expression like `now + (1000*60*60*24*7)` to add 7 days to the current date, when the user is adding the field.

### 3.2.6 PAGE FIELDS

In case of page fields you have also a possibility to control the options in the page edit view of the engine popup. For instance you can limit the templates list, from which the editor will be able to choose from. Within square brackets you can pass a comma separated list of template names.

You can also set one or more of the following keywords:







Keyword	Description
pageonly 	This will disable the link and move page options (unless other keywords are explicitly set) and only enable the option to create a new page.
linkonly 	This will disable the create and move page options (unless other keywords are explicitly set) and only enable the options to create a page link (internal or external).
move 	This keyword will enable the option to select existing Jahia pages to be moved to the current position in the site.
internal 	This will enable the option to create links to internal pages.
external 	This will enable the option to create links to external pages.
notitleoverwrite 	In case you use <code>linkonly</code> and <code>internal</code> , you could also add <code>notitleoverwrite</code> , what will remove the input field for the page title, as it will be automatically copied from the linked page

Table 8: Page field keywords

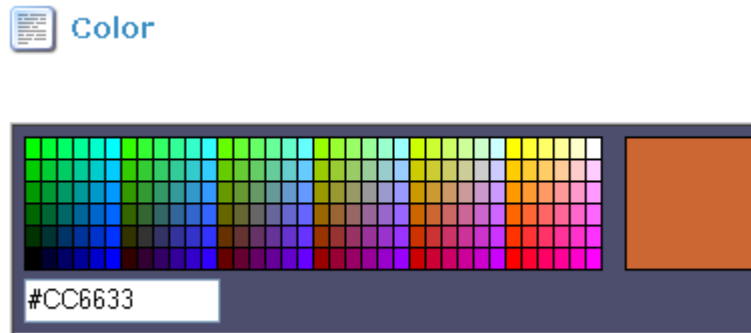
Here is an example:

```
value="<[Simple,Double] pageonly, move>"
```

That way the editor will only be able to choose between the templates named `Simple` and `Double`, and the options will only be to either create or move a page. When you select to move, then in Jahia 5.0.x the templates are not taken into account, and you will still have all pages selectable. In Jahia 4.2 and in Jahia 5.1 you will also on `move` and `link` only be able to move or link those pages, which are based on the templates `Simple` and `Double`, or you will be able to set your own `SelectPage-filter` (see section Limiting Pages to be Linked or Moved)

### 3.2.7 COLORS

You can set default colors with using the 6-digits hexadecimal color code after a hash (e.g. #FFFFFF for white, or #000000 for black). Jahia will display a color palette and will preselect the given color.



Picture 2.3.: Color Field

### 3.3 METADATA



[TODO DESCRIPTION How to define new metadata fields, how to create them, access them,... see: [http://lists.jahia.org/pipermail/template\\_list/2007-August/000040.html](http://lists.jahia.org/pipermail/template_list/2007-August/000040.html)]

### 3.4 CATEGORIES

Jahia also support a powerful categorization system. You can categorize your content according to different axis. Permissions can be setup on every node of your category tree.



[TODO DESCRIPTION]

### 3.5 PORTLETS



Jetspeed as portlet engine – JSR 168 – see own documentation: [https://www.jahia.net/jahia/webdav/site/jahia\\_net/shared/Documentation/ENG\\_JahiaPortletGuide\\_v1.0.pdf](https://www.jahia.net/jahia/webdav/site/jahia_net/shared/Documentation/ENG_JahiaPortletGuide_v1.0.pdf)

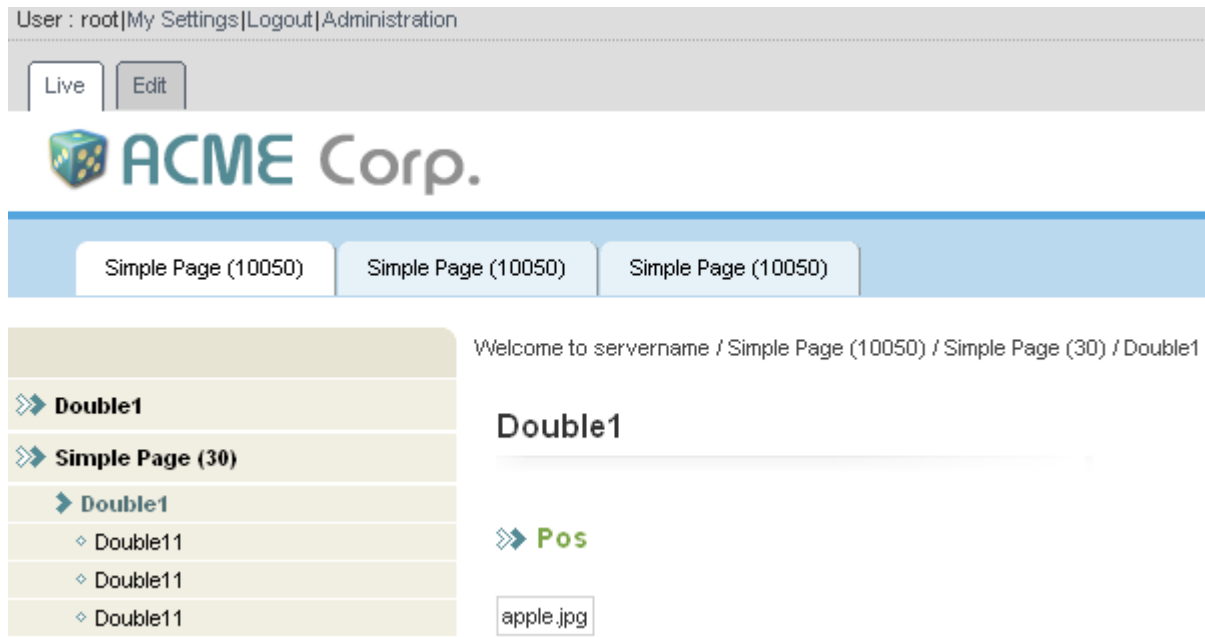
[TODO DESCRIPTION]

## 4 SITE STRUCTURE

Before getting down to the details of how to build sub pages navigation, we must first insist on some points that were presented in the chapter Introduction to Template Development of this documentation.

If you are familiar with other CMS systems, usually they use some kind of *navigation* object to navigate through the structure of pages. This navigation object is usually quite strict in the meaning, that it may only contain pages and maybe some metadata.

Jahia on the other hand does not use a specific navigation object to store pages, but rather allows the content designer to use Jahia pages as a field type embedded in a container.



Picture 2.4.: Example of a page hierarchy

So in order to present navigation elements in a page, you must declare a container list, that will contain page fields that represent the children pages of the current page.

The container structure is simple, containing only a field of type `Page`, but we could have also added other fields, in order for example to attach an image file to a page (for a small icon), or a field for storing metadata about the page, or even date fields to specify page publication time and expiration date.

#### TAGLIB VERSION

```
<!-- ===== Start Left menu ===== -->
<!-- start left menu declaration -->
<content:declareContainerList name='leftMenuContainerList' title='Left Menu'>
  <content:declareContainer>
    <content:declareField name='leftLink' title='Left Link' type='Page' />
  </content:declareContainer>
</content:declareContainerList>
<!-- end left menu declaration -->
```

## 4.1 LIMITING AVAILABLE TEMPLATES



[TODO DESCRIPTION See also chapter 3.2.6 Page fields]

## 4.2 LIMITING PAGES TO BE LINKED OR MOVED

The logo for Jahia, featuring the word "jahia" in a lowercase, sans-serif font. A small red diamond is positioned above the letter 'i'. This is followed by the text "SelectPage-Filter" and two arrows pointing right, one above the number "4.2" and one above the number "5.1".

[TODO DESCRIPTION]

## 4.3 GENERAL PAGES

[TODO DESCRIPTION]

## CHAPTER 3: CREATING CONTENT

Here you will find how to create the content, e.g. how to set the icons to call the edit popups, what possibilities you have to customize the view of the edit popups, and alternatively how to create content via input forms and via the Jahia API.

### 1 ACTION MENU

Jahia provides a tag that displays the related actions possible for a specific content object. This tag is called `actionMenu` and is located in the `jahiaHtml` tag library (because it actually renders HTML tags while executing). This tag will display the *add*, *update*, *delete* and other buttons for the containers.

The action menu tag looks like this :

#### TAGLIB VERSION

```
<jahiaHtml:actionMenu name='CONTENTOBJECTBEANNAME' namePostFix='NAMEPOSTFIX'  
  resourceBundle='RESOURCEBUNDLEKEY' useFieldSet='USEFIELDSET' />
```

#### JAVA API VERSION

```
jData.gui().html().drawBeginActionMenu("CONTENTOBJECTBEANNAME", null, null,  
  USEFIELDSET, "NAMEPOSTFIX", "RESOURCEBUNDLEKEY", null, out);  
...  
jData.gui().html().drawEndActionMenu("CONTENTOBJECTBEANNAME", null, null,  
  USEFIELDSET, "NAMEPOSTFIX", "RESOURCEBUNDLEKEY", null, out);
```

where

- `CONTENTOBJECTBEANNAME` is the name of the content bean object, whose action buttons we want to display.
- `RESOURCEBUNDLEKEY` is the name of the resource bundle to use for the display text of the buttons, to allow them to be available in multiple languages. With setting a `NAMEPOSTFIX` you could use action names specific to the content object. Otherwise or if you just want to display the default action names, then in the given bundle the keys `update`, `delete` and `add` must exist. From Jahia 5 onwards, you also need to define the values for the keywords `picker`, `picked`, `pickedlist`, `restore`, `copy`, `paste`, `editinword` and `source` to display the available actions.
- `NAMEPOSTFIX` may be used if we want to specify a postfix string to be appended to the key name for the action buttons. For example if the `namePostFix="Text"`, then the keys `updateText`, `deleteText` and `addText` will be accessed in the resource bundle in place of the default `update`, `delete` and `add` keys. The same applies to the new keywords in Jahia 5, which have mentioned above.
- `USEFIELDSET` must be `true` or `false` depending on whether the tag should use an HTML `<fieldset>` tag to display a border around the content object.

In the API version, you also have the possibility to set your custom action and lock icons, perhaps specific for a certain content object. If you like, you could also change the icons via CSS. The section Action Menu Customization shows, how this is done.

## 2 ENGINES AND EDIT-VIEWS

### 2.1 CONTROLLING THE ORDER AND GROUPING OF FIELDS

The fields in the engine edit views are ordered and grouped according to the field declaration order. Some field types require its own tab. These are: BigText, File, Page and Application. So Jahia starts grouping the fields on a tab, but as soon as one of these fields appear it creates a new tab to display one field. If there are more fields after that, they are placed and grouped together on a new tab again until there is a field, which requires its own tab.

Sometimes you will not want to have this automatic behavior, but will want to group the fields on your own. For this case you could use the ContainerEditView object.

As example let's assume you have for instance a site settings container with many fields, where you want to group the fields by logical groups. Here is an example of the container and field declarations:

```
<content:declareContainerList name="siteSettings" title="Site Settings"
  containerListType="<%=JahiaContainerDefinition.SINGLE_MANDATORY_TYPE%>">
  <content:declareContainer>
    <content:declareField name="footerText"
      title="Footer text"
      type="SharedSmallText"
      value="<%=defaultSettings.getFooterText()%>"
      titleKey="label.footerText"
      bundleKey="<%=resBundleID%>"/>

    <content:declareField name="titlePrefix"
      title="Title Prefix"
      type="SmallText"
      value="<%=defaultSettings.getTitlePrefix()%>"
      titleKey="label.titlePrefix"
      bundleKey="<%=resBundleID%> />

    <content:declareField name="pageWidth"
      title="Page width"
      type="SharedSmallText"
      value="<%=defaultSettings.getSizePage()%>"
      titleKey="label.pageWidth"
      bundleKey="<%=resBundleID%>"/>

    <content:declareField name="headerHeight"
      title="Header height"
      type="SharedSmallText"
      value="<%=defaultSettings.getHeaderHeight()%>"
      titleKey="label.headerHeight"
      bundleKey="<%=resBundleID%>"/>

    <content:declareField name="logo"
      title="Logo"
      type="File"
      value="<%=defaultSettings.getLogo()%>"
      titleKey="label.logo"
      bundleKey="<%=resBundleID%>"/>

    <content:declareField name="favicon"
      title="Favicon"
      type="File"
      value="<%=defaultSettings.getFavIcon()%>"
      titleKey="label.favIcon"
      bundleKey="<%=resBundleID%>"/>
```

```

<content:declareField name="colorPageBG"
  title="Page background color"
  type="SharedSmallText"
  value="<%=defaultSettings.getColorPageBG()%>"
  titleKey="label.colorPageBG"
  bundleKey="<%=resBundleID%>"/>

<content:declareField name="fMenuFont"
  title="Menu font"
  type="SharedSmallText"
  value="<%=defaultSettings.getMenuFont()%>"
  titleKey="label.menuFont"
  bundleKey="<%=resBundleID%>"/>

<content:declareField name="colorMenuText"
  title="Menu text color"
  type="SharedSmallText"
  value="<%=defaultSettings.getColorMenuText()%>"
  titleKey="design.colorMenuText"
  bundleKey="<%=resBundleID%>"/>
</content:declareContainer>
</content:declareContainerList>

```

In order to group the fields by customizing “sizes”, “texts”, “fonts”, “colors” and then all the file fields for capturing the “images” of the site, you could use the following code for ordering and grouping the fields according to your wishes:

```

<%
if (jData.gui().isEditMode()) {
  JahiaContainerSet cs = jData.containers();

  ContainerEditView cev = new ContainerEditView();

  //---- Sizes
  ContainerEditViewFieldGroup cevfg =
    new ContainerEditViewFieldGroup("sizes",
      getResourceBundle("label.sizes", "Sizes", jData), "");

  cevfg.addField("pageWidth", "");
  cevfg.addField("headerHeight", "");
  cev.addFieldGroup(cevfg);

  //---- Texts
  cevfg = new ContainerEditViewFieldGroup("texts",
    getResourceBundle("label.texts", "Texts", jData), "");
  cevfg.addField("footerText", "");
  cevfg.addField("titlePrefix", "");
  cev.addFieldGroup(cevfg);

  //---- Fonts
  cevfg = new ContainerEditViewFieldGroup("fonts",
    getResourceBundle("label.fonts", "Fonts", jData), "");
  cevfg.addField("fMenuFont", "");
  cev.addFieldGroup(cevfg);

  //---- Colors
  cevfg = new ContainerEditViewFieldGroup("colors",
    getResourceBundle("label.colors", "Colors", jData), "");
  cevfg.addField("colorPageBG", "");
  cevfg.addField("colorMenuText", "");
}

```



```
cev.addFieldGroup(cevfg);

//---- Images
cev.addField("logo",
    getResourceBundle("label.logo", "Logo", jData),
    "", "logo", "");

cev.addField("favicon",
    getResourceBundle("label.favicon", "Favicon", jData),
    "", "favicon", "");

cs.declareContainerEditView("siteSettings", cev);
}
%>
```

## 2.2 USING CUSTOM JSPs TO EDIT FIELDS

If you have requirements, which can not be achieved or customized with Jahia's default rendering of edit fields, you may also use your own JSPs, which you could either derive from the existing engine JSPs (located in folder `jsp\jahia\engines\shared`) or you can make your own JSPs.

You can set this JSP substitution in the field declaration by using code like this:

```
jData.containers().declareFieldDefProp("fieldName",
    JahiaFieldDefinitionProperties.FIELD_UPDATE_JSP_FILE_PROP,
    templatesPath + "fieldName_edition.jsp");
```

## 2.3 FIELD INPUT VALIDATION

This section describes how a template designer can define and implement field input validation rules, which will be checked before saving a container in the Jahia popup engine windows.

### 2.3.1 APACHE COMMONS VALIDATOR

To enable pluggable field input validation Jahia integrates Apache Commons Validator (<http://jakarta.apache.org/commons/validator/>). It uses the version, which is included with the packaged Struts release. This way Jahia also partially makes use of the Struts infrastructure (validator configuration, resource bundles,...).

You can find more information, hints and examples at the following links:

- <http://wiki.apache.org/jakarta-commons/Validator>
- <http://struts.apache.org/1.x/faqs/validator.html>
- There are two freely available book chapters on Validator (note that they were written for a slightly older version of the Validator library)  
[http://www.manning-source.com/books/husted/husted\\_ch12.pdf](http://www.manning-source.com/books/husted/husted_ch12.pdf)  
<http://www.oreilly.com/catalog/0596006519/chapter/ch11.pdf>

### 2.3.2 ADDING VALIDATIONS IN THE CONTAINER DECLARATION

You have to define a validator key and a container bean name at the same time you declare a container list. In the examples below, the validator key is called `peopleBean` and the bean name is `jahiatemplates.org.jahia.corporateportal.beans.PeopleBean`.

**TAGLIB VERSION:**


```
<content:containerList name="directoryPeopleContainer" validatorKey="peopleBean"
  containerBeanName="jahiatemplates.org.jahia.corporateportal.beans.PeopleBean">
  ...
</content:containerList>
```

**JAVA API VERSION:**

```
jData.containers().declareContainer( "directoryPeopleContainer", "People container",
  directoryPeopleFields , 5 , 0,
  "peopleBean", "jahiatemplates.org.jahia.corporateportal.beans.PeopleBean");
```

**2.3.3 PROVIDING A JAVA BEAN ACCORDING TO THE CONTAINER DECLARATION**

The Commons Validator framework expects that the objects to be validated are JavaBeans. As the container representation in the Jahia engine processing does not conform to the JavaBean-API yet, you can either

-  create an interface with all the getter methods to the field names in the container (or their alias names) and we will use introspection internally. If you specify the return type `JahiaMltHelper`, Jahia will not only return the value of the current language, but the value for all languages.

You can place the interface in the Java source directory of your templates.

For example:

```
public interface PeopleValidationBean {
    public String getDirectoryPeopleFirstName();
    public String getDirectoryPeopleLastName();
    public JahiaMLTHelper getDirectoryPeopleDescription();
    public String getDirectoryDepartmentName();
    public JahiaMLTHelper getDirectoryDepartmentNameMLT();
}
```

- or if you want to not only expose simple getters to the fields, but also other methods or combinations, then you could create a `JavaBean`, which extends the `ContainerValidatorBase` class and provides a constructor with the `ContainerFacadeInterface` and the `ParamBean` object as arguments. The field values should be taken from the `ContainerFacadeInterface` object and for that you should use the inherited methods `getJahiaField(String)` or `getJahiaMultiLanguageField(String)`.

You can place the bean in the Java source directory of your templates.

For example:

```
import org.jahia.data.containers.ContainerFacadeInterface;
import org.jahia.data.containers.ContainerValidatorBase;
import org.jahia.engines.validation.JahiaMltHelper;
import org.jahia.params.ParamBean;

public class PeopleValidationBean extends ContainerValidatorBase {
    public PeopleValidationBean(ContainerFacadeInterface newCf, ParamBean newParams) {
        super(newCf, newParams);
    }

    public String getDirectoryPeopleFirstName() {
        return getJahiaField("directoryPeopleFirstName");
    }
}
```

```
}
public String getDirectoryPeopleLastName() {
    return getJahiaField("directoryPeopleLastName");
}
public JahiaMltHelper getDirectoryPeopleDescription() {
    return getJahiaMultiLanguageField("directoryPeopleDescription");
}
public String getDirectoryDepartmentName() {
    return getJahiaField("directoryDepartmentName");
}
public JahiaMltHelper getDirectoryDepartmentNameMLT() {
    return getJahiaMultiLanguageField("directoryDepartmentName");
}
}
```

Whether you should provide a getter, which returns only the value in the current language (`String`) or the multilingual value (`JahiaMltHelper`) or both depends on the validation rule(s), which should be applied. Most of the standard validators expect the field value as a `String`, but there are Jahia specific multilingual validations, which need to get access to all language entries of a field and for those fields you should return a `JahiaMltHelper` object. If you want to apply both (standard validators and Jahia multilingual validators) to the same field you should provide two different getter-methods. In this special case, the getter method returning the `String` must match the field name property, while for the getter method returning the `JahiaMltHelper` you should append the suffix `MLT` to the field name.

### 2.3.4 DEFINING VALIDATION RULES

The validation rules are defined in an XML file. You can create a file for your templates (e.g. `corporateportal-validation.xml`). Currently the semantics of the XML file are form-based, but we can nevertheless use it solely with our JavaBean. For your bean validation rules you have to define a unique name in the `<form>` element, which is the same as the `validatorKey` in the container declaration. In the `<field>` element, you define all validation rules for the fields, which need to be checked. The field property must be the same as the field name in the container definition and must also match the property name in the getter-method of the interface or the JavaBean.

There are some standard built-in validations (in `validator-rules.xml`), which are described in the previously mentioned user guides for Apache Commons Validator.

In the example below three fields are defined as `required` and the email field will also be checked for a valid email format.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE form-validation PUBLIC
"-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.1.3//EN"
"http://jakarta.apache.org/commons/dtds/validator_1_1_3.dtd">
<form-validation>
  <formset>
    <form name="peopleBean">
      <field property="directoryPeopleFirstName" depends="required">
        <arg key="firstname.label" position="0"/>
      </field>
      <field property="directoryPeopleLastName" depends="required">
        <arg key="lastname.label" position="0"/>
      </field>
      <field property="directoryPeopleEmail" depends="required,email">
        <arg key="email.label" position="0"/>
      </field>
    </form>
  </formset>
```

```
</form-validation>
```

The element `<arg>` defines the key, which will be looked up in the message resource and is used to replace the placeholder in the error message (defined in `validator-rules.xml`). You can also define your own messages, which may have more placeholders. You don't have to use a message resource, but use hard coded text. The details about the possibilities are described in the Apache Commons Validator user guides.



There are also Jahia specific validations, which are defined in `validator-jahia-rules.xml`. These are:

- `requiredMandatoryLang` - validator to check whether values for all mandatory languages are set
- `maxLengthAllLang` - validator to check that the input for each language does not exceed a specified maximum length. Requires a `maxLength` variable.

```
<field property="directoryDepartmentName"
  depends="requiredMandatoryLang,maxLengthAllLang">
  <arg key="department.name.label" position="0"/>
  <arg name="maxLengthAllLang" key="${var:maxLength}" resource="false"
    position="1"/>
  <var>
    <var-name>maxLength</var-name>
    <var-value>30</var-value>
  </var>
</field>
```

- `minLengthAllLang` - validator to check that the input for each language isn't less than a specified minimum length. Requires a `minLength` variable.

