Documentation

Jahia EE v6

Templating Mechanisms

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

May 2009

Jahia

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

www.jahia.com
The Company website

www.jahia.org
The Community website

Summary

1 Overview

This document is focusing on Jahia Templating mechanisms. It is not the full Jahia Templating Guide, but a large overview of Jahia EE v6 templating. For Jahia developers already working with Jahia 5, we have paid attention to emphasize specifically the differences between both versions.

If the Jahia main concepts did not changed, the templating work has been widely modified, simplified and its capabilities were extended.

The main changes are:

- ✓ Externalized content definitions
- ✓ Taglib reorganization
- ✓ Tags simplification
- ✓ 0 scriplets in templates provided by Jahia
- ✓ 100% tags and JSTL templates
- ✓ Template inheritance mechanism
- ✓ Resource bundle refactoring
- ✓ Theme management
- ✓ Skins management

Templates:

- ✓ No API calls, no scriplets
- ✓ Use Jahia tags
- ✓ Use JSTL
- ✓ XHTML standard

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

2 Jahia Objects

2.1 Fields

Fields are the smallest content entities manipulated within Jahia. Whatever the type, content is stored in fields: pages, texts, images, booleans, integers, float, etc.

Jahia allows integrators to manipulate fields without having to create their own data model, or SQL queries, in fact Jahia provides a complete Taglib to deal with fields and does all the dirty back-end job for you.

In a way, the only role devoted to integrators is to use the fields within their templates, where they want to give access to data (editing or displaying). In comparison to some other CMS, which provide only unstructured text areas to store content, Jahia offers a structured approach, allowing to declare fields, but also to group them in coherent larger entities: containers.

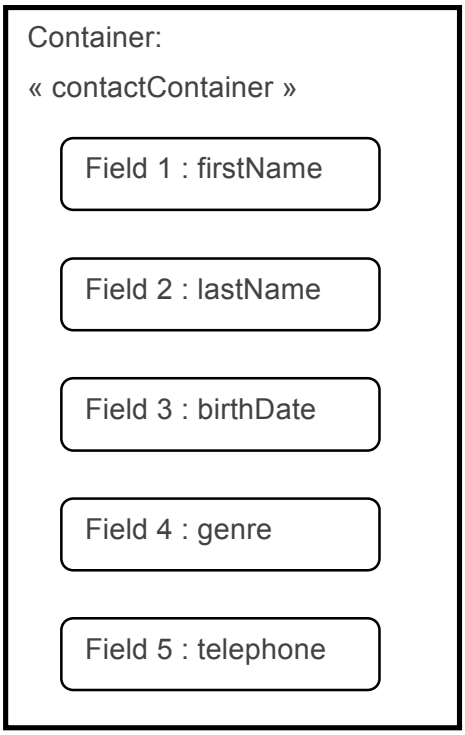
A field is defined by a name and a type

Field

2.2 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). For example we will use a container to represent a contact. The grouping of fields in containers also defines the ordering (or ranking) of the fields within the contact, which will be used by Jahia's edition tools to present the content to the user. This way when a new contact must be entered, the order of the fields presented will match the order of the fields when declaring the contact container. There is also a possibility to define the order and grouping of fields, by using a definitions.grp file.