```
<field property="directoryDepartmentName"
  depends="requiredMandatoryLang,minLengthAllLang">
  <arg key="department.name.label" position="0"/>
  <arg name="minLengthAllLang" key="${var:minLength}" resource="false"
    position="1"/>
  <var>
    <var-name>minLength</var-name>
    <var-value>3</var-value>
  </var>
</field>
```

-  `requiredMandatoryLangIfSet` - validator to check whether values for all mandatory languages are set, only if the text was set in at least one language
-  `requiredIfLinkValid` - validator to check whether page or link title has been set, only if selected option is not "No link" or "Reset link"

If you use the multi-language based Jahia specific validations you have to make sure that the field in the JavaBean returns a `JahiaMltHelper` object.

If you want to use standard validators and Jahia multilingual validators on the same field you should provide two getter methods. The property name of the getter method returning the `JahiaMltHelper` object should be passed as argument named `mltProperty` on position 0.

For example:

```
<field property="directoryDepartmentName" depends="required,requiredMandatoryLang">
  <arg key="department.name.label" position="0"/>
  <arg name="mltProperty" key="directoryDepartmentNameMLT" resource="false"
    position="0"/>
</field>
```

This example will ensure that the field is mandatory in the currently edited language and that it is filled in all languages, which are configured as mandatory.

### 2.3.5 CONFIGURING AND ENABLING THE VALIDATOR PLUG-IN

To enable the Validator you have to add the following message-resources and plug-in definition to the `struts-config.xml` file or you better create a separate Struts configuration file for your templates (e.g. `struts-config-corporateportal.xml`).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
  "http://struts.apache.org/dtds/struts-config_1_2.dtd">
<struts-config>
  <!-- Message Resources -->
  <message-resources parameter="ApplicationResources" null="false"/>
  <!-- Validator PlugIn -->
  <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property property="pathnames"
      value="/WEB-INF/etc/struts/validator-rules.xml,
            /WEB-INF/etc/struts/validator-jahia-rules.xml,
            /WEB-INF/etc/struts/corporateportal-validation.xml"/>
  </plug-in>
</struts-config>
```

This definition causes Struts to load and initialize the Validator plug-in. The `pathnames` property defines a comma-delimited list of Validator config files, which are loaded upon initialization.

The code snippet shows how to load the `ApplicationResources` file as message resource and for the validator plug-in the Apache Commons Validator standard built-in validations (`validator-rules.xml`), the Jahia specific validations (`validator-jahia-rules.xml`) and the validation definitions for the beans in the templates (`corporateportalvalidation.xml`).

If you created your own Struts configuration file, you have to add it to the `web.xml` configuration file:

```
<init-param>
  <param-name>config</param-name>
  <param-value>/WEB-INF/etc/struts/struts-config.xml,
              /WEB-INF/etc/struts/struts-config-corporateportal.xml
</param-value>
</init-param>
```

### 2.3.6 DISPLAYING VALIDATION ERROR MESSAGES

Validation of fields will only be performed after activating the Save button in the Jahia engines. If there are validation errors, a general message "There are some validation errors!" will appear in the page header and next to each field there will be a detailed message explaining why the input is not valid.

If the fields are spread over more pages and there are errors in the currently opened page, the page remains opened. If there are no validation errors in the currently opened page, then the first page with an validation error is opened.



Data

Tools

<b>Edit</b>	
<a href="#">Title(required)</a>	<div style="color: red; font-weight: bold; margin-bottom: 5px;">There are some validation errors !</div> <div style="margin-bottom: 5px;"> <b>Title(*)</b>  <span style="color: red;">Title is required.</span> </div>
<a href="#">Type</a>	
<a href="#">Layout</a>	
<b>Metadata</b>	
<b>Time Based Publishing</b>	

**Type**

**Layout**

Picture 3.1.: Validation error message

Note that no validation is performed, if you just change the current language. Validation is only performed on saving. This is why you should not use the validator `required` for multi language fields, as the `required` validator will just take into account the value of the current language. You should use the Jahia specific validator `requiredMandatoryLang`. Similar goes for `maxlength` and `minlength` validator.

Apache Commons Validator also allows for client-side validation, that means that JavaScript code is sent to the browser and the validation is performed before the request is sent to the server. This functionality is not supported in Jahia engines.

### 2.3.7 ADVANCED VALIDATION EXAMPLE

Here is an example with the `validwhen` validator:

This is the container declaration for a container list holding partners with their contact addresses.

```
<content:declareContainerList name="partnerContainer" title="Partner Container"
```

```

    validatorKey='partnerBean' containerBeanName='org.jahia.example.beans.Partner'>
<content:declareContainer>
  <content:declareField name="partnerName" title="Name"
    titleKey="label.partner.name"
    bundleKey="<%=resBundleID%>" type="SharedSmallText" />
  <content:declareField name="partnerAddressStreet" title="Street"
    titleKey="label.general.street"
    bundleKey="<%=resBundleID%>" type="SharedSmallText" />
  <content:declareField name="partnerAddressStreetNr" title="Street Nr."
    titleKey="label.general.street_number"
    bundleKey="<%=resBundleID%>" type="SharedSmallText" />
  <content:declareField name="partnerAddressZipCode" title="Zip Code"
    bundleKey="<%=resBundleID%>" type="SharedSmallText" />
  <content:declareField name="partnerAddressCity" title="City"
    titleKey="label.general.city"
    bundleKey="<%=resBundleID%>" type="SharedSmallText" />
  <content:declareField name="partnerAddressState" title="State"
    titleKey="label.general.state"
    bundleKey="<%=resBundleID%>" type="SharedSmallText" />
  <content:declareField name="partnerAddressCountry" title="Country"
    titleKey="label.general.country"
    bundleKey="<%=resBundleID%>" type="SharedSmallText" />
</content:declareContainer>
</content:declareContainerList>

```

Then you have to create the JavaBean to provide getter methods to the fields in the container.

```

/**
 * This interface is used to get the values for field validation.
 */
package org.jahia.example.beans;

public interface PartnerValidationBean {
    public String getPartnerName();
    public String getPartnerAddressStreet();
    public String getPartnerAddressStreetNr();
    public String getPartnerAddressZipCode();
    public String getPartnerAddressCity();
    public String getPartnerAddressState();
    public String getPartnerAddressCountry();
}

```

At last you need to specify the validation rules:

```

<form-validation>
  <formset>
    <form name="partnerBean">
      <field property="partnerName" depends="required, maxlength">
        <arg key="label.partner.name" position="0"/>
        <arg name="maxlength" key="{var:maxlength}" resource="false" position="1"/>
        <var>
          <var-name>maxlength</var-name>
          <var-value>30</var-value>
        </var>
      </field>
      <field property="partnerAddressStreet" depends="required">
        <arg key="label.general.street" position="0"/>
      </field>
      <field property="partnerAddressZipCode" depends="required">
        <arg key="label.general.zip_code" position="0"/>
      </field>
    </formset>
  </form-validation>

```

```

<field property="partnerAddressCity" depends="required">
  <arg key="label.general.city" position="0"/>
</field>
<field property="partnerAddressCountry" depends="required">
  <arg key="label.general.country" position="0"/>
</field>
<field property="partnerAddressState" depends="validwhen">
  <arg key="label.general.state" position="0"/>
  <var>
    <var-name>test</var-name>
    <var-value>((partnerAddressCountry != 'USA') or (*this* != null))</var-value>
  </var>
</field>
</form>
</formset>
</form-validation>

```

The validation rules define that the partner name is required and that it must not be longer than 30 characters. Furthermore the fields street, ZIP code, city and country are required. The field state is only required if the country is “USA”. This is done with the `validwhen` validator

## 2.4 FCK EDITOR TEMPLATES

[TODO DESCRIPTION chapter necessary?

See [http://wiki.fckeditor.net/Developer%27s\\_Guide/Configuration/Templates](http://wiki.fckeditor.net/Developer%27s_Guide/Configuration/Templates)]

## 3 MANAGING FORMS WITH JAHIA

This section describes how a template designer can easily use standard HTML forms to store submitted data directly into Jahia containers.

### 3.1 FORM HANDLERS

[TODO CODE EXAMPLES]



[TODO DESCRIPTION New File-upload feature]

The bulk of the data entered into Jahia is handled by popup-engines, but sometimes you will want to gather data directly from within a page; for example to create a guest book or news with blogging capabilities.

In the previous iterations of the Jahia tag library, saving user-submitted data to a Jahia container required excessive amounts of cryptic scriptlet code. From Jahia 4 onwards, a more elegant alternative was introduced to pipe user-submitted data directly into Jahia containers.

Jahia can automatically:

1. Process a submitted form
2. Map HTML form parameters to fields in a container
3. Set the appropriate permissions on the container
4. Add the container to the specified container list

To this end, the template developer simply declares a familiar `<form>` using standard HTML tags; an example of which is given here:



```
<form name="submitToContainerList" method="post">
  <input type="text" name="firstName">
  <input type="text" name="lastName">
  <input type="submit" name="submitToCList" value="Submit">
</form>
```

Next, we define the container list `addressList` where the submitted address data will be stored:

```
<content:declareContainerList name="addressList" title="Address list">
  <content:declareContainer>
    <content:declareField name="firstName" title="FirstName" type="SmallText" />
    <content:declareField name="lastName" title="LastName" type="SmallText" />
  </content:declareContainer>
</content:declareContainerList>
```

The final step is to notify Jahia where to store the submitted form data. This is done with a `formContentMapperHandler` tag which must be inserted before the above `<form>` declaration:

```
<content:formContentMapperHandler listName="addressList" submitMarker="submitToCList" />
```

The `formContentMapperHandler` tag defines the mapping between the `submitToContainerList` form and its associated container list `addressList`. The attribute `submitMarker` specifies which form on the page to link to and `listName` the associated container list. Its value can be set to any `<input>` tag of the form with a unique name (to the page) such as `submitToCList` in this case. If the `submitMarker` attribute is undefined, the form will not be processed.

Given the fields in the `submitToContainerList` form, the `formContentMapperHandler` tag expects to find the fields named `firstName` and `lastName` in the container list `addressList` on the current page. If this isn't the case, an error is generated.

The `formContentMapperHandler` tag supports additional helpful attributes:

- `immediatePublication`: This attribute specifies whether user-submitted containers should be immediately validated or left in staging mode. The default value is `false` i.e. it is saved in the staging mode only. The language is not specified so the content is automatically added in the browser's current language. If necessary, you can get around this by declaring the fields as shared type in the container (i.e. `type="SharedSmallText"`) so that the value will be independent of the browser's language.
- `storeAsUserName`: Since by default, the ACL on the stored containers will give Read/Write and Admin rights to user Guest, it is wise to restrict this level of access to only a single user. The following attribute allows this functionality by simply requiring its value to be set to the user which will be granted the said rights.
- `storeAsGroupName`: Same as `storeAsUserName` except that the ACL rights are given to the specified group, instead of a single user.

We conclude this section by showing the complete listing of a simple template demonstrating the form handling capabilities of Jahia. Notice how a JavaScript redirection is inserted to prevent multiple sequential posts of the same form. These occur frequently for example during user-initiated page reloads. Therefore, it is highly recommend to add this code snippet to your future form handling template.

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ page import="org.jahia.params.*" %>
```

```

<%@ page import="java.util.*" %>
<%@ page import="java.lang.*"%>
<%@ page import="java.util.*" %>
<%@ page import="org.jahia.content.*" %>
<%@ page import="org.jahia.data.*" %>
<%@ page import="org.jahia.data.beans.*" %>
<%@ page import="org.jahia.data.containers.*" %>
<%@ page import="org.jahia.utils.*" %>
<%@ taglib uri="/WEB-INF/etc/struts/struts-bean" prefix="bean" %>
<%@ taglib uri="/WEB-INF/etc/struts/struts-html" prefix="html" %>
<%@ taglib uri="/WEB-INF/etc/struts/struts-logic" prefix="logic" %>
<%@ taglib uri="/WEB-INF/etc/taglibs/jstl/c" prefix="c" %>
<%@ taglib uri="/WEB-INF/etc/taglibs/pager-taglib" prefix="pg" %>
<%@ taglib uri="jahiaHtmlLib" prefix="jahiaHtml" %>
<%@ taglib uri="JahiaLib" prefix="jahia" %>
<%@ taglib uri="contentLib" prefix="content" %>
<%
JahiaData jData = (JahiaData) request.getAttribute("org.jahia.data.JahiaData");
HashMap engineMap = (HashMap) request.getAttribute("org.jahia.engines.EngineHashMap");
ParamBean jParams = jData.params();
String savedCacheStatus = jParams.getCacheStatus();
jParams.setCacheStatus(ParamBean.CACHE_BYPASS);
String bypassUrl = jParams.composePageUrl(jData.page().getID());
jParams.setCacheStatus(savedCacheStatus);
%>

<content:declareContainerList name="testContainerList" title="Content Container list">
  <content:declareContainer>
    <content:declareField name="title" title="Title" type="SmallText"
      value="Lord of the Flies"/>
    <content:declareField name="bookformat" title="BookFormat" type="SharedSmallText"/>
    <content:declareField name="content" title="Content" type="BigText" />
  </content:declareContainer>
</content:declareContainerList>

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <content:i18n />
    <title>Add container with form</title>
  </head>
  <body>
    <logic:present parameter="submit">
<!-- this is required because we want to see the changes immediately
and also make sure we never post twice when the user does a page
reload. Also this should always be after the formContentMapperHandler tag
to make sure the processing was completed before redirecting.
-->
    <script language="javascript" type="text/javascript">
      <!--//--><![CDATA[//><!--
        window.location.href = '<%=bypassUrl%>';
      //--><![]]>
    </script>
  </logic:present>
  <h1>Add container with form</h1>
  <content:containerList name="testContainerList" id="testContainerList">
    <content:container id="testContainer">
      <h2>
        <content:textField name="title" defaultValue=""/>
        <content:textField name="bookformat" defaultValue=""/>

```

```
</h2>
<content:bigTextField name="content" defaultValue="" />
<br />
</content:container>
<hr />
</content:containerList>
<content:formContentMapperHandler listName="testContainerList"
    submitMarker="submitToTestContainerList" />
<form action="<%=bypassUrl%" name="submitToContainerList" method="post">
  <input type="text" name="title" value="Lord of the Flies" /><br />
  <input type="text" name="bookformat" /><br />
  <textarea name="content"></textarea><br />
  <input type="submit" name="submitToTestContainerList" value="Submit" />
</form>
</body>
</html>
```

## 4 CREATING CONTENT WITH JAHIA API

[TODO DESCRIPTION General pages – not hardcoding page Ids, template Ids]  
[TODO CODE EXAMPLES]

### 4.1 SETTING ACL RIGHTS

[TODO DESCRIPTION ACL setting via API]  
[TODO CODE EXAMPLES]



Since Jahia 5.0 SP3 ACL objects are not created for every content object, but just for the cases, where the ACLs are not inherited. You have to be careful if you were creating containers/fields by Jahia API.

### 4.2 SETTING CATEGORIES

[TODO DESCRIPTION Categories setting via API]  
[TODO CODE EXAMPLES]



Metadata categories

### 4.3 PUBLISHING CONTENT

[TODO DESCRIPTION also various workflows]  
[TODO CODE EXAMPLES]

# CHAPTER 4: ACCESSING CONTENT

This chapter covers the taglibs and API to access and display the content created with Jahia, also the sorting, filtering and paging of large container lists.

## 1 ACCESSING CONTAINER LISTS AND CONTAINERS

### 1.1 ACCESSING CONTAINER LISTS

Jahia has traditional ways of accessing some content objects, such as container lists, that have existed since version 2. They are mostly referred to as *absolute container lists* and *relative container lists*, but they really should be called: *absolutely addressed container lists* and *relatively addressed container lists*. The problem with the first names is that they seem to imply that it is a special type of container list, when it's not. The only thing that is *absolute* or *relative* is the way we access these container lists. Be aware that these access methods also have some limitations (that don't exist in the JavaBean API), which are presented below.

#### 1.1.1 ABSOLUTE ADDRESSING (AKA ABSOLUTE CONTAINER LISTS)

In order to understand absolute addressing, we must first make something clear: Jahia container list instances are local to a page. That means that if we have two container lists called `addressList` on two different pages, they are indeed different, even if they share the same definition (which is local to a template).

So 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:

```
<content:absoluteContainerList name="addressList" pageId="3">
  <!-- tags to display container list here -->
</content:absoluteContainerList>
```

##### JAVA API VERSION:

```
JahiaContainerList containerList =
    jData.containers().getAbsoluteContainerList("addressList", 3);
```

The above code will retrieve the container list named `addressList` on page 3.

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.

#### 1.1.2 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 `addressList` located on `page3`:

#### TAGLIB VERSION:

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

#### JAVA API VERSION:

```
JahiaContainerList addressList =  
  jData.containers().getRelativeContainerList("addressList", 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.

### 1.1.3 PAGE LEVEL ADDRESSING (TAGS ONLY)

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 `addressList` located on `page1`:

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

### 1.1.4 RESTRICTIONS

Due to internal restrictions of the Jahia legacy back-end, absolute and relative addressing of containers does **not** give access to sub container lists. So if you need to access a container list that has sub container lists, we recommend you access it using the JavaBean API technology described in the next section.

## 1.2 JAVA BEAN API

As described in section Common Variables and Jahia JavaBeans, Jahia puts some JavaBean compliant objects as attributes in each request object. From these objects, you can access all of Jahia's content. For example you can access a container list like this :

```
ContainerListBean = (ContainerListBean)  
  currentSite.getPage(3).getContainersLists().get("addressList");
```

There are a lot of possible combinations, but they make accessing content from anywhere quite straightforward. And as all these objects are JavaBean compliant, you can also access them using JSTL taglibs or Struts taglibs.

Another powerful example is the following :

```
List allNewsLists = currentSite.getAllContainerLists("newsList");
```

which retrieves all the container lists in a site that bear the name `newsList`. If the structure of these container lists is unified, you can then retrieve all the news entries on any page, and display for example the last 3 (combining with filters and sorts that we present in section Container List Sort, Search and Pagination).

The above call has one main drawback: performance. It loads up all the container lists, and looks also for the presence of sub container lists, which can put significant load on the database back-end. If you know that your news container lists will not contain any sub container lists, you can replace the above call with the following:

```
List allNewsLists = currentSite.getLightContainerLists("newsList");
```

Please note that if some of your container lists did indeed have sub container lists, this call will **not** give you access to them.

### 1.3 RE-USING CONTENT USING CONTAINER FILTERS

Another way of retrieving content that comes from different pages is to use container list filters. It is not presented now because we first need to introduce all the concepts of filtering, sorting and searching container lists first. Please refer to that section Container List Sort, Search and Pagination for more information on how to use these powerful tools to retrieve content from other pages.

### 1.4 CATEGORIES

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.

We will not present in this section how to use categories from an administration and user point of view. We assume that you are already familiar with the corresponding section of the Administration and User guides. So if you feel like you're not comfortable with these issues, now would be a good time to brush up on them.

An example of content classification is a book list, that needs to categorize the books. In this case the content manager would either re-use or input a classification tree and then use it to classify the books. Users of the system can then browse using the categories to view books, that are related to each other, or view books, that are in the same categories as the currently viewed book.

#### 1.4.1 BROWSING CATEGORIES

This usually involves looking at a specific root category and listing all the objects that are related to the category, such as sub-categories or other objects that are available in that category. One of the main open questions is where to start category browsing. We might have different entry points depending on the type of objects we are looking for in the categories. For example a category entry point might be different if we are looking at web applications or content objects, and the same goes for templates, etc...

##### RETRIEVING THE ROOT CATEGORY

First, we will show is how to retrieve the root category, which always exists:

```
<content:category key="root" id="rootCategory" subTreeID="categoryTree" />
```

The above tag is quite powerful. Basically what it does is :

- retrieves the root category in the backed
- stores the corresponding `CategoryBean` object under the `rootCategory` attribute in the `pageContext`
- stores the subtree for the root category in the `pageContext` attribute `categoryTree`. The class type for the tree object is `JTree`, which may be used with the `<jahiaHtml:tree>` tag for displaying.

## DISPLAYING THE CATEGORY TREE

Jahia provides a tag to render trees that use the `JTree` structure. This is a powerful tag that supports actions to fold/unfold nodes as well as selecting a node. We will not cover it in detail here, but rather just show some example code to display a category tree. You will note that we use the `categoryTree` attribute in the `pageContext` here as the `treeName` attribute. You can understand most of the attributes for the `<jahiaHtml:tree >` tag by looking at the usage in the displayed code:

```
<jahiaHtml:tree treeName="categoryTree" userObjectID="userObject"
    actionURL="<%=actionURL%>" nodeIndexID="nodeIndex"
    selectionParamName="selectednode"selectedNodeIndexID="selectedNodeIndex"
    selectionURLID="selectionURL" selectedUserObjectID="selectedUserObject">
<%
Object nodeInfo = pageContext.findAttribute("userObject");
Integer nodeIndexInt = (Integer) pageContext.findAttribute("nodeIndex");
int nodeIndex = nodeIndexInt.intValue();
String selectionURL = (String) pageContext.findAttribute("selectionURL");
Integer selectedNodeIndexInt = (Integer)
    pageContext.findAttribute("selectedNodeIndex");
int selectedNodeIndex = selectedNodeIndexInt.intValue();
Category curCategory = (Category) nodeInfo;
String catDisplay = curCategory.getTitle(jData.params().getLocale());
if (catDisplay == null) {
    catDisplay = "(key=" + curCategory.getKey() + ")";
}
if (nodeIndex == selectedNodeIndex) { %>
    <b><%= catDisplay %></b>
<%} else {%>
    <a href="<%=selectionURL %>"><%= catDisplay %></a>
<%}%>
</jahiaHtml:tree>
```

## DISPLAYING OBJECTS FOR THE CURRENTLY SELECTED CATEGORY

Now that we have displayed the category tree, we will want to add code to display the content of the currently selected category. First we retrieve from the `pageContext` the currently selected category, and then we load its corresponding objects using the `<content:category>` tag, as shown here :

```
<% Category selectedCategory = (Category)
    pageContext.findAttribute("selectedUserObject"); %>
<content:category key="<%=selectedCategory.getKey()%>" id="currentCategory" />
```

Now that this is done, here is an example of retrieval of children of the category that are of a specific type :

```
List of pages in category <bean:write name="currentCategory" property="title"/>:
<ul>
    <logic:iterate name="currentCategory" property="childsOfType(ContentPage)"
        id="curPageBean" >
        <%
            PageBean pageBean = (PageBean) pageContext.findAttribute("curPageBean");
            JahiaPage jahiaPage = pageBean.getJahiaPage();
            boolean isVisible = jahiaPage.checkReadAccess(jData.params().getUser());
```

```
if (isVisible) {%>
  <li><a href="
```

The above code displays links to all the objects of type `ContentPage` that are listed in the currently selected category.

For more advanced examples of displaying category objects the reader is referred to the packaged templates in Jahia, for example the `categories.jsp` and `document_listing.jsp` templates in the Corporate Portal Templates.

### 1.4.2 FINDING OBJECT CATEGORIES

It is also possible, from a content object, to retrieve all the categories it is part of. Basically this call looks like this:

```
Set categories = Category.getObjectCategories(contentObject.getObjectKey());
```

Unfortunately there is no tag equivalent, so this forces the use of scriptlets. You might also be wondering how to obtain the `contentObject` reference. Here is an example for a `PageBean` object :

```
JahiaPage jahiaPage = pageBean.getJahiaPage();
ContentPage contentObject = jahiaPage.getContentPage();
```

Another equivalent way of retrieving the content object could have been :

```
ContentPage contentObject = ContentPage.getPage(jahiaPage.getID());
```

Although a tad slower (only on first time access since caches are used), this is more general. In the same way you can also use :

```
ContentContainer.getContainer(int containerID);
ContentContainerList.getContainerList(int containerID);
ContentField.getField(int fieldID);
```

All these objects derive from the `ContentObject` class, on which you may call the `getObjectKey()` method.

### 1.4.3 FRONT-END CACHE ISSUES

When using categories, Jahia does its best to figure out changes in category assignment, but usually, as the logic of retrieval may be included in templates, it may not be aware of the latest changes, and therefore cannot invalidate the front-end cache for pages using categories. This is where cache expiration control becomes important for pages displaying category objects. See the section Cache Issues for details on working and setting cache expiration delays.

### 1.4.4 CONTAINER FILTERS AND CATEGORIES

It is also possible to use categories through the usage of container filters, making it possible to use the full extend of filtering, sorting and searching functionalities. This will be covered in the section Container List Sort, Search and Pagination.



## 1.5 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 as detailed below, while the slightly less functional JSTL EL and Struts EL can be used anywhere within the template.



The format of expressions is expected to change in a future release of Jahia. Users that choose to use this feature might have to update their templates when the new version comes out.

### 1.5.1 EXPRESSION LANGUAGE IN FIELD DECLARATIONS

Jahia is capable of using 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.

To this end, Jahia uses the JEXL (<http://jakarta.apache.org/commons/jexl/>) language to build expressions. JEXL's syntax is similar to Java, so it makes building expressions easy.



In the previous version of the Template Developer guide we showed you how to access content from other containers via JEXL. In the example we used the `getContainerByID()` method. The problem with this approach is, that container IDs are internal IDs and may change after import/export. So it is not a good idea to hardcode container IDs in your templates.

Here is now an example for a different JEXL expression :

```
currentUser.properties['firstname'] + ' ' + currentUser.properties['lastname']
```

Basically this is the equivalent of the following Java code :

```
currentUser.getUserProperties().get("firstname") + ' ' +
currentUser.getUserProperties().get("lastname")
```

### USING JEXL EXPRESSIONS

As we can see the difference with Java are minimal. Expressions may access the whole JavaBean API which we presented earlier.

In order to tell Jahia that we are inserting an expression instead of a value, we must build the expression within an expression marker, which we can build like this :

```
String expressionValue = ExpressionMarker.drawMarker(String expression);
```

We can then start using the expression value as a default value for a field. So if we resume, we build an expression using the following code :

```
String expressionValue = ExpressionMarker.drawMarker(
    "currentUser.properties['firstname'] + ' ' + currentUser.properties['lastname']");
```

We can then declare a field this way :

```
jData.fields().declareField( "expressionField", "Automatic value",
    FieldTypes.SMALLTEXT_SHARED_LANG,
    expressionValue );
```

where `expressionValue` evaluates to the following expression marker:

```
<jahia-expression expr="currentUser.properties['firstname'] + ' ' +
  currentUser.properties['lastname']" storeMarker="false"/>
```

Note that if `storeMarker` was set to `true`, the expression would not be resolved and stored uninterpreted. The thing to be careful about with that type of declaration is that when editing the field you will be able to modify the expression value, and if you modify it the expression will change. This can be very powerful but also not very user friendly.

Expression values can also be used in conjunction with multi-value fields, in which we want to populate the option list using expressions. Be warned though, that the expression in the field declaration must be constant and not dynamic. For instance, you would want to retrieve the multi-value options from fields in another container list, where the number of containers is always changing. If this causes your expression to change, because you have to access different container ids, then you should not use expressions, but rather solve such a requirement with event listeners.

## 1.5.2 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 section *Common Variables and Jahia JavaBeans*).

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:

```
...
//included by default in Jahia example template
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
...
<content:declareField name="popo" title="Popo title" type="SmallText" value="po po"/>
...
<br> Example EL usage:
<br>Username : <c:out value="${currentUser.username}"/>
<br>SiteID : <c:out value="${currentSite.id}"/>
<br>Homepage Title : <c:out value="${currentSite.homePage.title}"/>
<br>Homepage ID : <c:out value="${currentSite.homePage.id}"/>
<br>page ID : <c:out value="${currentPage.id}"/>
<br>faqCategoryContainer size : <c:out
  value="${currentPage.containerLists.faqCategoryContainer.size}"/>
<br>faqCategoryContainer size : <c:out
  value="${currentPage.containerLists['faqCategoryContainer'].size}"/>
<br>faqCategoryContainer name : <c:out
  value="${currentPage.containerLists['faqCategoryContainer'].name}"/>
<br>the first container's id : <c:out
  value="${currentPage.containerLists['faqCategoryContainer'].containers['0'].id}"/>
<br>faqTitle value: <c:out
  value="${currentPage.containerLists['faqCategoryContainer'].containers['0'].
  fields['faqTitle'].value}"/>
<br> languageStates : <c:out value="${currentPage.languageStates}"/>
```

```

<br>popo's value : <c:out value="\${currentPage.fields.popo.value}"/>
<br> popo's value : <c:out value="\${currentPage.fields['popo'].value}"/>
<br>name : <c:out value="\${faqCategoryContainerList.name}" />

<content:containerList name="faqCategoryContainer" id="faqCategoryContainerList">
  <content:container id="faqCategoryContainer">
    <h3>
      <content:textField name="faqTitle"/>
    </h3>
  </content:container>
  ...
</content:containerList> ...

```

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 `faqTitle` is the key in the `Map` to the targeted object it looks like this:

```

\${currentPage.containerLists['faqCategoryContainer'].containers[0].fields['faqTitle'].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.

### 1.5.3 EXPRESSION PERFORMANCE

Expressions are an extremely powerful way of re-using content, but they have a cost : performance. If you use a lot of expressions in a template, you will notice significant slowdown in some scenarios. In future versions of Jahia, this problem will be addressed, but probably at the cost of having to revise expression syntax.

## 2 CONTAINER LIST SORT, SEARCH AND PAGINATION

This section describes how a template designer can implement searching, filtering and sorting mechanisms on large data container lists directly within the template JSP file.

As we are focusing on large data container lists, a section is devoted to explain the new Container List Pagination mechanism, too.

### 2.1 SEARCH, FILTER AND SORT

[TODO DESCRIPTION How to create searchers on specific container lists/fields,...

We should also mention the existence of the log history table which is perhaps faster and better suited to make request such as "all the content changes occurred for the last 5 days".

We should also illustrate how to filter on metadata (e.g. all the content modified by shuber with category "DMS" for the last month". So here we should better explain the pros and cons of all possible methods we have within Jahia + their impact on performance or caches. ]

The figure below illustrates a container list of people with searching, filtering and sorting options. These options allow a user to define its search criteria.

Search:

filter:

result:  (Items/Page) [1 - 2] of 2

name	Skill	Level	Hourly Rate
FarewellJohn	Java/J2EE		150 USD
StevenSmith	Open Source		250 USD

addPeople  
 sort And Properties

Picture 4.1.: Search, filter, sort panel

### 2.1.1 JAHIA PAGE FORM

The Search, Filters and Sort options are HTML inputs that must be enclosed inside a HTML form.

More important is that these form values need to be sent to the currently displayed Jahia page. That is why the form action is an URL requesting the Jahia current page, and which bypasses the HTML front-end cache, so as to make sure the request gets processed by Jahia.

```
<form name="jahiapageform" action="/jahia/Jahia/pid/7/cache/off" method="POST">
...
</form>
```

#### JAHIA PAGE FORM TAG

Jahia provides a special tag named `jahiaPageForm`, which you can use to generate a HTML form, which action is an URL requesting the Jahia current page:

```
<content:jahiaPageForm name="jahiapageform">
...
</content:jahiaPageForm>
```

Attribute	Description	Mandatory
name	The form name. If not set, the default name is <code>jahiapageform</code> .	No
method	Should be <code>Post</code> or <code>Get</code> . The default value is <code>Post</code> .	No

Table 9: `jahiaPageForm` tag attributes

### 2.1.2 SEARCH OPTIONS

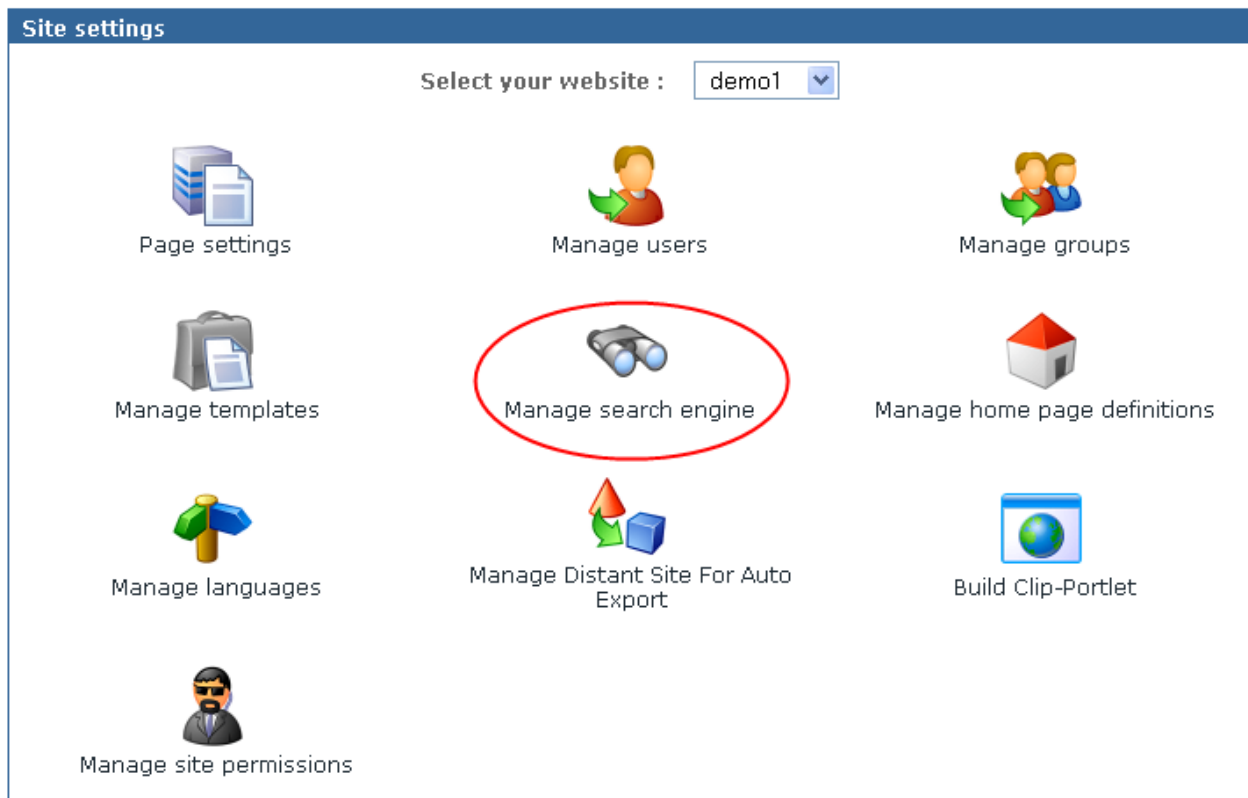
The search option allows the user to enter a search string used to match against containers of a given container list. The effective values used by the search engine are the values of text fields defined in the structure of containers of a given container list.

The search query can be a simple word or a more complex query, as long as it respects the syntax defined by the Apache Lucene Search Engine.

**SEARCH ENGINE**

This feature uses the built-in Search Engine based on the Apache Lucene Search Engine.

Within the Jahia Administration panels (in Site Mode) you must ensure, that the search index exists. You only need to create a search index once for each site.



Picture 4.2.: Manage search engine menu

**LINKING A SEARCH OPTION TO A CONTAINER LIST.**

To link a search option with a container list, simply name the search input as below:

"clistsquery\_" + container\_list\_name

Below is an example for the container list named `directoryPeopleContainer`:

```
<input type="text" name="clistsquery_directoryPeopleContainer" value="">
```

**SEARCH OPTION TAGS**

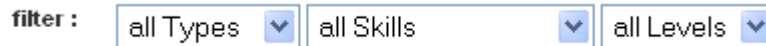
There are two tags that can help you to add the search option within the template:

```
<content:containerList name="directoryPeopleContainer">
  <input type="text" name="<content:ctnListSQueryInputName/>"
    value="<content:ctnListSQueryInputValue/>" />
</content:containerList>
```

The `ctnListSQueryInputName` tag automatically generates the correct Search option input name according to the enclosing container list, while the `ctnListSQueryInputValue` tag is used to populate the input value with the previous Search query entered by the user. As you can see, these tags must be enclosed within the `containerList` tag.

### 2.1.3 FILTER OPTIONS

Filters are typically select-box inputs containing several possible values, a user can choose to filter out large data container lists.



Picture 4.3.: Example with three filters

There are three filters in the figure above:

- by person's type: {employee,consultant}
- by person's skill: {Java,Perl,PHP,...}
- by person's level: {beginner,junior,senior,expert}

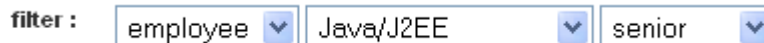
These filters are linked respectively to the fields: "people type", "people skill" and "people skill level".

But filters can be more complex, i.e :

- by date: { one day max, two days max, one week max, ... }
- by range value: { "<100", "100 to 1000", ... }

A filter is linked to one field of the container list at a time. When several filters are defined, the result of the overall filtering is a logical **AND** operation between these filters.

For example, the combination of these values,



Picture 4.4.: Example with choices from three filters

will display people, who are a Senior Java Developer with the status of Employee.

#### CONTAINER FILTER BEAN AND CONTAINER FILTER HANDLER.

Basically, you create a *Container Filter Bean* object (`ContainerFilterBean` class) for each filter (one filter per field). You then instantiate a *Container Filters Handler* with a vector of these Container Filter Bean.



The code below must be placed before the container list declaration!

1. Init the vector of filter beans:

```
// Our vector of container filter beans
Vector cFilterBeans = new Vector();
ContainerFilterBean containerFilter = null;
```

2. Retrieve user's skill choice. Here, the parameter name refers to the select box named `directoryPeopleSkill_filter`:

```
String selSkill = request.getParameter("directoryPeopleSkill_filter");
```

```
if ( selSkill == null ){
    selSkill = "All Skills"; // By default, select all Skills.
}
```

3. Instantiate the Filter Bean Object if needed, then store it in the vector of Filter Beans. The Filter Bean is created with the correct Field Name `directoryPeopleSkill`. Comparison clauses can then be added to the filter. If there are more than one comparison clause added to the filter, a logical OR operation is performed between the clauses of the filter.

```
if ( !selSkill.equals("All Skills") ) {
    // Create a filter only if needed.
    // The filter is created with the field name
    containerFilter = new ContainerFilterBean("directoryPeopleSkill");
    // Then we add a comparison clause , here an EQUAL comparison.
    containerFilter.addClause(ContainerFilterBean.COMP_EQUAL, selSkill);
}
if ( containerFilter != null ){
    // Add the Filter Bean to the vector
    cFilterBeans.add(containerFilter);
}
// reset the Container Filter Bean.
containerFilter = null;
```

4. Instantiate the Filters Handler with the Container List Name ( here `directoryPeopleContainer` ) and the vector of Filter Bean. Then store this Filters Handler in the request object.

```
// Now we create the Filters Handler, only if there is at least one
// available Filter Bean.
if ( cFilterBeans.size()>0 ) {
    ContainerFilters containerFilters =
        new ContainerFilters( "directoryPeopleContainer", jParams,cFilterBeans);
    // Store the list of filters in the request object.
    // It will be used later by the container list loader.
    request.setAttribute( "directoryPeopleContainer_filter_handler",
        containerFilters);
}
```

5. In this example, the People Skill Filter is linked to the select box named `directoryPeopleSkill_filter`

```
<select name="directoryPeopleSkill_filter">
  <option value="All Skills">All Skills</option>
  <option value="Java">Java</option>
  <option value="Perl">Perl</option>
  <option value="PHP">PHP</option>
</select>
```

#### LINKING A FILTERS HANDLER TO A CONTAINER LIST

In the Java listing above, we store the Filters Handler in the request object as an attribute named: `directoryPeopleContainer_filter_handler`.

This name respects the format used to link a filter handler with a container list:

**Container\_List\_Name + “\_filter\_handler”**

This handler will be detected by Jahia and it will be used to apply filtering, when loading the corresponding container list.

## 2.1.4 SORT OPTIONS

Containers can be sorted by one field at a time in ascending or descending order.

As with filters, you need to instantiate a *Sort Handler*, then store it in the request object.

Two information are required to initialize the sort handler:

- The Sort Field Name
- The Sort Order: ascending or descending

### SORT HANDLER

The listing below illustrates how to create the Sort Handler.



The code below must be placed before the container list declaration!

1. Retrieve the name of the Field on which to sort.

Here, this parameter is named `directoryPeopleContainer_sort` .

```
String peopleSort = request.getParameter("directoryPeopleContainer_sort");
if ( peopleSort == null ){
    peopleSort = "none"; // By default, no Sort required.
}
```

2. Retrieve the Sort Order.

Here, this parameter is named `directoryPeopleContainer_sort_order`.

```
String peopleSortOrder =
    request.getParameter("directoryPeopleContainer_sort_order");
if ( peopleSortOrder == null ){
    peopleSortOrder = "asc"; // By default set to Ascending.
}
```

3. Instantiate the Sort Handler if needed.

Set the wanted Order ( Asc, Desc ). By default a Sort Handler use Ascending order. The sort handler is then stored in the request object

```
if ( !peopleSort.equals("none") ){
    ContainerSorterBean sorter =
        new ContainerSorterBean("directoryPeopleContainer", jParams, peopleSort);
    if ( !peopleSortOrder.equals("asc") ){
        sorter.setDescOrdering();
    }
    // Store the sort handler in the request object.
    // It will be used later by the container list loader.
    request.setAttribute("directoryPeopleContainer_sort_handler", sorter);
}
```

### LINKING A SORT HANDLER TO A CONTAINER LIST

In the listing above, we store the Sort Handler in the request object as an attribute named : `directoryPeopleContainer_sort_handler`.

This name respects the format used to link a sort handler with a container list.



### Container\_List\_Name + “\_sort\_handler”

This handler will be detected by Jahia and it will be used to apply ordering when loading the corresponding container list.

#### **SORT VALUES AS NUMBER**

By default a sort handler sort the values as `String`, but you can force it to convert the `String` value to a long representation.

For example, when sorting people on the field “People Rate”, we want to apply a number ordering. In the Sort Handler constructor, we set the fourth parameter to `true`.

```

ContainerSorterBean sorter = null;
if ( peopleSort.equals("directoryPeopleRate") ){
    // we can force the sort comparison to convert field value to number
    // representation (long)
    sorter = new ContainerSorterBean("directoryPeopleContainer",
        jParams, peopleSort, true);
} else {
    // for all other field, we use the default String sort comparison
    sorter = new ContainerSorterBean("directoryPeopleContainer", jParams, peopleSort);
}

```

Another way to force a Sort Handler to apply number ordering is to call its method:

```

sorter.setNumberOrdering(true);

```

## 2.2 CONTAINER LIST PAGINATION

The Container List Pagination is an improvement of the initial Container List Scrolling mechanism available in Jahia 2.1.

This feature can be implemented independently of Search, Filter and Sort Options.

In particular it can be implemented as Get (Link URL) or Post (Form submission) request.

The figure below illustrates the elements of the Container List Pagination you can implement in the template file.

### 2.2.1 GENERAL CONSIDERATIONS

[TODO PICTURE]

#### **ENCLOSING PAGE FORM TAG AND CONTAINER LIST TAGS**

These elements are implemented using tags that must be enclosed inside a container list tag :

```

<content:containerList name="directoryPeopleContainer">
... // Elements of Pagination go here.
</content:containerList>

```

If you want these elements to be implemented for a Post request (form submission), you need to enclose the previous code inside a Jahia Page Form (HTML form, which action is an URL requesting the current page):

```

<content:jahiaPageForm name="jahiapageform">

```

```
<content:containerList name="directoryPeopleContainer">
  ... // Elements of Pagination go here.
</content:containerList>
</content:jahiaPageForm>
```

You need to include the Jahia default Javascript file `jahia.js`, too. Simply add the `JSToolsTag` at the top of your template file:

```
<%@ taglib uri="JahiaLib" prefix="jahia" %>
<content:JSTools/>
```

## WORKING WITH PAGINABLE CONTAINER LIST

These features are only useful if they are used with paginatable container lists.

You define a default item per page value (called window size) at the same time you declare the container list.

### TAGLIB VERSION

```
<content:containerList name="directoryPeopleContainer" windowSize="5">
  ...
</content:containerList>
```

### JAVA API VERSION

```
jData.containers().declareContainer( "directoryPeopleContainer", "People
  container", directoryPeopleFields , 5 , 0); // here , the window size is set to 5.
```

## TRACKING THE CONTAINER SCROLLING VALUE OF THE CURRENTLY DISPLAYED PAGE

There is a tag you can use to populate the container scrolling value of the currently displayed page as a hidden input.

Add the tag `cListPaginationCurrentPageScrollingValue` with the attribute `valueOnly` set to `false` as below :

```
<content:jahiaPageForm name="jahiapageform">
  <content:containerList name="directoryPeopleContainer">
    ...
    <content:cListPaginationCurrentPageScrollingValue valueOnly="false" />
    ...
  </content:containerList>
</content:jahiaPageForm>
```

This tag needs to be enclosed inside a `containerList` tag and a Jahia Page Form. It will generate a hidden input used by Jahia to keep track of the current position, when scrolling through the list of containers:

```
<input type='hidden' name='ctnscroll_directoryPeopleContainer' value='5_15'>
```

## 2.2.2 NEXT, PREVIOUS BUTTONS

There are two tags that help you implement these buttons:

- `previousWindowButton` tag

```
<content:previousWindowButton title="&lt;&lt;Prev" method="post"
  formName="jahiapageform" />
```

- nextWindowButton tag

```
<content:nextWindowButton title="Next&gt;&gt;" method="post"
  formName="jahiapageform" />
```

The above tags will generate respectively these URLs:

```
<a href="javascript:changePage(document.jahiapageform,
  document.jahiapageform.ctnscroll_directoryPeopleContainer,'5_0');">
  &lt;&lt;Prev
</a>
```

and

```
<a href="javascript:changePage(document.jahiapageform,
  document.jahiapageform.ctnscroll_directoryPeopleContainer,'5_10');">
  Next&gt;&gt;
</a>
```

If you want use these tags to generate a simple URL (no need to submit any form) as below:

```
<content:previousWindowButton title="&lt;&lt;Prev" />
<content:nextWindowButton title="Next&gt;&gt;" />
```

You will have URLs that look like:

```
<a href="http://localhost:8080/jahia/Jahia/cache/offonce/pid/7/
  ctnscroll_directoryPeopleContainer/5_0">&lt;&lt;Prev</a>
```

and