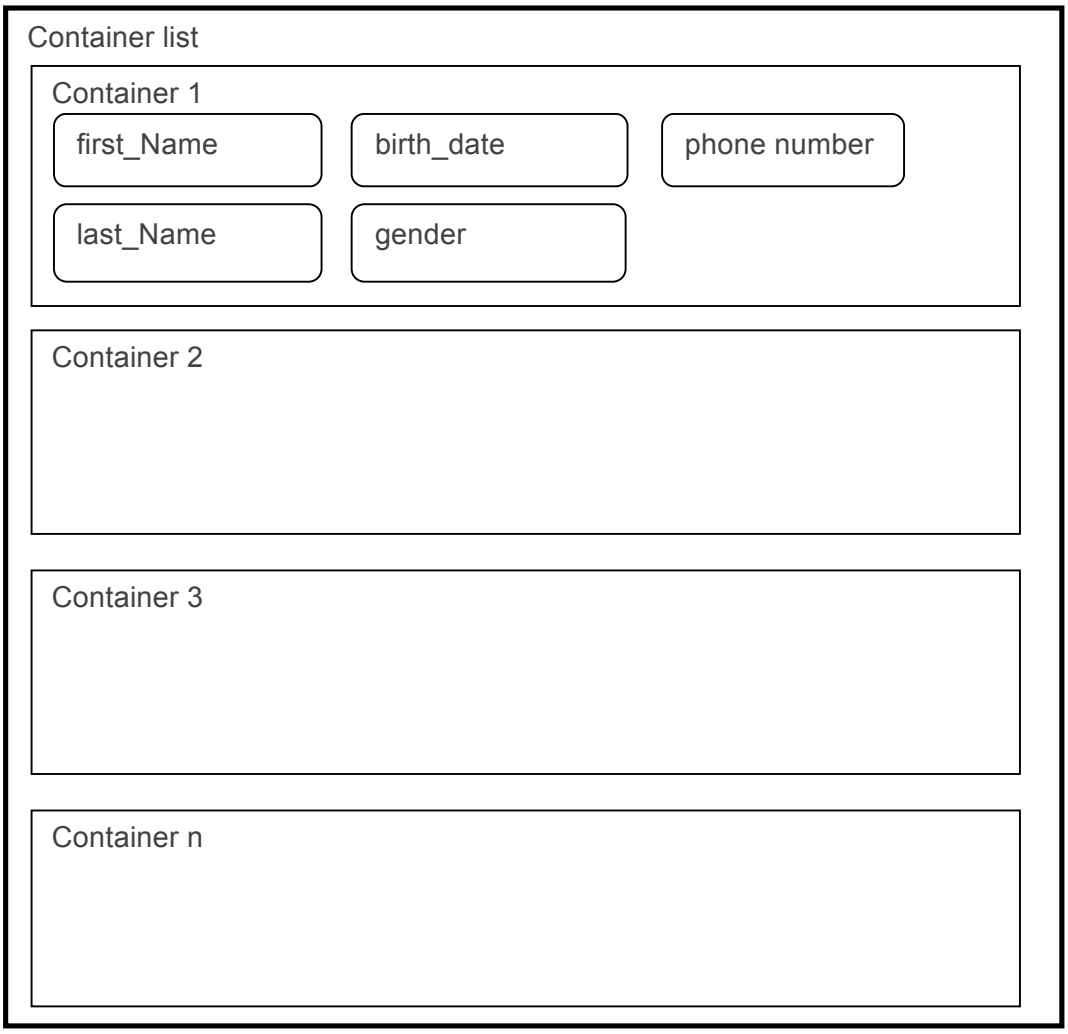
As we are able to identify various fields by a definition name, a container definition must also have a name. In this case the most obvious name for the container would be *contactContainer*, since this is what our logical grouping of fields will represent. So finally, we have a container definition that looks like this:



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

2.3 Container Lists

A container list is basically a list of containers. It also defines an order, which by default is the order the containers were added into the container list. Container lists may only be composed of containers of a specific definition.



Unlike 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. We can mix fields and sub container lists within a container structure. It is possible to include several sub container lists of different structures within a container.

An advanced way to leverage 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 for Jahia's box container structure, which in Jahia EE v6 is more elegantly supported with some easy-to-use tags.

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 be presented with the possibility of adding sub-containers for the container he has just added.

2.4 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 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 template has two responsibilities:

- Declaring the content structure for the page
- Rendering the instance of the content objects that comply to the structure

When a new page is created in Jahia, the content structure is used to present the entire matching user interface for input of new data. The next step is to load all the existing instances of content objects and to display them (of course there won't be any for a brand new page, but there are cases where objects will be created automatically upon first access, like in the case of mandatory containers).