```
<a href="http://localhost:8080/jahia/Jahia/cache/offonce/pid/7/
  ctnscroll_directoryPeopleContainer/5_10">Next&gt;&gt;</a>
```

The attributes for these tags are given below:

Attribute	Description	Mandatory
title	The title of the button.	No
method	If you want to implement a post (form submission) request version, you need to set this attribute to <code>post</code> . By default the method is <code>get</code> .	No
formName	The form name needed to generate the form submit Javascript code. The value must refer to the current enclosing Jahia Page Form name. It is mandatory when the method attribute is set to <code>post</code> .	No

Table 10: previousWindowButton and nextWindowButton tag attributes

### 2.2.3 QUICK PAGE ACCESS BUTTONS

The figure above shows some interesting elements :

[TODO PICTURE]

- Current page: The current page can be highlighted (in example draw in bold).
- Next range of page step: You can limit the number of quick page access buttons to be displayed in the navigation bar. Here, this value is 3, that is why a special “next range of Page Steps” button allows you to display the three next Page Steps (4, 5 and 6 if present).

There are 5 Container List Pagination tags.

As with the Next and Previous tags, they must be enclosed inside a `containerList` tag.

**cLISTPAGINATION TAG :**

It's the main enclosing tag that iterates through all the available Quick page Access Buttons to draw.

Attribute	Description	Mandatory
<code>nbStepPerPage</code>	The number max of Quick Page Access buttons to display at a time in the navigation bar.	No
<code>skipOnePageOnly</code>	Set to “false” if you want to force displaying the Quick Page Access Buttons even though there is only one page available (and it is the currently displayed pages). The default value is “true”.	No

Table 11: cListPagination tag attributes

**cLISTPAGINATIONPREVIOUSRANGEOfPAGES,cLISTPAGINATIONNEXTRangeOfPAGES TAGS :**

These tags are used to generate the buttons allowing to jump to the previous and next range of Quick Access Buttons.

In the figure above, the maximum of Quick Page Access Buttons to display per page has been set to 3 in the enclosing `cListPagination` tag.

[TODO PICTURE]

```
<content:cListPagination nbStepPerPage="3">
  <content:cListPaginationPreviousRangeOfPages method="post"
    formName="jahiapageform" title="&#160;..&#160;"/>
  ...
  <content:cListPaginationNextRangeOfPages method="post"
    formName="jahiapageform" title="&#160;..&#160;"/>
</content:cListPagination>
```

Attribute	Description	Mandatory
<code>title</code>	The title of the button.	No
<code>method</code>	If you want to implement a post ( form submission) request version, you need to set this attribute to <code>post</code> . By default the method is <code>get</code> .	No
<code>formName</code>	The form name needed to generate the form submit Javascript code. The value must refers to the current enclosing Jahia Page Form Name. It is mandatory when the method attribute is set to <code>post</code> .	No

Table 12: cListPaginationPreviousRangeOfPages and cListPaginationNextRangeOfPages tag attributes

**cLISTPAGINATIONPAGEURL TAG :**

This tag must be enclosed inside the `cListPagination` tag. It is used to generate the Quick Page Access Buttons of the current Range of Page Steps :

The current range of Page Steps contains the page 4, 5 and 6.

[TODO PICTURE]

This tag should be preceded by the `cListPaginationPreviousRangeOfPages` tag and followed by the `cListPaginationNextRangeOfPages` tag as shown in the listing below :

```
<content:cListPagination nbStepPerPage="3">
  <content:cListPaginationPreviousRangeOfPages method="post"
    formName="jahiapageform" title="#160;..#160;"/>
  <content:cListPaginationPageUrl method="post"
    formName="jahiapageform" />#160;
  <content:cListPaginationNextRangeOfPages method="post"
    formName="jahiapageform" title="#160;..#160;"/>
</content:cListPagination>
```

Attribute	Description	Mandatory
title	The title of the button.	No
method	If you want to implement a post ( form submission) request version, you need to set this attribute to <code>post</code> . By default the method is <code>get</code> .	No
formName	The form name needed to generate the form submit Javascript code. The value must refers to the current enclosing Jahia Page Form Name. It is mandatory when the method attribute is set to <code>post</code> .	No

Table 13: `cListPaginationPageUrl` tag attributes

**IFCLISTPAGINATIONCURRENTPAGE TAG:**

When iterating through the Quick Page Access Buttons list, this tag can be used to check if the current Quick Page Access Button to display refers to the currently displayed page. If so, we can highlight this button using bold characters, etc...

The code below show where this tag take place:

```
<content:cListPagination nbStepPerPage="3">
  <content:cListPaginationPreviousRangeOfPages method="post"
    formName="jahiapageform" title="#160;..#160;"/>
  <content:ifCListPaginationCurrentPage><b>
</content:ifCListPaginationCurrentPage>
  <content:cListPaginationPageUrl method="post"
    formName="jahiapageform" />#160;
  <content:ifCListPaginationCurrentPage></b>
</content:ifCListPaginationCurrentPage>
  <content:cListPaginationNextRangeOfPages method="post"
    formName="jahiapageform" title="#160;..#160;"/>
</content:cListPagination>
```

**2.2.4 CONTAINER LIST PAGINATION INFORMATION TAGS**

There are 3 tags used to give some information about the navigation through the paginated container list. Their usages are quite simple. They only need to be enclosed inside the `containerList` tag.

[TODO PICTURE]

```
<content:containerList name="directoryPeopleContainer">
  ...
  [<content:cListPaginationFirstItemIndex />-<content:cListPaginationLastItemIndex />]
  of <content:cListPaginationTotalSize />
  ...
</content:containerList>
```

#### **cLISTPAGINATIONFIRSTITEMINDEX TAG**

This tag draws the index value of the first container item currently displayed.

#### **cLISTPAGINATIONLASTITEMINDEX TAG**

This tag draws the index value of the last container item currently displayed.

#### **cLISTPAGINATIONTOTALSIZE TAG**

This tag draws the total number of containers of the enclosing container List.

## 2.2.5 CUSTOMIZABLE ITEMS PER PAGE OPTION

It is possible to allow the user to change the number of items per page used in the pagination of a given container list. The optional window size will override the one set when declaring the container list.

[TODO PICTURE]

You only need to provide a request parameter with a name that respects the following format :

#### **Container\_List\_Name + “\_windowsize”**

In example, the Select Box above is linked to a Container List named `directoryPeopleContainer`:

```
<select class="text" name="directoryPeopleContainer_windowsize"
  onChange="javascript:document.jahiapageform.submit()">
  <option value="5">5</option>
  <option value="10">10</option>
  <option value="20">20</option>
</select>
```

## 2.3 DEALING WITH MULTIPLE LANGUAGES

[TODO DESCRIPTION How to search just in the current language or all languages

```
ContainerSearcher searcher =
  ContainerSearcher(listId, containerSearchQuery, loadRequest);
searcher.setLanguageCodes(langCodes);]
```

## 2.4 EXAMPLES

The template file used to illustrate this documentation has been packaged as a demo template set file (`filter_demo.jar`). This template set contains only one template file named `Filters`. You can use to try the features described in this documentation.

There are two ways to deploy this template set:

## DEPLOYING AS SHARED TEMPLATE SET

1. Copy the file `filter_demo.jar` in the following directory :  
<jahia-home>/WEB-INF/var/shared\_templates
2. Restart Jahia
3. Create a new Virtual Site with the template set named `Filter_Demo`.

Note : Process this way only if you are able to create a new Virtual Site (Check your license limitation).

## DEPLOYING AS NEW TEMPLATE SET

Suppose your site is given a site key named `myjahiasite` and you want to deploy these new template files in it :

1. Copy the file `filter_demo.jar` in the following directory :  
<jahia-home>/WEB-INF/var/new\_templates/myjahiasite
2. Jahia will detect the new template set and automatically deploy it for the given site.
3. Now a new template named `Filters` should appear in the available templates list. Create a new page using this template.

## 2.5 FAQ

*Question:* To sort a `JahiaContainerList`, I do the following in a template:

```
csb = new ContainerSorterBean("MyList",jData.params(),"theField");
request.setAttribute("MyList_sort_handler",csb);
```

but when I goto the page I always see the sort-order I defined in the properties-popup of that list. Do I have to disable the sorting imposed from properties dialog, and if so, how?

*Answer:* The container list ranking ( the one set in container list properties engine) will be ignored when sort options are correctly set at template level.

Be sure that :

1. Filter and Sorter options must be declared before any container declaration ( not only before the container list you want to sort, but before any container list declaration ).
2. With absolute container, use Filter or Sort's constructor that requires the container list ID instead of the container list name as parameter.

```
ContainerSorterBean(int ctnListID, String fieldName) OR
ContainerSorterBean(int ctnListID, String fieldName, boolean numberSort)
```

To retrieve the absolute container list ID of a container list you've declared at the site's home page, you can achieve it this way:

```
int ctnID = ServicesRegistry.getInstance().getJahiaContainersService()
    .getContainerListID( "MyList", jData.params().getSite().getHomePageID() );
```

## 3 NAVIGATION

As we have seen in the technical overview, Jahia treats pages as any other elements of content, so creating navigation within a Jahia site is mostly a matter of manipulating content objects so that we can build page menus easily.

Navigation can range from the very simple to the quite complex, so we will introduce the different types of menus progressively.

### 3.1 PAGE PATH

First of all we will present a “location tool”, that we call the page path, which displays the position in the hierarchy of pages of the currently displayed page, as illustrated below:

Home page / Customers Extranet / Projects / Customer 1 / Collaboration Tools

**Picture 4.5.: Example for page path**

This type of navigation is used for both:

- view your current position in the tree
- navigate quickly back up to parent pages.

We will present taglib and scriptlet versions of this navigation tool.

#### TAGLIB VERSION

```
<content:currentPagePath separator=' &gt; ' maxchar='40' />
```

The taglib version is straightforward, since the `currentPagePath` tag is quite powerful and handles all the rendering for you. It takes two parameters: one to specify the separator used between the page names, and the other to specify the maximum displayed length for a page title. If the title is longer than the specified value, the title is truncated and “...” is appended to the output.

#### SCRIPTLET VERSION

The disadvantage of the taglib version is that customization is limited to what the tag will allow, and this might not always be sufficient for template developers. Using the scriptlet version to generate a page path is more flexible, and we present the code below :

```
<%
Enumeration thePath = jData.page().getPagePath(jParams.getOperationMode(), jahiaUser);
while (thePath.hasMoreElements()) {
    JahiaPage thePage = (JahiaPage) thePath.nextElement();
    if (thePage != null) {
        <a href="<%=thePage.getUrl(jParams) %>"><%=thePage.getTitle() %></a>
        <%if (thePath.hasMoreElements()) {
            &gt;
        }
    }
}
%>
```

The above scriptlet retrieves the page path through a Jahia API call, and then iterates through the returned pages to display their title, outputting a separator character between each page title. Note that the title is not truncated if longer than a certain length, so this example is not 100% equivalent to the taglib version. The list of pages returned by the API call is ordered from the root page to the current page, so we don't have to inverse it to display it in the *natural* order.

### 3.2 BASIC CHILDREN NAVIGATION

In section Site Structure we have explained how to declare a container list, that will contain page fields that represent the children pages of the current page. In this section we will show you how to access these



objects to display the navigation tree (the declaration is not shown in the code snippets, but has to be done before).

The `Page` field type retrieves a `PageBean` (or a `JahiaPage` object through the `getFieldObject()` method call, when using the Java API) and contains the `title`, the URL as well as other page attributes used for display in templates.

#### TAGLIB VERSION

```
<!-- start left menu display -->
<content:containerList name='leftMenuContainerList' id='leftMenuContainerList'>
  <content:container id="leftMenuContainer">
    <content:pageField valueId="leftLink" name='leftLink' />
    <logic:notEmpty name="leftLink">
      <a href="<bean:write name='leftLink' property='url' />">
        <bean:write name='leftLink' property='title' />
      </a>
    </logic:notEmpty>
    <jahiaHtml:actionMenu name="leftMenuContainer" namePostFix=""
      resourceBundle="jahiatemplates.Corporate_portal_templates" useFieldSet="false">
    </jahiaHtml:actionMenu>
  </content:container>
  <jahiaHtml:actionMenu name="leftMenuContainerList" namePostFix=""
    resourceBundle="jahiatemplates.Corporate_portal_templates" useFieldSet="false">
  </jahiaHtml:actionMenu>
</content:containerList>
<!-- end left menu display -->
<!-- ===== End Left menu ===== -->
```

#### SCRIPTLET VERSION :

```
<!-- start left menu display -->
<%
JahiaContainerList leftMenuContainerList = jData.containers().getContainerList(
  "leftMenuContainerList" );
ContainerListBean leftMenuContainerListBean = new
  ContainerListBean(leftMenuContainerList, jParams);
Enumeration leftMenuContainerEnumeration = leftMenuContainerList.getContainers();
while (leftMenuContainerEnumeration.hasMoreElements()) {
  JahiaContainer leftMenuContainer =
    (JahiaContainer) leftMenuContainerEnumeration.nextElement();
  ContainerBean leftMenuContainerBean = new ContainerBean(leftMenuContainer, jParams);
  JahiaPage leftLink = (JahiaPage) leftMenuContainer.getFieldObject("leftLink");
  if (leftLink != null) {%>
    <a href="<%=leftLink.getUrl(jParams) %>"><%=leftLink.getTitle() %></a>
  }
  jData.gui().html().drawBeginActionMenu(leftMenuContainerBean, null, null,
    false, "", "jahiatemplates.Corporate_portal_templates", null, out);
}
jData.gui().html().drawBeginActionMenu(leftMenuContainerListBean, null, null,
  false, "", "jahiatemplates.Corporate_portal_templates", null, out);
%>
<!-- end left menu display -->
<!-- ===== End Left menu ===== -->
```

### 3.3 ADVANCED NAVIGATION (WITH RECURSIVE DISPLAY)

**jahia** >5.1 [TODO DESCRIPTION We will offer a new tag for creating the menu]

The examples we presented above only show a flat list of child pages for the current page, but don't allow us to present a sub-tree of page, starting at the current position. We will now assume that all the pages have a container list that we will call `navigationContainerList`. It will contain a field called `navigationLink` of type `Page`. Note that the templates themselves do not need to be the same on all these pages, but they must all include our navigation container list.

In such a case, all the sub-pages that we want to display will contain a `navigation` container list with the same name and with the same structure, including a field of type `Page`.

[TODO PICTURE]

In the above illustration, we give an example of pages that all include a `navigation` container list. They have been populated here, but some of the child reference arrows have not been displayed, mostly for space reasons, but they should be there.

One thing that is specific to fields of type `Page` is that they can actually contain three types of values :

- either they are actually a *real sub page*, meaning that the page ID stored in this field will actually be a child page of the current page. This means for example that if we delete the parent page, the child page will also be deleted, since it is *really* positioned underneath it in the content tree.
- It may also be a *link* to another Jahia page, anywhere in the tree, which means that if the current page is deleted, the linked page will remain, since it still has another parent.
- It could be an *external URL*, such as <http://www.jahia.org>, which should be pretty straight-forward.

So when building navigation container lists with fields of type `Page`, we can actually build menus that either point to sub-pages, links to Jahia pages, or external URLs.

The process to declare and display a sub-tree navigation menu will therefore look something like this :

1. declare the navigation container list
2. retrieve navigation container list for the current page
3. iterate through the containers, for each container, test the page field value type, if it's *real* sub-page, recursively call step 2

Because of the need for a recursive call, we will present this navigation menu as a scriptlet, since there will probably be a need for design customization. It might also be possible to create a taglib from this, but at the expense of limiting customization, and obfuscating a bit the process.

So first let's show the declaration part, which must be in all the templates of the pages that we want to display in our sub-tree:

```
<%  
// declarations  
%>  
<content:declareContainerList name='navigationContainerList' title='Left Menu'>  
  <content:declareContainer>  
    <content:declareField name='navigationLink' title='Link' type='Page'  
      titleKey='link' bundleKey='<%=resBundleID%>' />  
  </content:declareContainer>  
</content:declareContainerList>
```

As we can see this declaration is pretty standard.

The code used to display will be presented as a recursive method, that will contain, what we presented in steps 2 and 3 in our above process. The method takes a few parameters:

- `jData`: the current `JahiaData` context object, which offers accessors to the content objects stored in Jahia.
- `rootPageId`: the original starting `pageID`, this will be the page ID integer that is the page identifier for the page that made the original method call.
- `currentPageId`: the page ID we are currently processing. This is used to know on which page we must access the navigation container list for the current recursive call
- `currentLevel`: indicates the current recursion level, used to indent the display visually so that we see the current depth in the sub-tree
- `pageContext`: the current JSP page context object, in which we can store and retrieve contextual `JavaBean` objects. Here it is only used to retrieve the output object to write out HTML code.

Here is the full recursive method, which we will review in detail:

```
<%
// display
%>
<%!
public void getPageSubTree( JahiaData jData, int rootPageId, int currentPageId,
    int currentLevel, PageContext pageContext) throws JahiaException {
    JspWriter out = pageContext.getOut();
    JahiaContainerList navigationContainerList = jData.containers().
        getAbsoluteContainerList("navigationContainerList", currentPageId);

    try {
        if (navigationContainerList != null) {
            Enumeration navigationContainerEnum = navigationContainerList.getContainers();
            int c = 0;
            String levelIndent = "";
            for (int i = 0; i < currentLevel; i++){
                levelIndent += " ";
            }
            while (navigationContainerEnum.hasMoreElements()) {
                c++;
                if (c == 1 ){
                    out.println(levelIndent + "<div id=\"level\" + currentLevel + \">");
                }
                JahiaContainer navigationContainer = (JahiaContainer)
                    navigationContainerEnum.nextElement();
                ContainerBean navigationContainerBean = new
                    ContainerBean(navigationContainer, jData.params());
                JahiaPage navigationLink = (JahiaPage) navigationContainer.getFieldObject(
                    "navigationLink" );
                if (navigationLink != null) {
                    out.print(levelIndent + " <span><a ");
                    if (navigationLink.getID() == jData.page().getID()){
                        out.print("id=\"current\" ");
                    }
                    out.print("href=\"\" + navigationLink.getUrl(jData.params()) +
                        \"\">"+navigationLink.getTitle()+"</a>");
                    jData.gui().html().drawBeginActionMenu(navigationContainerBean, null, null,
                        false, "Link", "jahiatemplates.Basic_templates", null, out);
                    out.println("</span>");
                    // displays sub links
                    if (jData.gui().isPageInPath(navigationLink.getID())) {
                        currentLevel++;
                        getPageSubTree(jData , rootPageId, navigationLink.getID(), currentLevel,
                            pageContext);
                    }
                }
            }
        }
    } catch (JahiaException e) {
        // ...
    }
}
```

```

    }
  }
}
if (c > 0 ){
  out.println(levelIndent + "</div>");
}
ContainerListBean navigationContainerListBean = new
  ContainerListBean(navigationContainerList, jData.params());
if (! navigationContainerListBean.isActionURIsEmpty() ) {
  out.println("<span>");
  jData.gui().html().drawBeginActionMenu(navigationContainerListBean , null,
    null, false, "Link", "jahiatemplates.Basic_templates", "links", out);
  out.println("</span>");
}
}
} catch (IOException ioe) {}
}
%>

```

An example of using this method in a template would be:

```

<%
getPageSubTree( jData , homePageId, homePageId, 1, pageContext);
%>

```

where `homePageId` would be the integer containing the site's home page, or the current page ID, depending on where you want the sub-tree display to start. Now let's cut this method up to see what the tricky parts are, and how they work together to perform the full display of the sub-tree:

```

JspWriter out = pageContext.getOut();
JahiaContainerList navigationContainerList = jData.containers().
getAbsoluteContainerList("navigationContainerList", currentPageId);

```

The above code retrieves the output object, and then retrieves the container list on the page. Here instead of a regular `getContainerList()`, we use a `getAbsoluteContainerList`. With the `getContainerList()` method call, only the current page container lists may be retrieved, but the `getAbsoluteContainerList()` method call allows us to retrieve a container list on another page, using absolute addressing of a container list. So we pass the `currentPageId` parameter that contains the page ID for which we want to display the current children in the current recursive pass.

Now that we have retrieved the correct container list, we can start processing the containers, iteratively, and output indented page titles :

```

try {
  if (navigationContainerList != null) {
    Enumeration navigationContainerEnum = navigationContainerList.getContainers();
    int c = 0;
    String levelIndent = "";
    for (int i = 0; i < currentLevel; i++){
      levelIndent += " ";
    }
    while (navigationContainerEnum.hasMoreElements() ) {
      c++;
      if (c == 1 ){
        out.println(levelIndent + "<div id=\"level\" + currentLevel + \">");
      }
      JahiaContainer navigationContainer = (JahiaContainer)
        navigationContainerEnum.nextElement();
    }
  }
}

```

```
ContainerBean navigationContainerBean = new
ContainerBean(navigationContainer,jData.params());
```

The next step is to actually display the contents of the current page field in the current container. We also output the action menu, which allows us to edit the field value directly from the navigation menu. The code below also includes a check if the page currently being displayed is the current page, in which case we will want to probably modify the output to highlight it, which is done here by using a CSS style called `current`.

```
JahiaPage navigationLink = (JahiaPage) navigationContainer.getFieldObject(
"navigationLink" );
if (navigationLink != null) {
out.print(levelIndent + " <span><a ");
if (navigationLink.getID() == jData.page().getID()){
out.print("id=\"current\" ");
}
out.print("href=\"" + navigationLink.getUrl(jData.params()) +
"\>" + navigationLink.getTitle() + "</a>");
jData.gui().html().drawBeginActionMenu(navigationContainerBean, null, null,
false, "Link", "jahiatemplates.Basic_templates", null, out);
out.println("</span>");
```

We now come to the real “meat” of this method, which is the recursive call. Here we want to only make a recursive call if the children to be displayed are actually *real* pages, which we test by testing if the children are in the page path (see the page path navigation tool section above for details). This way we insure that only pages in the current tree (and therefore excluding links and external URLs) will be displayed in the navigation menu. If this test is successful, we increment the current recursive level and perform the recursive call:

```
// displays sub links
if (jData.gui().isPageInPath(navigationLink.getID())) {
currentLevel++;
getPageSubTree(jData , rootPageId, navigationLink.getID(),currentLevel,
pageContext);
}
```

The rest of the method finishes the display of the current container, displaying its action menu, and ends:

```
if (c > 0 ){
out.println(levelIndent + "</div>");
}
ContainerListBean navigationContainerListBean = new
ContainerListBean(navigationContainerList,jData.params());
if (! navigationContainerListBean.isActionURIsEmpty() ) {
out.println("<span>");
jData.gui().html().drawBeginActionMenu(navigationContainerListBean , null,
null, false, "Link", "jahiatemplates.Basic_templates", "links", out);
out.println("</span>");
}
}
} catch (IOException ioe) {}
}
%>
```

So there we have it: we can now recurse through an entire Jahia site if we set the initial page ID to the home page ID, making it possible to actually display a full site map. In practice, you might want to customize this method to limit the number of displayed levels, or do other fancy type of displays, but for the sake of simplicity only the basic principle is presented here. The templates packaged with Jahia are a good place to look for custom examples of the principles presented here.

## 4 FILTERING

[TODO DESCRIPTION not covered in 3.1 ???]

## 5 OTHER TEMPLATE ISSUES

### 5.1 REFERENCES TO STATIC RESOURCES

[TODO DESCRIPTION]

Unfortunately, many times it happens that the URLs for static resources (images, CSS etc.) are not correctly built.

e.g.:

```

```

or

```
<link href="../../../global.css" rel="stylesheet" type="text/css" />
```

The problem is not only that those resources are not found and the layout gets corrupted, but also that they result in URLs going to Jahia servlet that, if it does not find any page ID or key in the URL, simply renders the home page.

Resolution

You should review all templates used in the site and either use the `<content:contextURL/>` tag, like it is already done in many of the templates:

```

```

You can detect the wrong URLs using e.g. some development browser plugins (e.g. "View Dependencies" for Firefox) or just by looking at the log file, where you can see Jahia INFO log for those wrong URLs:

---

# CHAPTER 5: SPECIAL ENGINES

*Here a template developer can learn how to make use of existing Jahia engines, e.g. for searching, displaying the site map and how to use the API to perform the publishing of Jahia content.*

## 1 SEARCH AND ADVANCED SEARCH

This chapter first gives an overview of Jahia's search architecture. It then explains how to insert the search facility inside your templates. This is then extended to provide advanced search capabilities.

### 1.1 SEARCH ARCHITECTURE OVERVIEW

Jahia uses the Jakarta Lucene technology from the Apache Foundation (<http://jakarta.apache.org/lucene>) to index content. Lucene is a high-performance, full featured text search engine, which relies on an index file and maximizes search performance whilst minimizing RAM usage. Lucene automatically creates and updates in background an index from the Jahia database. PDF file searching is supported via the PDFBox library and while Microsoft Word, Excel and PowerPoint file searching uses the Jakarta POI library.

It is important to note the conceptual differences between the search functionality detailed in section Container List Sort, Search and Pagination and a Lucene search. In that chapter, we discussed how to search and sort specific Jahia container lists using filters and how to display them. This can also be done with Lucene, which supports searching for any content including containers, external files and page content. The difference is that Lucene doesn't search directly on the Jahia database. Instead it needs to update its index using a background thread. Therefore, there are no guarantees that the returned search results carried out on a container list will reflect its latest state in the Jahia database. For searches where high integrity is required, it is preferable to use Jahia's database search mechanism described in the following chapter.

The diagram below depicts the flow of events when a user requests a search:

[TODO PICTURE]

1. Jahia receives the search request from the user.
2. It then carries out the search on Lucene given the submitted user query. A list of search objects is returned and stored in the session.
3. Jahia then forwards the response to the `searchresult.jsp` template file, which uses the `results` attribute to render the view.
4. `searchresult.jsp`'s output is routed to the user.

The `searchresult.jsp` file can be modified leaving every aspect of the search results design to the JSP coder, in a similar manner to that used for the sitemap described in section Sitemap.

In step 2, the search results returned by Lucene are filtered to prune all illegal results. First, it checks the ACL (Access control list) on the page and then on each field. If the current user does not have the necessary rights, the result is pruned from the final result set. Secondly, it checks for results that are in the current staging mode, so that searches made in Live mode will not give results from uncommitted pages i.e. not active.

### 1.2 USING THE SEARCH FUNCTIONALITY

#### 1.2.1 INSERTING A SEARCH LINK

In its simplest form, providing search facilities within a template is just a matter of adding the following tag:



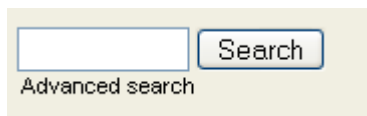
```
<a href="<%=jData.gui().html().drawSearchLauncher()%">"><content:resourceBundle
resourceBundle="jahiatemplates.Basic_templates" resourceName="search"/></a>
```

This in turn calls the `org.jahia.engines.search.Search_Engine` class which doesn't implement any search algorithm, it calls the `org.jahia.services.search.JahiaSearchService` for that. Instead, `Search_Engine` is here as an intermediary between the user and the service. Once `JahiaSearchService` has carried out the search, it stores the results in the `searchResult` session attribute (and the search query in `searchString`), which is subsequently used by a JSP file to display sorted results to the user.

`Search_Engine` forwards the request to `searchresult.jsp` file located in the current template package, which deals with the actual rendering of the search view. If it cannot find it, it will use the default `/jsp/jahia/engines/search/searchresult.jsp` file instead.


## Inserting a Search Form

A search form which includes access to advanced search options can be inserted directly instead of a simple link as above. For this example, we take as a basis the `corporate_templates` template set. This simple search form is included in the `header.inc` file and renders to this:



Picture 5.1.: Search Form

When the user clicks on advanced search, the following page appear:



Search results for ""

Search powered by   
Picture 5.2.: Advanced search

The advanced search page allows the user to search on a variety of fields such as file formats, document dates or search domains. Currently, the advanced search option page is also generated by `searchresult.jsp`.

**jahia >4.1** Advanced searching facility was introduced and is only supported from Jahia version 4.1 onwards.

Before listing the necessary code to render the search form, a few extra html `<input>` parameters are introduced:

- `maxPageItems`: the maximum number of items to display on each search result per page. This is settable from the advanced search page and the result page. It is not settable from the simple search form so it uses the value used in previous searches or the default value of 10.
- `freeSearch`: this is a copy of the query string present in the simple search form before the user clicks on the Advanced Search link. It is then used by `searchresult.jsp` to populate the value of the `freeText` input in the advanced search options view.
- `searchView`: this selects the type of search view to render. In the example, the default value is `simple` which supports only a search string. Advanced search functionality is introduced by using the `advSearch` value.
- `searchlang`: If no value is provided, Lucene will only include in the search results in the user's language. If the value is set to `all_lang` then content in all languages is included. (refer to following section for more details).
- `JahiaSearchResultHandlerImpl.oneHitByPage`: The name of this input is actually generated by the following Java code: `PageSearchResultBuilderImpl.ONE_HIT_BY_PAGE_PARAMETER_NAME`. It takes a Boolean value. If it is set to `true`, it will display one search result entry per page even if more than one file or text field matches the search query. If the value is `false`, it displays all matching file fields as individual entries in search result page (but only once for multiple matching text fields on the same page).

Below is the code necessary to display the search form. For conciseness, some portions of the code have been removed and replaced by "...” label.

```
<head>
...
<script language="javascript" type="text/javascript">
  <!--
  //This code is included by default in all pages through the header.inc
  //It updates the maxPageItems value with the currently user selected one and
  //includes it as a hidden input in the form
  //It defines the maximum number of items per page to display in the search results.
  function checkMaxPageItems(theForm) {
    if ( document.searchpager && document.searchpager.maxPageItems
        && document.searchpager.maxPageItems.selectedIndex != -1 ){
      theForm.maxPageItems.value = document.searchpager.maxPageItems.
        options[document.searchpager.maxPageItems.selectedIndex].value;
    }
  }
  //Stores the current query string as a hidden form input so that when user clicks
  //on advanced search, the query is pasted into the free search input field by
  // default
  function setFreeSearchInput(theForm) {
    theForm.freeSearch.value=theForm.search.value;
  }
  -->
</script>
<%@ include file="splash.inc"%>
...
</head>
```

```

...
<body>
...
<form name="searchForm" method="post"
    action="<%=jData.gui().html().drawSearchLauncher()%>"
    onSubmit="checkMaxPageItems(document.searchForm)" >
//find the previous query string in the session and put as default value,
//otherwise leave it empty
<input type="text" name="search"
    value='<%=JahiaTools.getStrParameter(request,"search","")%>' />
//Updates all necessary input fields when the user selects simple search
<a href="javascript:checkMaxPageItems(document.searchForm);setFreeSearchInput(
    document.searchForm);document.searchForm.submit()" >
    " width="15" height="15" border="0" align="middle"/>
</a>
//used by searchresult.jsp to know how many items to display by page
<input type="hidden" name="maxPageItems" value="10" /><br>
//Updates all necessary input fields when the user selects advanced search
<a href="javascript:checkMaxPageItems(document.searchForm);setFreeSearchInput(
    document.searchForm);document.searchForm.searchView.value='advSearch';document.
    searchForm.submit()">Advanced search
</a>
//display all matching fileFields (but only once for matching textfield) as
//individual entries in search results page
<input type="hidden"
    name="<%=PageSearchResultBuilderImpl.ONE_HIT_BY_PAGE_PARAMETER_NAME%>"
    value="false" />
//default view is simple search. Value is changed by setFreeSearchInput () to
//advSearch if the user selects advanced search
<input type="hidden" name="searchView" value="simple" />
<input type="hidden" name="freeSearch" value='' />
//searches for query on pages in any language, by default it only searches in pages
//in the the current language (so just get rid of that hidden input to do so)
<input type="hidden" name="searchlang" value='all_lang' />
</form>
...
</body>

```

## 1.3 FURTHER ADVANCED SEARCH CUSTOMIZATION

### 1.3.1 ADVANCED SEARCH ARCHITECTURE

In advanced search mode, the `org.jahia.engines.search.Search_Engine` class is also called to prepare the search environment. It stores all the search parameters in the session `HashMap engineMap` and lets the `org.jahia.engines.search.SearchViewHandler` object do all the actual searching by calling the `org.jahia.services.search.JahiaSearchService` class. Implement your own `org.jahia.engines.search.SearchViewHandlerImpl` to create you custom search mechanism.

Note that the actual search method, i.e. `searchViewHandler.search()`, is called from the `searchresult.jsp` file in the advanced search mode and stored in the `engineMap` under the `searchResult` attribute. Please examine the source code of `searchresult.jsp` of the `corporate_template` for further details.

### 1.3.2 GLOBAL VERSUS SITE SEARCHING

Each virtual site has its own Lucene index, so by default searches won't return matches from other virtual sites running on the same Jahia server. To carry out a global search (i.e. across all virtual sites), you'll

need to set the `searchDomain` attribute to the `anywhere` value. Additionally, you can limit the search space to a subset of sites by setting the `searchDomain` attribute to a list of site-keys in the following manner:

```
<SELECT name="searchDomain" multiple>
  <OPTION value="anywhere" <%if (searchViewHandler.getDomains().size()==0 ||
    searchViewHandler.getDomains().contains("anywhere")){%>selected<%}>>Anywhere
<%
  Map searchHandlers = ServicesRegistry.getInstance().getJahiaSearchService()
    .getSearchManager().getSearchHandlers();
  Iterator it = searchHandlers.values().iterator();
  SearchHandler searchHandler = null;
  while ( it.hasNext() ){
    searchHandler = (SearchHandler)it.next();
    if ( !"default".equals(searchHandler.getName()) ){
      %>
      <OPTION value="<%=searchHandler.getName()%" <% if (searchViewHandler.
        getDomains().contains(searchHandler.getName())){%>selected<%}>>site :
      <%=searchHandler.getTitle()%"
    <%}
  }%>
</SELECT>
```

Note the usage of the `multiple` attribute in the `<SELECT>` tag. It allows one or more languages to be included in the search domain.

**jahia**  
**>5.0** Global versus local search facility, along with range queries, were introduced and are only supported from Jahia version 5.0 onwards.

### 1.3.3 PAGE SUBSET SEARCHING

It is also possible to search all child pages of a given page. This is done by using the `FlatSiteMapViewHelper` class to get the subtree of pages from a given root (typically the current page). The page ID for each subtree page is extracted and then added to the Lucene search string as a (possibly large) bracketed OR condition.

Example code to generate the search query suffix is given below:

```
<%
//get the current pageID
ContentPage startPage = ContentPage.getPage(jData.page().getID());

//get the list of child pages to construct the query
//retain active and staged pages
int pageInfosFlag = ContentPage.ACTIVE_PAGE_INFOS | ContentPage.STAGING_PAGE_INFOS;

// SiteMapViewHelper.DISPLAY_ALL_LEVEL means that all subsequent children under the
// current pageID node in the tree will be returned.
// SiteMapViewHelper.DEFAULT_LEVEL means that only the immediate children under the
// currentpageID node in the tree will be returned.
FlatSiteMapViewHelper flatSiteMap = new FlatSiteMapViewHelper(jParams.getUser(),
  startPage, pageInfosFlag, jParams.getLocale().toString(),
  FlatSiteMapViewHelper.DISPLAY_ALL_LEVEL);

//Append the pageID of selected children to the current query string
searchQuery = searchQuery.append("(");
for (int j = 0; j < flatSiteMap.size(); j++) {
  int pageId = flatSiteMap.getPageID(j);
  if (j > 0){
```

```

    searchQuery.append(" OR ");
  }
  searchQuery.append("pageid:" + pageId );
}
searchQuery.append("");
%>

```

### 1.3.4 SEARCHING SPECIFIC LANGUAGES

If you want to search for content not just in all languages or the current user's locale, you can set the form attribute to a list of all the languages you wish to search over. This list is simply a series of ISO639 language codes followed by optional ISO3166 country codes. To add this to your search form, simply add a HTML select tag with the list of available search languages. For example:

```

<SELECT name=searchlang multiple>
  <OPTION value="" selected>current language</OPTION>
  <OPTION value=ar>Arabic</OPTION>
  <OPTION value=bg>Bulgarian</OPTION>
  <OPTION value=ca>Catalan</OPTION>
  <OPTION value=zh-CN>Chinese (Simplified)</OPTION>
  <OPTION value=zh-TW>Chinese (Traditional)</OPTION>
  <OPTION value=hr>Croatian</OPTION>
  <OPTION value=cs>Czech</OPTION>
  <OPTION value=da>Danish</OPTION>
  <OPTION value=nl>Dutch</OPTION>
  <OPTION value=en>English</OPTION>
  <OPTION value=et>Estonian</OPTION>
  <OPTION value=fi>Finnish</OPTION>
  <OPTION value=fr>French</OPTION>
  <OPTION value=de>German</OPTION>
  <OPTION value=el>Greek</OPTION>
  <OPTION value=iw>Hebrew</OPTION>
  <OPTION value=tr>Turkish</OPTION>
</SELECT>

```

### 1.3.5 CONTAINER AND FIELD SEARCHING

The search can be limited to a certain type of content object using a the following syntax in the search query string:

- `query_string = James AND container_definition_name:directoryPeopleContainer` will look for "James" in any field contained in containers of type `directoryPeopleContainer`.
- `query_string = James AND fieldname:directoryPeopleFirstName` will look for "James" only in fields of type `directoryPeopleFirstName` but in any containers.

`container_definition_name` and `fieldname` are special attributes stored in the search indexes for each indexed fields. So for example:

```

//Append a search string over a specific container type to the current query string
searchQuery.append("( James AND container_definition_name:directoryPeopleContainer )");

```

Other interesting attributes supported by Jahia are:

```

FIELD_FIELDID = "fieldid"; //limit search to a specific field ID
FIELD_JAHIAID = "jahiaid"; //limit search to a specific site ID
FIELD_PAGEID = "pageid"; //limit search to a pageID
FIELD_FIELDNAME = "fieldname"; //limit search to a field name
FIELD_FIELDTYPE = "fieldtype"; //limit search to a field type

```

```

FIELD_CTNIID = "ctnid"; //limit search to a specific container ID
FIELD_CTNLISTID = "ctnlistid"; //limit search to a specific container list ID
FIELD_VALUE = "value"; //limit search to a specific value
FIELD_RIGHT = "right"; //limit search to content with a particular ACL rights
FIELD_VERSION = "version"; //meaning changes according to value in FIELD_WORKFLOW_STATE
FIELD_WORKFLOW_STATE = "workflow_state"; //limit search to content in a specific
                        // staging state where 0=versioned, 1=actif,
                        // >1 = staged
FIELD_LANGUAGE_CODE = "language_code"; //limit search to content in a specific language
FIELD_KEY = "field_key"; //limit search to in a specific Lucene key (this is the key
                        // stored by Lucene -not Jahia- for every index entry)
CONTAINER_DEFINITION_NAME = "container_definition_name"; //limit search to a
                        // container with the same definition name. You can also use
                        // FIELD_CTNIID if you have the container ID
CATEGORY_KEY = "category_key"; //limit search to a specific category

```

These constants are defined in the `org.jahia.services.search.JahiaSearchConstant` class.

The syntax used is the one defined by Lucene at

<http://jakarta.apache.org/lucene/docs/queryparsersyntax.html>. So for example, to search for content with a date within a certain date range, the following code is necessary:

```

String queryStr = "fieldname:newsDate AND value:[";
Calendar cal = Calendar.getInstance(TimeZone.getTimeZone("UTC"));
cal.clear();
cal.set(2004, 1, 15, 0, 0, 0); // 15 feb 2004

// Convert date to the long representation since this is the format which it
//is stored under in Lucene
queryStr += cal.getTimeInMillis() + " TO ";
cal.clear();
cal.set(2004, 1, 19, 0, 0, 0);
queryStr += cal.getTimeInMillis() + "];";

```

## 1.4 CUSTOMIZING THE RESULTS VIEW

To customize the search results page, you need to edit either Jahia's default `searchresult.jsp` page or the one included in your template set.

The paging mechanism in `searchresult.jsp` makes use of the publicly available Pager tag Library (<http://jsptags.com/tags/navigation/pager/index.jsp>).

The hit score and along with a short text snippet is returned for each result. Depending on the search settings, a hit is returned per page found containing the query string at least once in one of its fields.

## 2 SITEMAP



[TODO DESCRIPTION new AJAXed version]



[TODO DESCRIPTION new Tag]

[TODO DESCRIPTION Sitemap chapter: we should explain the different options a template developer has to make some "sitemap" page. Currently your documentation is more focused on existing engines and how to customize them. I would rather focus him on template developer needs (aka: Displaying a sitemap)]

and illustrating all the possible ways to display a sitemap with all the pros and cons. For instance a template developer can perfectly generate a sitemap template (similar to a navigation menu) with only the 3 or 4 levels. This will results to a page which could be cached by the HTML cache and which can go quite fast. Currently the sitemap engine calls Zimbra so I am not sure it scales very well. BTW. you should mention that the AJAX sitemap template page should have the "no Robots" (I do not remember the exact HTML tag) so that Google bots or son on to dont try to click on all the AJAX sub-menus icons ]

### 3 XML IMPORT/EXPORT



[TODO DESCRIPTION Import/export: Several customers wants to know the way to dynamically import some content (e.g. convert some RSS feeds into a Jahia containerlist so that the content of the RSS feeds is indexed by Lucene, could be workflow or categorized,...). There is a document from Thomas which basically explain how to generate an import file and how to deploy such a file in WebDAV (file coding convention) in order to directly import it. It would be great to integrate it (I will try to send it to you if I can retrieve it) ]

### 4 WORKFLOWS

[TODO DESCRIPTION if not covered in ch3 4.3, like Time based publishing]



Jahia 5 features several new workflow options. Of course it will be possible to continue to use the standard 2 steps workflow (notification + validation/publishing). But we also added support for:

- No workflow: starting from a point in the sitemap you may decide to directly publish all your changes in live mode
- NSteps workflows: This is an extended version of the standard Jahia workflow which is customizable and let you define N (3,4,5,etc...) different possible steps in a workflow
- Connection to a third party Business Process Management Server: Jahia 5.0 will support third party advanced business process server. A new optional server (Jahia BPM Server - previously named Sensei) could be bought separately. Such a server support all kind of possible workflows and is fully compliant with the BPML standard (more about it in a future blog entry).

Jahia 4.0 was limited on a validation on a per page basis. Sometimes it was quite difficult to deal with it especially when you wanted to validate the left column or a box without wanting to validate the right column or the rest of the page. In Jahia 5.0, even if by default and for ease of use reasons, the workflow is customized on a per page basis, any power user can decide to force a distinct specific workflow on any object on the page (e.g. a column, a box, ...). So you can now have as many workflow as you want on a per content object basis. You can even combine several of the workflows options as described above.

# CHAPTER 6: CHANGING THE STYLE

*This chapter is only about the possibilities to change the look&feel and styles of pages and fragments generated by Jahia.*

## 1 ENGINE POP-UP CUSTOMIZATION

The aim of this chapter is to describe the mechanisms used in Jahia with engine popups customization. In fact, this customization involve several features in Jahia:

- Localizing the Portal with resource bundles.
- Customizing the Portal Appearance with different look-and-feels along with the ability to change the portal's banner, background image, company logo and cascading style sheet.
- The ability to use ACL to hide fields from being editable in the Add and Update engine popups depending of groups or users.

### 1.1 LOCALIZING THE PORTAL WITH RESOURCE BUNDLES

While not all static resources can currently be translated in different language(s) to meet a particular country or region, the ability to use resource bundles to customize Jahia engine popups is available.

#### JAHIA DEFAULT ENGINE RESOURCE BUNDLE

Jahia comes with a default resource bundle file to use with engine popups. This file must be accessible in the CLASS-PATH and is actually located in the classes directory of Jahia:

```
<jahia-home>/WEB-INF/classes/JahiaEnginesResources.properties
```

If no other specific language or region resource bundles are provided, this is the one Jahia will use. Different resources are listed in this file as shown below. They can be a CSS file, images or labels.

```
stylesheet = /jsp/jahia/engines/css/jahia.css
headerLogoImg = /jsp/jahia/engines/images/header.gif
headerBgImg = /jsp/jahia/engines/images/header_bg.gif
hrImg = /jsp/jahia/engines/images/hr.gif
pixImg = /jsp/jahia/engines/images/pix.gif
// buttons
cancelOnButtonImg = /jsp/jahia/engines/images/buttons/cancel_on.gif
cancelOffButtonImg = /jsp/jahia/engines/images/buttons/cancel_off.gif
loginOnButtonImg = /jsp/jahia/engines/images/buttons/login_on.gif
loginOffButtonImg = /jsp/jahia/engines/images/buttons/login_off.gif
...
// labels
loginToJahia = Login to Jahia
loginUsername = Username
loginPassword = Password
loginStayAtCurrentPage = Stay at current page
loginJumpToHomePage = Jump to personal Home page
```

Note : As you can see, the default language translation is English and some resources refer to a physical file (CSS and images), so they must be valid relative paths.

*These paths are relative to the **Jahia Webapp Context** directory.*



**DESIGNING ENGINE GUI WITH RESOURCE BUNDLE**

Jahia provides a tag `<content:engineResourceBundle ...>` that can be used to dynamically request a given resource from the engine resource bundle. To request the value of the resource named `stylesheet`, you only need to call the tag and provide the resource name:

```
<content:engineResourceBundle resourceName="stylesheet">
```

In the same way, for a resource that is an image, we have something like that:

```
" width="126" height="63">
```

The generated HTML code is:

```

```

Note: The tag `<content:serverHttpPath />` is needed to build the full image url. This tag returns the server request URI part of the URL: `"http://127.0.0.1:8080/jahia"`.

As an example, the login popup GUI make use of some resources listed below:

[TODO SCREENSHOT & GRAPHICS]

The JSP code for this login popup looks like that:

```
...
<%@ taglib uri="JahiaLib" prefix="jahia" %>
...
<html>
  <head>
    <title>Jahia</title>
    <link rel="stylesheet" type="text/css" href="<content:serverHttpPath/>
      <content:engineResourceBundle resourceName="stylesheet"/>" />
  </head>
  <body>
...
  <p class="text"><b><content:engineResourceBundle resourceName="loginToJahia" /></b>
  </p>
  <form name="loginForm" method="POST" action="<%=engineUrl%>">
    <table border="0" width="210" align="center">
      <tr>
        <td width="50%" class="text">
          <content:engineResourceBundle resourceName="loginUsername" />
        </td>
        <td width="50%">
          <input class="input" type="text" name="username" size="20"
            value="<%=username%>" />
        </td>
      </tr>
      <tr>
        <td width="50%" class="text">
          <content:engineResourceBundle resourceName="loginPassword" />
        </td>
        <td width="50%">
          <input class="input" type="password" name="password" size="20"></td>
        </tr>
    </table>
  </form>
...

```

```
</table>
</body>
</html>
```

To be able to use tags provided by Jahia, you must import them first. Add this directive at the beginning of your JSP file:

```
<%@ taglib uri="JahiaLib" prefix="jahia" %>
```

For more information about taglibs, please refer to the Jahia Taglib Documentation.

## PROVIDING RESOURCE BUNDLE FOR THE NEEDED LANGUAGE(S)

To create a resource bundle in a different language:

1. Copy the `JahiaEnginesResources.properties` file into a new one.
2. Name the new file to designate the appropriate language.  
For example:  
A German-language resource bundle would be named `JahiaEnginesResources_de.properties`  
A French-language resource bundle for the country of Switzerland would be named `JahiaEnginesResources_fr_CH.properties`.
3. Translate the text strings in the new file into the desired language. Provide translated versions of graphical resources ( logo, buttons... ).
4. Put this new file in a place accessible in the CLASS-PATH or in the same place as the default file `JahiaEnginesResources.properties`.

When a user accesses the portal, Jahia queries the user's browser software and operating system to determine the user's preferred language. If there is a corresponding resource bundle for this language, Jahia will use it. If not this is `JahiaEnginesResources.properties`, the default one that is used.

## SETTING RESOURCE BUNDLES AT SITE LEVEL

Until now, we discussed how to provide different resource bundles for needed languages in addition to the Jahia default resource bundle. But all of them are done at *Server Level*.

At *Server Level*, when you provide a French-Language resource bundle named `JahiaEnginesResources_fr.properties`, all engines of all virtual sites hosted by the Jahia Portal Server share this resource.

What if we have two virtual sites? Furthermore, what if we want the first virtual site still using the default resource bundle ( `JahiaEnginesResources.properties` ) and the second to use a French translated resource bundle?

What we are talking about is to provide resource bundle at *Site Level*.

To do this for a given virtual site, you simply need to append the **virtual site's sitekey** to the name of resource bundles files you want to provide for this virtual site.

In example, for the given virtual site:

Site title: My virtual site.  
Site server name: www.myjahiasite.com  
Site key: myjahiasite

Create resource bundles named like that:

`JahiaEnginesResourcesMYJAHIASITE.properties` ( the default for this site )  
`JahiaEnginesResourcesMYJAHIASITE_fr_CH.properties`

...

The appended sitekey value must be in upper case.

Jahia always looks for virtual site's resource bundles first. If none of them are available, it will use the shared server resource bundles.

### SETTING RESOURCE BUNDLES AT PAGE LEVEL

You can even set resource bundle at *Page Level*. In the template JSP file, use the tag `setEngineResourceBundle` :

For example, in your template JSP file, you should have something like that :

```
...
<%@ taglib uri="JahiaLib" prefix="jahia" %>
...
<content:setEngineResourceBundle resourceBundle="SubSiteResource" />
...
```

where `SubSiteResource` is the name of your resource bundle file: `SubSiteResource.properties`.

Engine popups launched from pages using this template will automatically use this resource bundle. When resource bundle are set at *Page Level*, they have precedence over resource bundle set at *Site Level*.

### SETTING RESOURCE BUNDLES AT USERS AND GROUPS LEVEL

You can set resource bundles for users and groups. In the template JSP file, use the tags `setUsrEngineResourceBundle` and `setGrpEngineResourceBundle` :

In example, in your template JSP file, you should have something like that :

```
...
<%@ taglib uri="JahiaLib" prefix="jahia" %>
...
<content:setUsrEngineResourceBundle resourceBundle="albertResource" userName="albert"/>
...
<content:setGrpEngineResourceBundle resourceBundle="employeesResource"
  groupName="employees"/>
...
```

where `albertResource` is the name of the resource bundle file for the user `albert`:

`albertResource.properties` and `employeesResource` refers to the `employeesResource.properties` file to use for users of the group `employees`.

Engine popups launched from pages using this template will automatically use these resource bundles looking at the current identified user.

When resource bundle are set at *User or Group Level*, they have precedence over resource bundle set at *Site Level*.

User settings have precedence over Group settings.

## 1.2 CUSTOMIZE THE PORTAL'S APPEARANCE

As with portal localization, portal's look and feel can be customized using the same resource bundles but by providing different graphical and style elements. Most important here is the customization of CSS.

**ENGINE POPUPS GRAPHICAL ELEMENTS.**

Graphical elements like `headerLogoImg`, `headerBgImg`, `hrImg`, ... refer to Jahia's default image files located in the directories:

```
<jahia-home>/jsp/jahia/engines/images/ // header logo, hr, header bg...
/icons // small icon images
/buttons // button images
```

[TODO SCREENSHOT & GRAPHICS]

Resource key	Physical file	Description
<code>org.jahia.header.image</code>	<code>/jsp/jahia/engines/images/header.gif</code>	Header logo
<code>org.jahia.headerBg.image</code>	<code>/jsp/jahia/engines/images/header_bg.gif</code>	<b>Header background</b>
<code>org.jahia.hr.image</code>	<code>/jsp/jahia/engines/images/hr.gif</code>	Horizontal rule
<code>org.jahia.cancelOff.button</code>	<code>/jsp/jahia/engines/images/buttons/cancel_off.gif</code>	Cancel off button
<code>org.jahia.okOff.button</code>	<code>/jsp/jahia/engines/images/buttons/ok_off.gif</code>	Ok off button

**Table 14: Examples for resource keys in Jahia engine's resource bundle**

You can modify them to give a look-and-feel shared by engines of all virtual sites.

If you want to give a different look-and-feel only for a given site, you should provide a different set of graphical elements and keep the original files unchanged. Follow these steps:

1. Create a resource bundle file for the site as described further (or use the one you created for this site).

*Tip : We recommend that you store your new set of graphical elements in a folder structure that respects the default structure used by Jahia.*

For a site with a sitekey `mydemosite`, these customized graphical elements are stored in the folders:

```
<jahia-home>/jsp/jahia/templates/mydemosite/customization/engines/images/buttons/icons
```

2. Set resources in the resource bundle file to refer to the new files, i.e. :

```
headerLogoImg=
/jsp/jahia/templates/mydemosite/customization/engines/images/header.gif
```

**Note :** *Because* `/jsp/jahia/templates/mydemosite` *is the site's templates files root directory, you can substitute* `/jsp/jahia/templates/mydemosite` *with* `<siteTemplate>` :

```
headerLogoImg= <siteTemplate>/customization/engines/images/header.gif
```

Jahia will automatically perform the correct translation.

**JAHIA CSS FILE.**

Jahia uses a default CSS file named `jahia.css` and located in:

```
<jahia-home>/jsp/jahia/engines/css/jahia.css
```

Currently, only a few style elements are used by the engine popups.

[TODO SCREENSHOT & GRAPHICS]

CSS File	Class	Description	Default value
jahia.css	text	Primary Text style and used both in engine popups and administration JSP files for simple texts. Used by the element <code>&lt;body&gt;</code> of engines popups to give the main the background color.	<code>.text { font-family: Verdana, Arial, Helvetica, sans-serif; font-size:12px; font-style: normal; color: #19313E; background-color: #92A4AD}</code>
jahia.css	text2	Red color text style used for messages , warnings and to highlight currently displayed fields in Add and Update Content engine popups. Background-color should be the same as for <code>.text</code> .	<code>.text2 { font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 12px; font-style: normal; color: #B42C29; background-color: #92A4AD }</code>
jahia.css	input	Style for HTML form inputs	<code>.input { fontfamily: Verdana, Arial, Helvetica, sans-serif; fontsize: 10px; fontstyle: normal; color: #000000}</code>

**Table 15: Examples for styles in Jahia engine's CSS file**

If you want to give a different look-and-feel for only a given site, you should provide a separate CSS file. Proceed in the same way as with graphical elements. The steps are:

1. Create a resource bundle file for the site as described further (or use the one you created for this site).
2. Copy the default CSS file `<jahia-home>/jsp/jahia/engines/css/jahia.css` in a new file you can modify.  
Tip : *We recommend you to respect the default structure used by Jahia. For a site with a sitekey `mydemosite`, store it in a structure like that :*  
`<jahia-home>/jsp/jahia/templates/mydemosite/customization/engines/css/jahia.css`
3. Change the stylesheet resource in the site's Resource bundle file to refer to the new files:  
`stylesheet = /jsp/jahia/templates/mydemosite/customization/engines/css/jahia.css`

Note : Because `/jsp/jahia/templates/mydemosite` is the site's templates files root directory, you can substitute `/jsp/jahia/templates/mydemosite` with `<siteTemplate>` :  
`stylesheet = <siteTemplate>/customization/engines/css/jahia.css`

Jahia will automatically perform the correct translation.

### 1.3 CUSTOMIZE ADD AND UPDATE POPUPS WITH ACL

Jahia's Add and Update Container engine popups can be customized with ACL to hide some fields of being editable regarding to certain users or groups.

#### SETTING FIELDS ACL

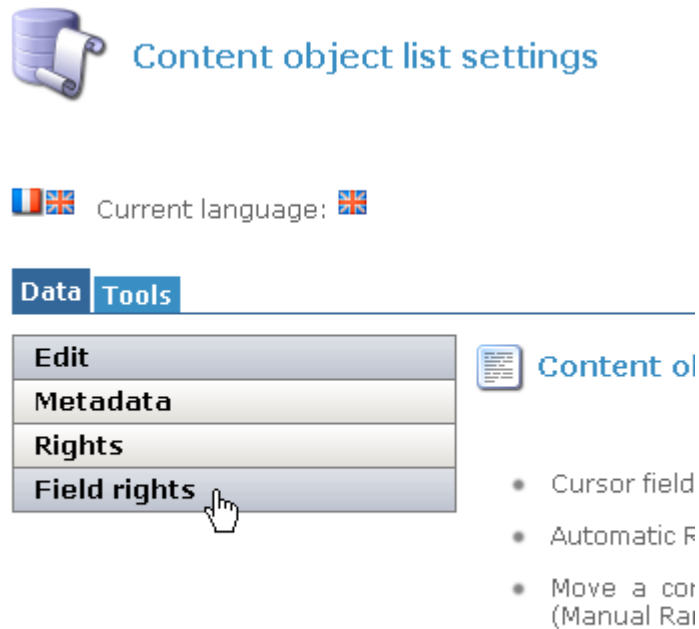
The ACL we are talking about is the one set in the container list properties popup. Taking the example of News Template taken from the `white_templates` [TODO CHECK] set, we follow these steps:

1. Open the container list properties popup by clicking the  icon :

[TODO SCREENSHOT]

Note : This icon is visible only when the container list has been fully created. A container list is created only when its first container is added to it. So if this icon is not visible, you must create a dummy container (a news item) in it. You can delete it later if needed [TODO TEST].

2. Then choose `Field Rights` in the actions select box.



Picture 6.1.: Field Rights Selection

3. Choose the group or user you want to remove the ability to edit a given field listed in the fields list:
3. Remove all permissions ( $r, w, A$ ) if you want to completely hide a field in the Add or Update Container popup for a given user or group.
4. Remove Write and Admin permissions ( $w, A$ ) if you want a field to be read-only, not editable in the Add or Update Container popup for a given user or group.

 Rights

Current Field: Title ▾ ← 1

User and group list

Click on a user/group to edit its permissions. Permissions with (\*) are inherited from the parent object and control access to this object.

UG	Perm	Src	UGId	Prop
u	RWA*	jahia	steven	
g	R--*	jahia	Employees	(john, steven)

Undo all changes

Permissions

	Allow	Deny
r Read	<input checked="" type="radio"/>	<input type="radio"/>
w Write	<input checked="" type="radio"/>	<input type="radio"/>
A Administration	<input checked="" type="radio"/>	<input type="radio"/>

Operations

Add new users / groups...

Remove the user/group from the local object

Inheritance

Copy the user/group permissions to the local object.

Prevent all inherited permissions from parent object.

Make local copy of all inherited permissions.

Don't copy inherited permissions.

Picture 6.2.:

Examples of different layout of a same Update Container popup but with different permissions settings on fields ACL:

[TODO SCREENSHOT]

This is the default layout with default field ACL settings. All the fields are editable.

[TODO SCREENSHOT]

The same Update Container popup but for the current logged user, Read, Write, Admin permissions (r, w, A) have been removed on the fields Date and Content. The user can only modify the field Title.

[TODO SCREENSHOT]

The same Update Container popup but for the current logged user, Read, Write, Admin permissions (r, w, A) have been removed on the fields Date and Content, while only the

permission Read is left on the field `Title`. The user can see the field `Title` but he cannot modify its value.

## 1.4 USER REGISTRATION, LOGIN AND SETTINGS CUSTOMIZATION



Since version 4.0.3, Jahia includes a new template that allows users to register themselves, login and edit their own settings (aka “MySettings” engine). In this chapter, we review the registration, login and profile updating processes in detail. We further show how to add new properties to a user profile.

### 1.4.1 REGISTRATION

#### OVERVIEW

Guest users can register with your site via the Jahia's *new user registration engine*. Simply include the following code:

```
<%
if (! jData.gui().isLoggedIn()) {
%>
  <td width="100%" align="right" class="topmenubuttons">
    <a href="javascript:<%=jData.gui().html().drawLoginLauncher()%>">
      <jahia:resourceBundle resourceBundle="jahiatemplates.Corporate_portal_templates"
        resourceName="login" defaultValue="login"/>
    </a>
    
    | <a href="<%=jData.gui().html().drawNewUserRegistrationLauncher()%>">
      <jahia:resourceBundle resourceBundle="jahiatemplates.Corporate_portal_templates"
        resourceName="register" defaultValue="register"/>
    </td>
%> else {%>
  <td width="100%" class="topmenubg"></td>
%>%>
```

This will present unlogged users with the opportunity to register. For example:

[TODO SCREENSHOT]

The more interesting part here is the `jData.gui().html().drawNewUserRegistrationLauncher()` code which generates the URL that calls the `org.jahia.engines.users.NewUserRegistration_Engine` engine. This engine routes the user to the following screen rendered by the `newuserregistration.jsp` file:

[TODO SCREENSHOT]

If the registration is successful, the `NewUserRegistration_Engine` routes the user to the `userregistrationok.jsp` page (below left picture) - or back to `newuserregistration.jsp` if an error occurred (below right picture):

[TODO SCREENSHOT]

A summary of the process is depicted in the following diagram:

[TODO PICTURE]



## CUSTOMIZING THE REGISTRATION

To customize the registration page, you need to edit the `newuserregistration.jsp` page which by the way should be in the root of your template package (i.e. there is no default registration page in Jahia). Please refer to the example `newuserregistration.jsp` page distributed with the `corporate_templates` set.

### CUSTOM USER PROPERTIES

To add your custom user properties to the registration process, you will need to include additional form inputs with your property name prefixed with the `newUserProp_` marker. So for example, your email property will be called `newUserProp_email` as in the example below:

```
<td>
  <jahia:resourceBundle resourceBundle="jahiatemplates.Corporate_portal_templates"
    resourceName="newUserEmail"/> :
</td>
<td>
  <input type="text" name="newUserProp_email"
    value='<%=getSingleValue(req,"newUserProp_email")%>' />
</td>
```

Note that the `getSingleValue()` function simply checks to see if the user previously submitted the form and fills in the stored values saved in the user session. All your custom properties are saved in the Jahia database with the user profile and we detail how to change them later in this chapter.

### GROUP MEMBERSHIPS

New users are given the opportunity to register to one or more groups via the “*Groups to be part of*” input field. `NewUserRegistration_Engine` returns the list of all available groups in the `groupList` engineMap session attribute to `newuserregistration.jsp` page, minus the administrator group for obvious security reasons.

```
<td>
  <jahia:resourceBundle resourceBundle="jahiatemplates.Corporate_portal_templates"
    resourceName="newUserGroupList"/> :
</td>
<td>
  <select name="newUser_groupList" multiple="yes">
    <%
      Set groupNameSet = (Set) engineMap.get("groupList");
      Iterator groupNameIter = groupNameSet.iterator();
      while (groupNameIter.hasNext()) { %>
        <option><%= (String) groupNameIter.next() %></option>
      <% } %>
    </select>
  </td>
```

The user is then automatically added as a member to any of the groups he selects.



It is therefore your responsibility to filter out any secure or non-public groups returned via the `groupList` attribute to the `newuserregistration.jsp` page. Otherwise, new users will be able to register to them without any necessary prior authorization by the administrator.

### REGISTRATION DATA VALIDATION

`NewUserRegistration_Engine` checks the submitted user data for:

- A missing or existing user name
- A missing or short password
- Non-matching or missing check password
- Invalid groups (i.e. admin group)

It currently doesn't check the absence or validity of any custom user properties you may have added. For this, you will need to append the necessary code to `NewUserRegistration_Engine`.

## 1.4.2 LOGIN

This sections looks at how users login and logout of Jahia.

### LOGIN

As detailed in the previous section, the code to display in your template a link to allow users to login is as follows:

```
<a href="javascript:<%=jData.gui().html().drawLoginLauncher() %>">
  <jahia:resourceBundle resourceBundle="jahiatemplates.Corporate_portal_templates"
    resourceName="login" defaultValue="login"/>
</a>
```

This will generate a URL containing the `/engineName/login/` string that hands over control to the `org.jahia.engines.login.Login_Engine` engine. After initializing the user's session state (such as checking for cookie support), `Login_Engine` will forward the user to the `\jahia\jsp\jahia\engines\login\login.jsp` popup. Upon successful login, the user session is updated and an event is fired.

If you plan to change style of the login popup, refer to the Engine Pop-up Customization section. However, if functional customization is necessary, we invite you to directly change the code in `login.jsp`.

The `Login_Engine` engine will also be called when a user tries to directly access a protected page while in guest mode. Once the user has logged on, the login engine will check if the user has the rights to access the originally targeted page, if he doesn't, he will be forwarded to his default home page -priority is given to the user homepage followed the user's first found group homepage.

## 1.4.3 LOGOUT

To enable the logout operation from your template, simply include the following code:

```
<a href="<%=jData.gui().html().drawLogoutLauncher() %>">
  <jahia:resourceBundle resourceBundle="jahiatemplates.Corporate_portal_templates"
    resourceName="logout" defaultValue="logout"/>
</a>
```

This hands over control to the `org.jahia.engines.logout.Logout_Engine` which clears up the user session, resets the session to guest and fires up a logout event.

Note that you can also use the `content:logButton` or `content:logRollover` tags to render buttons initiating login and logout operations.

## 1.4.4 SETTINGS

This section first details how to update user profiles. It then details how to add new, possibly read-only, properties.

## PROFILE UPDATE FOR USERS

At any time, the user can change his profile properties set during the registration process, via the `MySettings` functionality. Access to the `MySettings` page can be added to your template using the following code:

```
<a href="<%=jData.gui().html().drawMySettingsLauncher()%>">
  <jahia:resourceBundle resourceBundle="jahiatemplates.Corporate_portal_templates"
    resourceName="mySettings" defaultValue="My Settings"/>
</a>
```

If selected, this will generate the following output:

[TODO SCREENSHOT]

The engine behind the `MySettings` functionality has the following super unexpected name: `org.jahia.engines.mysettings.MySettingsEngine.MySettingsEngine` delegates the view rendering to `mysettings.jsp` (see above screenshot). If the user changes are successful, he is forwarded to the `mysettingschanged.jsp` page.

## PROFILE UPDATE FOR ADMINISTRATORS

The administrator can also change any user's profile data via the user administration panel (picture below left):

[TODO SCREENSHOT]

This popup uses the `\jahia\administration\user_management\user_edit.jsp` file.

The administrator can also create a user from scratch and explicitly specify the user's associated properties using the *Create User* functionality. This uses the `\jahia\administration\user_management\user_create.jsp` file to generate the popup depicted above (right).

## ADDING CUSTOM PROFILE PROPERTIES

We will now go through the steps necessary to add a new user profile property. This requires updating the following files:

1. MySettings Engine: `mysettings.jsp`
2. Administrator Panel: `user_edit.jsp`, `user_create.jsp`
3. JahiaAdministration.properties resource bundle

Starting with point 1, we simply add another `<input>` field to the `mysettings.jsp` `<form>`:

```
<tr>
  <td>
    <jahia:resourceBundle resourceBundle="jahiatemplates.Corporate_portal_templates"
      resourceName="mySettingsMYCUSTOM_PROPERTY"/> :
  </td>
  <td>
    <input type="text"
      <% if (isPropReadOnly(engineMap, "MYCUSTOM_PROPERTY")) {
        %>disabled="true"<% //disables editing of this property since it is read only
      }%>
      name='<%=MySettingsEngine.USER_PROPERTY_PREFIX+"MYCUSTOM_PROPERTY"%>'
      value='<%=getUserProp(engineMap, "MYCUSTOM_PROPERTY")%>' />
    </td>
```


```
</tr>
```

where `MYCUSTOM_PROPERTY` is the name of your custom property. The `getUserProp` and `isPropReadOnly` methods utilize the user properties stored in the session by `MySettingsEngine` and are defined by:

```
<%!
public String getUserProp(Map engineMap, String propName) {
    //loads the user property stored in the session by MySettingEngine engine
    UserProperty propValue = (UserProperty)
        engineMap.get(MySettingsEngine.USER_PROPERTY_PREFIX + propName);
    if (propValue == null) {
        return "";
    } else {
        return propValue.getValue();
    }
}
public boolean isPropReadOnly(Map engineMap, String propName) {
    UserProperty propValue = (UserProperty)
        engineMap.get(MySettingsEngine.USER_PROPERTY_PREFIX + propName);
    if (propValue == null) {
        return false;
    } else {
        return propValue.isReadOnly();
    }
}
}%>
```

The result is that read-only properties are supported. This is especially useful when a user's profile is retrieved from the LDAP (i.e. Lightweight Directory Access Protocol) since modifying LDAP user profiles directly from Jahia usually isn't desirable. Jahia can add additional user properties, which are stored in Jahia's local database, to a LDAP profile.

Therefore, user profiles become an assortment of both remote LDAP and local user data. The local Jahia user data is by default not in read-only mode. For more information on LDAP, please refer to Chapter 4 of the Jahia's Administration Guide.

 Read-only properties were introduced and are only supported from Jahia version 4.0.6 onwards.

The next step, i.e. Step 2, involves adding the facility to edit your new property from the Administration panel.

Let us first turn our attention to the changes necessary to the `user_edit.jsp` file. As with `mysettings.jsp`, adding a new `<input>` field to the `<form>` will do the trick:

```
<tr>
  <td align="right" nowrap>
    <jahia:adminResourceBundle resourceName="org.jahia.admin.MYCUSTOM_PROPERTY.label"/>
  </td>
  <td>
    <input class="input" type="text" size="40" maxlength="255"
      <% if (isPropReadOnly(userProps, "MYCUSTOM_PROPERTY")) { %>
        disabled="true"
      <%}%>
      name='<%=ManageUsers.USER_PROPERTY_PREFIX+"MYCUSTOM_PROPERTY"%>'
      value='<%=getUserProp(userProps, "MYCUSTOM_PROPERTY")%>' />
    </td>
</tr>
```

The `user_create.jsp` will require a similar addition to its main form:

```
<tr>
  <td align="right" nowrap>
    <jahia:adminResourceBundle resourceName="org.jahia.admin.MYCUSTOM_PROPERTY.label"/>
  </td>
  <td>
    <input class="input" type="text"
      name='<%=ManageUsers.USER_PROPERTY_PREFIX+"MYCUSTOM_PROPERTY"%>'
      size="40" maxlength="255"
      value='<%=getUserProp(userProperties,"MYCUSTOM_PROPERTY") %>'>
    </td>
</tr>
```

The final step 3 is simply adding the appropriate bundle keys and values used throughout this example in the `WEB-INF/classes/JahiaAdministration.properties` resource bundle. So for example, you'll need to add an entry describing your new user property in `org.jahia.admin.MYCUSTOM_PROPERTY.label`.

## 1.5 SUMMARY

Full customization of popups (i.e. the ability to substitute a completely different JSP files for engines) is voluntarily forbidden for the following reasons:

- Due to the inherent complexity of the engine popups back-end source code (i.e. how fields are displayed and how forms are submitted). Making substantial changes here might yield unexpected results.
- Maintaining these templates up-to-date when Jahia's API changes will be non-trivial. Using resource bundle has the advantage that Portal Localization is also integrated. Adding ACL on fields gives more control with content edition.

## 2 ACTION MENU CUSTOMIZATION

[TODO DESCRIPTION]



[TODO DESCRIPTION Ability to set the icons via CSS on a per template set basis]

## 3 GENERAL ADMIN TOOLBAR

[TODO DESCRIPTION]



[TODO DESCRIPTION New tags and CSS]

## CHAPTER 7: EXTENDING JAHIA

*This chapter describes ways to correctly extend (or fork) Jahia in order to ease migration and upgrades afterwards.*

### 1 EVENT LISTENERS

This section introduces event listeners, which is a powerful event based mechanism supported by Jahia to allow advanced customization.

Jahia has the capability to emit over 50 different types of events ranging from the creation, update or deletion of a field, a container or a page to a user login into Jahia. These events are generated by Jahia and dispatched by `org.jahia.services.events.JahiaEventGeneratorBaseService` to any internal listeners which extend the `JahiaEventListener` class. These listeners are declared in the `\WEB-INF\etc\config\listeners.registry` file, an excerpt of which is given below:

```
# Jahia Event Listeners Registry
#
# This file configures Jahia listeners that can intercept internal events such
# as content modifications, user login/logout, page loads, etc...
#
PortletsEventListener=jahiatemplates.org.jahia.portlets_api.PortletsEventListener
LoggingEventListener=org.jahia.services.audit.LoggingEventListener
LayoutManagerListener=org.jahia.layout.PortletsEventListener
ContainersChangeEventListeners=org.jahia.data.containers.ContainersChangeEventListeners
FieldsChangeEventListeners=org.jahia.services.containers.FieldsChangeEventListeners
SiteMapEventListener=org.jahia.data.viewhelper.sitemap.SiteMapEventListener
BlogPingListener=org.jahia.blogs.BlogPingListener
#
# Activate the JSP Event listener if you want to be able to implement JSP
# listeners using JSP files. In order to do this you can either modify the
# default /jsp/jahia/events/eventlistener.jsp file or include an
# "eventlistener.jsp" file in your template directory.
#
JSPEventListener=org.jahia.services.events.JSPEventListener
#
# Activate the Groovy Event listener if you want to be able to implement event
# listeners using Groovy files. In order to do this you can either modify the
# default /jsp/jahia/events/eventlistener.groovy file or include an
# "eventlistener.groovy" file in your template directory.
#
GroovyEventListener=org.jahia.services.events.GroovyEventListener
```

So for example, the above `org.jahia.data.viewhelper.sitemap.SiteMapEventListener` event listener intercepts messages relating to page creation or deletion operations. It enables Jahia to keep the sitemap up to date by invalidating the sitemap if any of these messages are caught.




Template developers can implement their own listener classes and register them in the `listeners.registry` file, however for most purposes, we recommend using the `JSPEventListener` mechanism or from Jahia 5 onwards the new `GroovyEventListener`.

## 1.1 SUPPORTED JAHIA EVENTS









Jahia identifies each event type by the `jahiaEvent` String value. Below are the currently supported events types:

[TODO >JAHIA VERSION]

Event Type	Description
<code>addContainerEngineAfterInit</code>	Triggered when the <i>add container popup engine</i> has successfully been initialized and loaded all related objects in <code>Engine_Map</code>
<code>addContainerEngineAfterSave</code>	Triggered just after saving a container in the <i>add container popup engine</i> .
<code>addContainerEngineBeforeSave</code>	Triggered just before saving a container, with all its mandatory fields are valid, in the <i>add container popup engine</i> . After this event is fired, the container is saved in the database.
<code>afterGroupActivation</code>	Event fired before <code>WorkflowService.activate(...)</code>
<code>afterServicesLoad</code>	Should be fired once all of Jahia's services are loaded (Not currently triggered)
<code>aggregatedContentActivation</code>	Aggregated event fired after dependent content objects have been activated, i.e. that <code>ContentObject.activate()</code> was called. The passed <code>jahiaEvent.getObject()</code> is a list of <code>ContentActivationEvent</code> objects, which hold the activated content objects within their <code>jahiaEvent.getObject()</code> .
<code>aggregatedContentObjectCreated</code>	Aggregated event fired once dependent content objects have been first created (stored in persistence). The passed <code>jahiaEvent.getObject()</code> is a list of <code>JahiaEvent</code> objects, which hold the created content objects within their <code>jahiaEvent.getObject()</code> .
<code>aggregatedEventsFlush</code>	Event fired to notify all aggregated events will be processed. The event's object is the list of all aggregated events.
<code>aggregatedObjectChanged</code>	Aggregated event triggered when the state of content objects changed (e.g. Language field marked for deletion, old version restored, new page definition set, new page parent set, new page title set, language version of page marked for deletion, container saved, file ACL changed, etc). The passed <code>jahiaEvent.getObject()</code> is a list of <code>WorkflowEvent</code> objects, which hold the changed content objects within their <code>jahiaEvent.getObject()</code> .
<code>beforeContainerActivation</code>	A container is about to be validated from the staging mode.
<code>beforeFieldActivation</code>	A field is about to be validated from the staging mode.
<code>beforeServicesLoad</code>	Should be fired before all of Jahia's services are loaded (Not currently triggered).
<code>beforeStagingContentIsDeleted</code>	Event is triggered before a staging content is being deleted.
<code>categoryUpdated</code>	Event fired when a category is updated
<code>containerAdded</code>	A container was added.

Event Type	Description
containerDeleted	Triggered when Jahia deletes a container
containerListPropertiesSet	Triggered when Jahia sets properties on a container list
containerUpdated	Event fired once a content object has been updated (changes stored in persistence)
containerValidation	Event is fired, when container is being validated in the Add or Update Container Engine.
 contentActivation	Event fired after a content object has been activated, i.e. That <code>ContentObject.activate(...)</code> was called.
contentObjectCreated	Event fired once a content object has been first created (stored in persistence ).
 contentObjectDelete	Event fired on content object delete
 contentObjectRestoreVersion	Event fired on content object restore version
 contentObjectUndoStaging	Event fired once a content object has been updated ( changes stored in persistence )
 contentObjectUpdated	Event fired once a content object has been updated ( changes stored in persistence )
fieldAdded	A field was added.
fieldDeleted	A field was deleted.
fieldUpdated	A field was updated.
fileManagerAclChanged	Event fired when ACLs on a Slide resource are modified. The <code>JahiaEvent</code> object contains the modified <code>DAVFileAccess</code> object with the new ACLs set.
 flushEsiCacheEvent	Event triggered when flushing the ESI cache
 groupAdded	Event triggered, when a user group was added
 groupDeleted	Event triggered, when a user group was deleted
 groupUpdated	Event triggered, when a user group was updated
 metadataEngineAfterInit	Event fired after the <code>Metadata_Engine</code> (which displays the popup that lets the user add a new container) has been initialized ( <code>engineMap init</code> ) and before processing last and current engine request.
 metadataEngineAfterSave	Event fired after the <code>Metadata_Engine</code> has saved the metadata fields for the current content object. The Event source object is the calling <code>Metadata_Engine</code> , the event



Event Type	Description
metadataEngineBeforeSave 	object is the <code>ObjectKey</code> instance of the content object. Event fired before the <code>Metadata_Engine</code> start to save the metadata fields for the current content metadata facade. The Event source object is the calling <code>Metadata_Engine</code> , the event object is a <code>ContentMet^adataFacade</code> instance.
objectChanged 	Triggered when the state of a content changes (e.g. Language field marked for deletion, old version restored, new page definition set, new page parent set, new page title set, language version of page marked for deletion, container saved, file ACL changed, etc)
pageAccepted 	Event triggered, when a page has been accepted for publication.
pageAdded pageDeleted 	A page was added. Triggered when Jahia adds a page Event triggered, when a page has been deleted.
pageLoaded pagePropertiesSet pageRejected 	Should be fired when a new page is loaded (Not currently triggered) Triggered when Jahia sets properties on a page Event triggered, when a page has been rejected and not published.
rightsSet siteAdded 	Triggered when Jahia sets ACL rights on a content object such as a container list, a container, a field or a page. Event triggered, when a new Jahia site has been added.
siteDeleted 	Event triggered, when a Jahia site has been deleted.
templateAdded 	Event triggered, when a new template has been added to Jahia.
templateDeleted 	Event triggered, when a template has been deleted.
templateUpdated 	Event triggered, when a template has been updated.
timeBasedPublishingEvent 	Event fired to notify a retention rule's state change
updateContainerEngineAfterInit updateContainerEngineBeforeSave	Triggered when the <i>update container popup engine</i> has successfully been initialized and loaded all related objects Triggered just before saving the updates to a container, with all its mandatory fields are valid, in the <i>update container popup engine</i> . After this event is fired, the container is saved in the database.
userAdded	Event triggered, when a user was added




Event Type	Description
 userDeleted	Event triggered, when a user was deleted
 userLoggedIn userLoggedOut userPropertiesSet	A user logged into Jahia. A user logged off Jahia. Event fired when a user property is set
 userUpdated	Event triggered, when a user was updated

Table 16: Listing of all Jahia events

## 1.2 USAGE

### 1.2.1 JSP EVENT LISTENER

The `JSPEventListener` class routes all events to a separate JSP file called `eventlistener.jsp`. A custom version of this JSP file should be placed in your template package root folder; otherwise Jahia uses the default file in `/jsp/jahia/events/eventlistener.jsp`. The advantage of using JSP files to catch events instead of Java class files is that there is no need to compile code or reboot Jahia for changes to take effect.

[TODO PICTURE]

The `eventlistener.jsp` file does not behave like other Jahia templates in that its output is not routed to the end-user. Instead, the JSP commands are executed and the output is ignored.

For each events thrown by Jahia, Jahia saves the event description into the request attribute which is accessible by `eventlistener.jsp`. The description consists of an `eventName` request attribute which is a `String` event type identifier. A `JahiaEvent` object is also stored under the `jahiaEvent` request attribute; the associated `JahiaEvent` holds:

- the timestamp when the event was triggered -accessible via `JahiaEvent.getTime()`.
- a reference to the object that generated the event -accessible via `JahiaEvent.getSource()`.
- a reference to the `ParamBean` object (with request and response) -accessible via `JahiaEvent.getParams()`.
- a reference to the object targeted by this event (e.g. the field id, the container id) accessible via `JahiaEvent.getTarget()`.



Note that the `eventlistener.jsp` file is executed once in its entirety for each event.

In this section, we will list simple examples scripts triggered from the `eventlistener.jsp` file by events dispatched by Jahia. Before going into production you should thoroughly test the impact on performance of capturing multiple events. Below are snippets located consecutively in a `eventlistener.jsp` file:

First we retrieve the current event from the session:

```
String eventName = (String) request.getAttribute("eventName");
JahiaEvent jahiaEvent = (JahiaEvent) request.getAttribute("jahiaEvent");
ParamBean jParams = jahiaEvent.getParams();
```

The following code will execute for every triggered event:

```
org.apache.log4j.Logger logger =
org.apache.log4j.Logger.getLogger(getClass());
System.out.println("The event currently being processed is : " + eventName);
```

The next script outputs a message to both the log file and the system console every time a user logs on by capturing all `userLoggedIn` events:

```
if ("userLoggedIn".equals(eventName)) {
    logger.info("User " + jParams.getUser().getUsername() + " logged in" );
    System.out.println("User logged in");
}
```

The next script outputs a similar message every time a user logs out:

```
if ("userLoggedOut".equals(eventName)) {
    logger.info("User " + jParams.getUser().getUsername() + " logged off" );
    System.out.println("User logged off");
}
```

Notice that full access to the Jahia's `ParamBean` is possible.

The next snippet outputs a messages every time a container is added:

```
if ("containerAdded".equals(eventName)) {
    JahiaContainer theContainer = (JahiaContainer)jahiaEvent.getObject();
    logger.info("Added container " + theContainer.getDefinition().getName() + " with ID "
        + theContainer.getID() + " thank you very much!");
}
```

The following code catches container deletion events and only outputs a message if the container's definition is `contentContainermain_1`:

```
if ("containerDeleted".equals(eventName)) {
    JahiaContainer theContainer = (JahiaContainer)jahiaEvent.getObject();
    if ( "contentContainermain_1".equals(theContainer.getDefinition().getName()) ) {
        logger.info("Deleted container of def name contentContainermain_1 with ID " +
            theContainer.getID() + " cheers!");
    }
}
```

We can implement a filtering mechanism so that the event processing code is skipped if the event doesn't belong to a restricted pre-defined set of events `eventsToTrap`:

```
if (eventsToTrap.size() == 0) {
    eventsToTrap.add("addContainerEngineAfterInit");
    eventsToTrap.add("addContainerEngineBeforeSave");
    eventsToTrap.add("containerAdded");
    eventsToTrap.add("containerUpdated");
}
if ( eventsToTrap.contains(eventName) ){
    // Put your event processing code here
}
```


The reader is advised to refer to the `eventlistener.jsp` file included in the demo templates for more examples.



Notice that some versions of the `eventlistener.jsp`, which were delivered with our demo templates were tailored to be used only with the standard Jahia engine popups. For instance if the `engineMap` was not found in the session, the event listener did not do anything. If you create content via the Jahia API, then the event listener should not depend on those engine objects.

## 1.2.2 GROOVY EVENT LISTENER

The disadvantage of the JSP Event Listener is that it only works in an environment, where a `HttpRequest` object is available. So for background tasks, which are not triggered by a request, or when using the `SoapParamBean`, which internally does not hold a request object, the JSP event listener is not triggered.

 That is why since Jahia 5 we have also introduced the possibility to implement the event listener with Groovy (see <http://groovy.codehaus.org>). This scripting technology has the same advantage as JSPs, so that there is no need to compile the code or to restart the server, when changes are implemented. It is also possible to simply copy the code from the JSP file to the `eventlistener.groovy` file located in your template directory. If the file is not existing, Jahia will look for the `EventListener.groovy` file in the directory `WEB-INF\var\scripts\groovy\events` (if you have set the `jahiaVarDiskPath` in `jahia.properties`, then you have to look there instead of `WEB-INF\var`).

In order to implement a filtering mechanism and to call the Groovy script only for the events, in which you are interested, you should use the following method:

```
// register comma separated list of events we want trap
GroovyEventListener.registerEvents("contentObjectCreated");
```

The `eventName` and `jahiaEvent` variables are already bound, so you can use them straight ahead like in the `eventlistener.jsp` file, where you had to get them from the request object first.

## 2 PORTLETS SUPPORT

[TODO DESCRIPTION chapter necessary ???]

## 3 INTEGRATE OWN STRUTS ACTIONS

[TODO DESCRIPTION chapter necessary ???]

## 4 DEPLOY NEW LIBRARIES (JARs)

[TODO DESCRIPTION warn users about possible problems of installing a new version of an existing JAR on the Jahia correct behavior]

## CHAPTER 8: ADVANCED TOPICS

This chapter will throw some light on topics for advanced Jahia template developers, like the usage of event listeners, cache control, schedulers,...

### 1 INTERNATIONALIZATION

#### 1.1 WHAT IS INTERNATIONALIZATION?

Since most websites are now viewed by speakers of different backgrounds, enabling your website to serve different versions of content translated into the appropriate language is essential. Making your website global is termed *Internationalization*, also referred to as *i18n*. Its goal is to minimize the engineering changes in the code to add support for a multitude of languages and regions to your website. The process is referred to as *Localization* and consists of simply adding locale-specific components translated into the targeted languages. These components are called resource bundles and are usually specific to a given locale. A locale is defined for a specific cultural or geographical region; it is therefore typically composed of two codes: the language code and the (optional) country code separated by an underscore such that:

- fr\_CH is French for speakers in Switzerland
- fr\_FR is French for speakers in France
- de\_DE is German for speakers in Germany
- en\_US is English for speakers in American
- en\_UK is for proper English

The language codes are defined in ISO639 and the optional country code is defined in ISO3166. We urge readers not familiar with localization to visit the many great resources on the Web on this topic (see Appendix section for references).



Most browsers allow users to specify their language preferences. For example, with IE7, they can set the order of preferred languages via the *Extras -> Internet Options -> Languages* button.

#### 1.2 RESOURCE BUNDLES

Resource bundles contain key/value pairs for specific locales used during internationalization. They group various label strings, CSS class URLs, image URLs, and various other objects that are locale dependent. The locale code is appended to the base name you have given to your resource bundle. This yields resource bundle filenames of the following format:

Resource Bundle Name	Details
<code>Simple_templates.properties</code>	This is the base name of the resource bundle
<code>Simple_templates_en.properties</code>	Used for English locales when country code isn't available
<code>Simple_templates_en_US.properties</code>	Used for English speakers in America
<code>Simple_templates_en_UK.properties</code>	Used for English speakers in the UK
<code>Simple_templates_fr_CH.properties</code>	Used for French speakers in the Switzerland

**Table 17: Resource bundle file names examples for different locales**

These files should be located in the following paths:

```
tomcat\webapps\jahia\WEB-INF\classes\jahiatemplates\Simple_templates.properties
```

```
tomcat\webapps\jahia\WEB-INF\classes\jahiatemplates\Simple_templates_en.properties
tomcat\webapps\jahia\WEB-INF\classes\jahiatemplates\Simple_templates_en_US.properties
tomcat\webapps\jahia\WEB-INF\classes\jahiatemplates\Simple_templates_en_UK.properties
tomcat\webapps\jahia\WEB-INF\classes\jahiatemplates\Simple_templates_fr_CH.properties
```

Resource bundles from all template packages are located here. These files are copied automatically during deployment of a template package if the resource bundles are appropriately declared in the `templates.xml` (more details below).

Below is a snippet of the `Simple_templates.properties` resource bundle file. Assuming `en_UK` is the default language (it is therefore identical to `Simple_templates_en_UK.properties`), we get:

```
...
myContainerList.title = Title
content = Content
file = File
Food = Beans
Lift = Lift
org.jahia.stylesheet.css = /jsp/jahia/engines/css/jahia_beer.css
header.image = /jsp/jahia/templates/myjahiasite/Simple_templates/images/header.gif
...
```

Notice the key/value format. The bundle `Simple_templates_en_US.properties` would look like this:

```
myContainerList.title = Title
content = Content
file = File
Food = Beans
Lift = Lift
org.jahia.stylesheet.css = /jsp/jahia/engines/css/jahia_beer.css
header.image = /jsp/jahia/templates/myjahiasite/Simple_templates/images/header.gif
...
```

Notice the key/value format. The bundle `Simple_templates_en_US.properties` would look like this:

```
...
myContainerList.title = Title
content = Content
file = File
Food = Burgers
Lift = Lift
org.jahia.stylesheet.css = /jsp/jahia/engines/css/jahia_coke.css
header.image = /jsp/jahia/templates/myjahiasite/Simple_templates/images/header_US.gif
...
```

And finally, `Simple_templates_fr_CH.properties`:

```
...
myContainerList.title = Titre
content = Contenu
file = Fichier
Food = Tartiflette
Lift = Ascenseur
org.jahia.stylesheet.css = /jsp/jahia/engines/css/jahia_milk.css
header.image =
/jsp/jahia/templates/myjahiasite/Simple_templates/images/header_fr_CH.gif
...
```

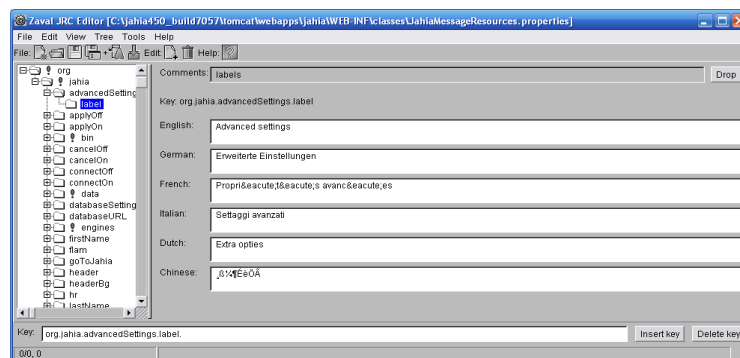
Note how some entries point to physical files such as CSS or images. The paths must be valid relative paths. These paths are relative to the *Jahia Webapp Context* directory. You can use any key naming convention as long as they are unique across all resource bundles with the same base name. In this example, the key `myContainerList.title` will be used for the title of a container list called `myContainerList`.

An other important fact to remember is that resource bundles are loaded during Tomcat's startup. So if you modify them, changes will only take effect after restarting Tomcat.

### 1.3 BUNDLE EDITOR

Editing resource bundles can be quite tedious so we recommend using a visual resource editor to streamline editing. For example, you can use the freely available Zaval JRC Editor (<http://www.zaval.org/products/jrc-editor/download/index.html>). A screenshot is given below:

[TODO BETTER SCREENSHOT]



Picture 8.1:

### 1.4 USAGE

There are multiple ways to make use of the data inside resource bundles. We describe four resource accessors:

- specialized resource tags
- tags which support resource bundles
- script-based and
- marker-based.

### 1.5 RESOURCE BUNDLE TAGS

Jahia provides the `content:resourceBundle` tag to display a specific value in a resource bundle:

```
<content:resourceBundle resourceBundle="myjahiasite_Simple_templates"
  resourceName="live"/>
```

This tag can be used to dynamically request a given resource from the engine resource bundle. If user's locale is not set, the locale used is the one returned by `paramBean.getLocale()`. The attribute `resourceBundle` holds the name of the resource bundle, where the `live` key is looked up. The value `myjahiasite_Simple_templates` isn't the real name of the resource bundle file but a lookup key to it in the `resourcebundles_config.xml` file located in `tomcat\jahia\WEB-INF\etc\config`. Below is an example listing of such a file:

```
<?xml version="1.0" encoding="UTF-8"?>
<registry>
  <resource-bundle>
```

```
<key>myjahiasite_Simple_templates</key>
<file>Simple_templates</file>
</resource-bundle>
</registry>
```

which corresponds to the physical file `Simple_templates.properties`. This corresponds to the bundle for the default language, which is used if a resource bundle for the user's current locale isn't defined.

Jahia also supports the `content:message` tag which is functionally similar to `content:resourceBundle`, except that the lookup is hard-coded to the `JahiaMessageResources.properties` resource bundle:

```
<content:message
  key="org.jahia.bin.JahiaConfigurationWizard.root.adminUserName.label"/>
```

It is inspired by the Struts bean message tag except that it checks a session variable (`org.jahia.services.multilang.currentlocale`) to determine if a locale has been chosen. Other related tags include the `content:adminResourceBundle` and `content:engineResourceBundle` tag, which are used to dynamically request a given resource from the admin resource bundle and the engine resource bundle respectively. The reader is referred to the Jahia Taglib Documentation for further details.

## 1.6 TAGS WITH NATIVE SUPPORT FOR INTERNATIONALIZATION

Some tags natively support resource bundles such as the `jahiaHtml:actionMenu` tag for which we give an example below:

```
<jahiaHtml:actionMenu name='testField' namePostFix=' '
  resourceBundle='jahiatemplates.Corporate_portal_templates' useFieldSet='false'/>
```

This basically displays the GUI interface, in the language closest to the current user's locale, for actions on the enclosing Jahia content. The `jahiatemplates.Corporate_portal_templates` resource bundle should contain all keys utilized by the `jahiaHtml:actionMenu` tag. The `content:tabButton` allows template developer to specify the resource bundle and the key to use for the label of the rendered button. The `content:declareField` and `content:declareContainerList` tags also support internationalized titles so that the engine displays the title in the appropriate language. Other noteworthy tags are:

- the `content:displayLanguageCode` tag displays the current language.
- the `content:displayLanguageFlag` tag displays the flag associated to the current language.
- the `content:i18n` tag sets the encoding charset and the content type of the page.

This is called at the beginning by all templates. You can set your own `charset/content-type` using its attributes but we advise against doing so. This is because Jahia might set the `charset/content-type` before this tags are executed. This might cause unpredictable results. We recommend you do thorough tests before going live.

## 1.7 RESOURCE MARKERS

It is also possible to use resource bundle markers to access resources in bundles. They behave much like an Expression Language version of the `content:resourceBundle` tag. So for example the following marker:

```
<content:declareField name='body' title='Body' type='BigText' titleKey='blog.body'
  bundleKey='<%=resBundleID%>' value='<jahia-resource id="myjahiasite_DOC_TEMPLATES"
  key="insertContent" default-value="Insert Content here"/>' />
```



would initialize the `BigText` field to the text at key `insertContent` of resource bundle `myjahiasite_DOC_TEMPLATE` and defaults to “Insert Content here” if it cannot find it. Note that the field's title is also internationalized. Below is an example of how resource markers are used to generate an internationalized `multi_value` marker:

```
<%
//creates the multivalue marker given a series of keys and a resourceBundle identifier
public String getMultivalues(String resBundleID, String values[]){
    StringBuffer sb = new StringBuffer("<jahia_multivalue[ :");
    ArrayList al = new ArrayList();
    for (int i = 0; i < values.length; i++){
        al.add( values[i] );
    }
    Collections.sort(al);
    for (int i = 0; i < al.size(); i++) {
        if (i > 0){
            sb.append(":");
        }
        //creates a jahia-resource marker
        sb.append(ResourceBundleMarker.drawMarker(resBundleID, (String)al.get(i), ""));
    }
    sb.append(">");
    if (al.size() >= 0){
        sb.append(ResourceBundleMarker.drawMarker(resBundleID, (String)al.get(0), ""));
    }
    return sb.toString();
} // getMultivalues

//Initialise the options available in the internationalized multi-value marker
String boxTitleTypes[] = {"greyBackground",
    "transparent", "coloredBorder", "greyBorder", "coloredBackground"};
//generate the name of the resource bundle to use
String resBundleID = jParams.getSite().getSiteKey() + "_Basic_templates";
%>

<content:declareField name='boxTitleLayout' title='Title Layout' type='SharedSmallText'
    titleKey='titleLayout' bundleKey='<%=resBundleID%>'
    value='<%=getMultivalues(resBundleID, boxTitleTypes)%>' />
```

This would generate something of the form for an English locale:

```
value="<jahia_multivalue[grey background:transparent:colored border:grey border:colored
background]>"
```

And something like this for the French locale:

```
value="<jahia_multivalue[fond gris:transparent:bords colores:bords gris:fond colore]>"
```

## 1.8 SCRIPT-BASED INTERNATIONALIZATION

Of course it is possible to carry out all internationalization operation using the Jahia API directly. For example, we can rewrite the first example above in the following manner:

```
<content:declareField name='body' title='Body' type='BigText' titleKey='blog.body'
    bundleKey='<%=resBundleID%>' value='<%=getResourceBundle ("insertContent", "Insert
content here", jData)%>' />
```

with

```
<%
public String getResourceBundle(String resourceName, String defaultValue, JahiaData
jData) {
    String resourceBundle = "jahiatemplates.Simple_templates";

    ResourceBundle res = null;
    String resValue = null;
    try {
        res = ResourceBundle.getBundle(resourceBundle, jData.params().getLocale());
        resValue = res.getString(resourceName);
    } catch (MissingResourceException mre) {
    }
    if (resValue == null) {
        resValue = defaultValue;
    }
    return resValue;
} // getResourceBundle
%>
```

## 1.9 CHANGING CURRENT LANGUAGE ON-THE-FLY

The listing below is a useful way to display all available languages except the current one and allow the user to switch between them. The output will be of the form:

[TODO SCREENSHOT][French | English | German]

```
<%
// returns the list of active languages for the current site
ArrayList languageSettingsAsLocales =
    jParams.getSite().getLanguageSettingsAsLocales(true);
for (int i = 0; i < languageSettingsAsLocales.size(); i++) {
    Locale loc = (Locale) languageSettingsAsLocales.get(i);
    String lc = loc.toString();
    Integer languageState = (Integer) languagesStates.get(lc);
    boolean displayLink = false;
    String currentLanguageCode = jParams.getLocale().toString();
    // don't display current language
    if (!currentLanguageCode.equals(lc)) {
        if (languageState != null) {
            // live mode
            if (jData.gui().isNormalMode()) {
                // only display the link if a page is active in current language
                if (cp.hasEntries(ContentPage.ACTIVE_PAGE_INFOS, lc)) {
                    displayLink = true;
                }
            }
            // preview & edit mode
        } else {
            displayLink = true;
        }
    }
    if (displayLink) {
%> | <a href="<%=jData.gui().drawPageLanguageSwitch(lc) %>"><%=lc%></a> <%
    }
}
%>
```

## 1.10 ENGINE INTERNATIONALIZATION

Please refer to Section Engine Pop-up Customization for details on how to customize Jahia's various engine and admin popups to support multiple languages.

## 2 CACHE ISSUES

Jahia uses a multi-layered caching system to offer best performance regardless of usage scenario. But in some usage scenarios these caches might be a little too aggressive. In this section we will not cover the back-end caches or the web application caches, since they are not relevant to template developers.

In Jahia 4 the only integrated front-end cache was the HTML-cache. By integrated we mean, that Jahia has a tracking system to know, which cached pages need to be invalidated, when content has been changed and approved. You could also use the OSCache for fragments caching in Jahia 4, but you would have to control the life-cycle on your own. With Jahia 5 we now also offer a fully integrated fragments cache, called ESI (Edge side include) cache.

### 2.1 USING HTMLCACHE

Before showing how to control the cache system from a template, let's quickly look at how the front-end HTMLCache works in Jahia.

#### 2.1.1 REQUEST PROCESSING FLOW

In the graphic below, we show the flow of request processing for a Jahia page. As you can see the front-end cache is a central part of a request. The template is called only if there was no existing entry in the cache.

[TODO PICTURE]

Jahia offers a mechanism to control front-end cache expiration, which we present below.

#### 2.1.2 CONTROLLING CACHE EXPIRATION

Front-end cache control is done by specifying the expiration delay of a page. Here is an example of setting the expiration delay to 5 seconds :

```
<%  
long cacheExpirationDelay = 5000; // [ms]  
jParams.setCacheExpirationDelay(cacheExpirationDelay);  
%>
```

There are other possible values for the delay which include :

- 0 : the page will expire immediately
- never setting it : the page will never expire in time, but will be subject to refreshing based upon user modifications.

One example scenario for setting the cache expiration delay is for example if you are using templates to include RSS news feeds from other sites. In this case you will want to set the expiration delay to whatever the refresh rate of the RSS feed is.

#### 2.1.3 DISPLAY DETAIL PAGE BY USING REQUEST PARAMETER

[TODO DESCRIPTION]

Some templates display a container list and by clicking on a container a detailed view of the container (same page-ID, just a request parameter is suggesting to display details of a container). For such pages only the overview page should be cached. To prevent that detail pages go to the cache you should always use 'cache/bypass' in the URLs for the detailed page.

## 2.2 USING FRAGMENTS CACHE

The above mentioned front-end cache always stores the entire HTML page. As the page could be different per user, per workflow-state, per language, per scheme (http/https) and perhaps per user agent (browser), the cache can grow very fast depending on the page size and whether you have many logged in users.

An alternative or additional performance gain can be achieved by using fragment caches. Here we present two alternatives to do that:

### 2.2.1 ESI CACHE

For intranet or site with heavy personalization Jahia developed the first ESI server with full source code available (based on [www.esi.org](http://www.esi.org)). This remote cache server allows you to aggregate and share multiple fragments per page. The main goal of ESI is to share HTML fragments of a page and to regroup similar fragments per group of users.

This advanced fragments cache is fully embedded in Jahia and will automatically invalidate as soon as content has been changed and approved in edit mode.

If you would like to use the ESI cache please get all the needed information in the JESI Caching Quickstart documentation available on the Jahia homepage.

### 2.2.2 OSCACHE

The OSCache is a light-weight alternative to the ESI cache, but is not fully embedded in Jahia, what means that you would have to control the cache invalidation on your own, perhaps by using event listeners.

The OSCache can perfectly be used for fragments of the page, which take much time to render and the result is always the same, especially if it is the same on every page (header, footer, menu, last 5 news,...). However you might want to use it only in LIVE mode, if you want to show the edit menu items in EDIT mode.

You could do it like this:

```
<%
ContainerBean boxContainer = (ContainerBean)
    pageContext.getAttribute("boxContainerBean");
int boxContainerId = boxContainer.getId();
boolean isCacheActive = jData.gui().isNormalMode();
%>

<cache:cache time='<%= isCacheActive ? 120 : 0 %>' key='<%= "box" + (new
    Integer(boxContainerId)).toString() %>'>
    [... the display of the box...]
</cache:cache>
```

In this example the box will be cached in LIVE mode for 120 seconds. You can decide, whether you want to cache a fragment for minutes or hours. It depends how often it changes. For instance the header and footer will not change often. For the navigation, it might happen more frequent that new pages are coming or old pages are being deleted. In order to prevent 404 errors, if a fragment is still showing a page, which has been deleted, you could also call

```
<cache:flush key='<%= "box" + (new Integer(boxContainerId)).toString() %>' />
```

in EDIT mode on the JSP, where the container gets changed.

For the key, it also depends whether you are using access rights on the navigation or the OScached container list, so each user might have a different view. Then you might want to add the user-id to the key, then the fragment will be cached per user. You should surely limit the time on these fragments.

To use the OScache you need to deploy the JAR to the lib directory of the Jahia web application, you need to add the `oscache.tld` to the `web.xml` and then you can use the taglib in your templates.

## 3 CROSS-LINK SYSTEM

This advanced chapter introduces the concept of Links which are used by Jahia to store relationships between various objects.

### 3.1 INTRODUCTION

Currently in Jahia we have a system of cross references that stores data only in RAM, and has no persistence. Basically we store a cross reference in the following format:

[TODO PICTURE]

So in effect if the left argument is for example a container list, and the right arguments are the pages that reference this container list, we are representing in RAM the collection of pages that reference this container list (in either a direct or absolute way, the distinction is not currently made in Jahia).

This system has been satisfactory in Jahia to implement the current cross references required to maintain the consistence of the HTML cache system. Basically it allows us to interrogate the cross reference system to know which pages must be flushed when a certain container list is modified. The same is done for the fields and containers.

### 3.2 ADDING PERSISTENCE

The problem with the in RAM-only system is that it needs to be recreated entirely from scratch every time a Jahia system is restarted. The upside of this is that only the existing relationships are kept, and the ones that have been removed no longer exist in memory. But this functionality could also be done with a persistent storage mechanism, at the cost of consistency checkers implementations.

In the search for a database format for storing cross reference links, the idea of a generic system of link came to light. Basically, can we store more than reference links if we choose a generic storage system?

### 3.3 GENERIC LINKS

In order to answer this question, it is easier to consider a graph of links between objects. One thing that is important to remember is that we are only looking at instance objects, not object types. Entity-relation graphs represent relationships between types (or classes) of objects, but here the interest is on instances of objects.

[TODO PICTURE]

The illustration above shows how objects and links are related. Here we can see three different “types” of links represented :

- **reference links** : these links allow to offer a simple reference link, such as the one we used until now for building cross reference tables
- **category links** : these links indicate the membership of an object to another object that is a category (this is actually going backwards the arrow, a left-to- right description would be “Protocols includes Object 1”).
- **semantic links** : these links indicate similitude in the semantic of the object. This builds a notion of sense for the objects. This is very similar to the types of links that Tim Berners-Lee described in his book “Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web” (ISBN: 006251587X) and allows to create webs of semantic links that allow to navigate within the meaning of objects.

We will look at examples of other types of links that could be interesting, but first let's study what is needed to fully categorize a link, and what information could be useful to have in a content management system such as Jahia.

As we can see there are different directions for links : unidirectional and bidirectional. Actually we can bring it all down to unidirectional since bidirectional links can be fully represented by using two unidirectional links between the same objects.

So far we have the following information to describe a link :

- its left argument, or left object, or the source of the link
- its right argument, or right object, or the destination of the link
- the link type (for example : “semantic”, “category”, “reference”, “parent”, “child”, “language(or translation)”, and so on...)
- the creation date
- the creation user
- the last modification date
- the last modification user
- link metadata : this is an important part. it allows to add an extensible set of attributes within a link. Link metadata itself can be categorized in 3 sub-sections: left arg metadata, right arg metadata and common metadata.
- status of the link : is the link currently active ? It could be useful to be able to change the status of a link to temporarily deactivate it and then restore it to its initial values (if we just delete we loose the link information originally stored)

Some of the link descriptions we have done up until now can be compared to the X-link specification available at the W3 Consortium. Corroborating the two link system could maybe bring up omissions in the current system or interesting ideas to expand on this idea.

#### STILL TO BE RESOLVED

- What about workflow state of content objects ?
- What about the language of content objects ?
- What about the version (ID) of content objects ?
- What happens when we move objects ? Is this equivalent to changing links ?

### 3.4 LINK EXAMPLES

We present here a few examples of possible link types, and how they are used, especially using metadata information.

#### LANGUAGE LINK (OR TRANSLATION LINK)

left arg metadata : languageCode=en

right arg metadata : languageCode=fr

This represents a link between an English content object and a French content object indicating that it has the same semantic of content and a language relationship.

### CATEGORY LINK (OR “BELONGS TO”)

This link type establishes a category link between two objects, from left to right, looking something like this:

[TODO PICTURE]

where the “Object” could also be a category of course.

### REFERENCE LINK (OR CROSS REFERENCE)

This type of link is used to create a reference (in hypertext sense between two objects).

[TODO PICTURE]

Types of reference usage : “More info...”, “Related”

### OPEN QUESTIONS

References seem close to semantic links, or more generally, can we categorize all links as semantic links ? Maybe we should avoid using the “semantic” type for links and only use specialized link types because they convey the actual sense of the link ?

## 3.5 DATABASE TABLE FORMAT

Here is a tentative database table format for representing the links. The first table represents the actual link data.

ID	left_OID	right_OID	type	status	creation_date	creation_user	lastmodif_date	last_modif_user
1	category	object1	category	on	2003/02/22 01:00	bill:1	2003/03/03 01:00	steve:2
2	object1	object2	reference	off	2003/02/22 02:00	steve:2	2003/03/03 02:00	bill:1
3	object1	object2	language	on	2003/02/22 03:00	larry:1	2003/03/03 03:00	tux:3
4	category	category	synonym	on	2003/02/22 04:00	tux:3	2003/03/03 04:00	larry:1
	1	2						

Table 18: Link table

The second table contains the link metadata, which is extensible to an infinite number of parameters, allowing the links to have a varying number of parameters for their metadata.

link_ID	position	name	value
3	left	languageCode	en
3	right	languageCode	fr
4	common	languageCode	en
2	common	expirationDate	2003/02/23

Table 19: Link metadata (or properties) table

## 4 SCHEDULER

[TODO DESCRIPTION Quartz jobs]

## 5 INSTALLING, DEPLOYING AND MIGRATING YOUR TEMPLATES

[TODO DESCRIPTION explain some key parameters such as template recompilation between live and edit mode (in the jahia.properties), precompilation of templates in the web.xml, tips and tricks on how to modify an existing template (directly in the JSP directory - by redeploying a full jar - do you need some reboot or to flush the work directory or not, etc...). It would be also nice to give some tips about the right way to store resource bundles or so on so that you can then easily migrate your changes from one release of Jahia to the other. The template developer should keep in mind that someone will have to maintain the system and if he is forking all the current resource bundle, they will be certainly override by our new product bundles available in the latest version of Jahia. ]

### 5.1 PACKAGING YOUR TEMPLATES

Jahia allows you to package your custom templates into an easily distributable JAR file. This feature enables you to easily allow administrators to employ these templates during future virtual site creation operations.

We will now look at the steps required to construct this JAR file and how to subsequently deploy it.

#### 5.1.1 PACKAGE CONSTITUENTS

A template *package* consists of the following components:

1. An icon symbolizing this template package, which will be displayed in the template set selection page during virtual site creation. An example of which is given below:

[TODO SCREENSHOT]

2. A set of JSP files along with any other utilized files such as image files or CSS files.
3. A `templates.xml` descriptor: describing the template package overall properties, declaring the various JSP supported by this template package along with a few properties.
4. A set of (optional) Java class files if required by the template set.
5. A set of (optional) resource bundles if required by the template set.

Components (2) and (3) are bundled together into a single JAR file (or in a single directory). Similarly, components (4) and (5) are bundled together into a single JAR file, which is placed inside the previous JAR file (or directory). This forms a single JAR files (or directory) containing all the files necessary for the template package (except the icon in (1)).

#### 5.1.2 DETAILED EXAMPLE

We now illustrate the construction of a simple template package consisting of a couple of JSP template files which call a custom class. Below is a list of the necessary components along with the locations they need to be placed at:

Component	Details	Target
1	The package icon: <code>simple_templates.gif</code>	Copy to <code>\tomcat\webapps\jahia\jsp\jahia\templates_pre view</code>
2	Two JSP template files: <code>homeAgenda.jsp</code>	Package these files into



Component	Details	Target
3	and <code>simpleAgenda.jsp</code> along with a few necessary files in <code>\css</code> and <code>\images</code> The XML descriptor: <code>templates.xml</code>	<code>simple_jahiatemplates.jar</code> and do any one of the following options: <ul style="list-style-type: none"> <li>● Copy it to <code>\tomcat\webapps\jahia\WEBINF\var\shared_templates</code> so that the template package is accessible to all virtual sites on your server.</li> <li>● Copy to <code>\tomcat\webapps\jahia\WEBINF\var\templates\[sitekey]</code> so that the template package is accessible to the virtual site with site key <code>[sitekey].(*)</code></li> <li>● Add the template package manually through Jahia's administration popup.</li> </ul>
4	The custom class required by both JSP files: <code>agenda.class</code>	Package these files into an archive also called <code>simple_jahiatemplateclasses.jar</code> and place it in the root of the above
5	Two resource bundles: <code>simple_templates.properties</code> and <code>simple_templates_en.properties</code>	<code>simple_jahiatemplates.jar</code> .  Once you deploy your template package, this .jar is copied to <code>\tomcat\webapps\jahia\WEBINF\jahiatemplates</code> . Since other template packages also copy their own .jar files in this directory, be careful to give your classes a unique name or package name. This also applies to the resource bundle.  In case the names are identical, the most recently deployed files will overwrite the existing files.

Table 20: Steps to create a Jahia template package

Below is a the contents of the associated and self-explanatory `templates.xml` file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tpml>
  <parameter name="package-name">Simple templates</parameter>
  <parameter name="root-folder">simple_templates</parameter>
  <parameter name="classes-file">simple_jahiatemplateclasses.jar</parameter>
  <parameter name="provider">http://www.jahia.org</parameter>
  <parameter name="thumbnail">simple_templates.gif</parameter>
  <template browsable="1" visible="1" homepage="1">
    <parameter name="name">home</parameter>
    <parameter name="filename">home.jsp</parameter>
    <parameter name="display-name">Home</parameter>
  </template>
  <template browsable="1" visible="1">
    <parameter name="name">standard</parameter>
    <parameter name="filename">standard.jsp</parameter>
    <parameter name="display-name">Standard</parameter>
  </template>
</tpml>
```

Now let's look at some of the fields in the above listing in more detail:

- `root-folder` is the directory where the JAR (or directory) file will be decompressed (or copied) into in `\var\templates\[sitekey]` or `\var\shared_templates`.
- `classes-file` is the name of the JAR file inside the package template JAR file (or directory), where components (3) and (4) are located.
- `package-name` is display name of this template package.
- To each template JSP file is associated a `<template>` node:
- `visible="1"` means that the template will be appear in the list of available templates by the users when adding a new page. Default value is 1. You can change this value later on in the Administration panel.
- `homepage="1"` means that this template can be used as a homepage. Default value is 0. Setting a homepage template in a template package is optional as long as you have another template package for your site which does declare a homepage template.
- `name="name"` is the unique identifier of the template and you should avoid using whitespaces in it.
- `name="display-name"` is the display name of the template, and is set to `name="name"`'s value if `name="name"` is not set.
- `browsable="1"` is now deprecated.

(\*) Make sure that you have Automatic Deployment check box active in the Administration panel as depicted below so that Jahia can automatically dispatch your new template packages:

[TODO SCREENSHOT]

when the templates have been added, you'll get the following console messages in DEBUG mode:

```
264953 [JahiaQuartzScheduler_Worker-0] DEBUG
org.jahia.services.templates_deployer.JahiaTemplatesDeployerBaseService.deploy:182 - Trying to deploy
C:\jahia\tomcat\webapps\jahia\WEB-INF\var\new_templates\myjahiasite\simple_jahiatemplates...
264969 [JahiaQuartzScheduler_Worker-0] DEBUG
org.jahia.services.templates_deployer.JahiaTemplatesDeployerBaseService.deploy:215 - Delete meta-inf folder=C:\jahia\tomcat\webapps\jahia\jsp\jahia\templates\myjahiasite\simple_templates
266032 [JahiaQuartzScheduler_Worker-0] DEBUG
org.jahia.services.templates_deployer.JahiaTemplatesDeployerBaseService.registerTemplates:482 - Added template
homeAgenda.jsp
266063 [JahiaQuartzScheduler_Worker-0] DEBUG
org.jahia.services.templates_deployer.JahiaTemplatesDeployerBaseService.registerTemplates:482 - Added template
simpleAgenda.jsp
```

---

## CHAPTER 9: APPENDIX

### 1 TAGLIB TUTORIAL

**jahia**

>5.1 [TODO CODE EXAMPLE

for now see Chapter 3 and Chapter 5 in previous Template guide:

[https://www.jahia.net/jahia/webdav/site/jahia\\_net/shared/Documentation/Draft\\_Jahia\\_Template\\_Developer\\_Guide\\_v0.32.pdf](https://www.jahia.net/jahia/webdav/site/jahia_net/shared/Documentation/Draft_Jahia_Template_Developer_Guide_v0.32.pdf)

### 2 REFERENCES

[TODO DESCRIPTION URLs, other guides]

---

## ***LIST OF ILLUSTRATIONS***

--