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.

Until Jahia 5 the content definitions were done directly in the templates by using tags. From Jahia EE v6 onwards, content definitions are externalized in one or several Java Content Repository (JCR) inspired CND files. It is still 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.

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.

2.4.1 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 `pageField` and the pageField is a child of the parent page.

This can be summarize like this

containerList -> container -> pageField -> page (child)

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.

This is the standard way for building navigation in Jahia templates. One of the nice output 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.

2.5 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
- optional : a 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
- optional: a default theme and other additional themes
- optional: a password policy

The administration of a Jahia virtual site is covered in the Jahia Administration Guide. The important points relating to template development are

- the set of templates that are made available for a Jahia site
- 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.

2.6 Template set

A template set is a collection of templates, and all related resources like jsp files, jspf files, images, scripts… allowing these templates to be recognized, compiled and executed by Jahia when requested.

Template sets are now packaged in war files.

A maven archetype is available to create and start a new project for template development.

Setup procedure for creating a Jahia Template Project using maven:

Create a new folder in your file system to be used as development workspace or use an existing one. This folder could be the workspace folder used by your IDE.
If you are using Eclipse you could do the next step directly in Eclipse instead of the Command line window by using the M2Eclipse plug-in. See JahiaPedia Template Developer Guide section for details on how to do it.

Open a Command line window and change the current directory to your workspace directory. You now need to create a project for your templates. We have created Maven archetypes to automatically create a template project structure for you. You should use the following command to create a new templates project:

mvn archetype:generate -DarchetypeCatalog=http://maven.jahia.org/maven2

You have then to choose an archetype:

jahia-templates-archetype	Use this for a new template project, which is not inheriting from any existing template project.
jahia-web-templates-archetype	Use this if you want to use the provided web-templates and just want to add/modify some templates or look & feel.

You will then be asked to enter the following parameters:

packageDisplayName	Display name of your template package (e.g. My custom Jahia templates)
providerUrl	URL of the organization providing your templates: (e.g. http://www.myorganization.org)
jahiaPackageVersion	Jahia version you are using (e.g. 6.0-EE)
resourceBundleName	Name of the automatically created resource bundle holding all the labels in your templates (e.g.MyCustomTemplates). This term has to conform to Java naming conventions and must not use spaces, but simply the CamelCase convention.
artifactId	Your template project's artifact name (e.g. my-custom-templates) and will also be the name of the project's root folder. Don't use space or underscore, but either write all together or delimit with hyphens.
version	Initial version of your template project (e.g. 0.1)
groupId	Will only be asked if you do not confirm the settings. The group ID must be org.jahia.templates, so that the Jahia's maven deployment plug-in will work correctly.

You will be asked to check and confirm the entries. Then the required Jahia template project structure will be automatically created, unless Maven will encounter some errors or inconsistencies.

The structure looks like this:

project base

```
|-- pom.xml
`-- src
`-- main
`-- java
`-- resources
|  `-- jahiatemplates
|  `-- YourResourceBundle.properties
`-- webapp
    |-- definitions.cnd
    |-- home.jsp
    |-- rules.dsrl
    |-- skins
    |-- theme
    `-- WEB-INF
        |-- templates.xml
        `-- web.xml
```

The pom.xml is already set up with the dependencies to the Jahia taglib and API; it also includes the optional configuration used by the Maven Eclipse plugin to create the classpath and Eclipse project structure.

In the Java folder you may add under your package structure any Java files needed by your custom templates.

In the resources folder there is already a default resource bundle for the labels and other resources in your templates. You can translate these resources to other locales by creating a new properties file and append the language (and country) key to the resource bundle name.

In the webapp folder there is already a definitions.cnd file, where you will create the content definitions (page and container content structure) for your templates (see Jahia Compact Nodetype Definitions Notation). There is also a very simple template for your site home page (home.jsp). Also there is a skeleton file for any rules/events you want to react on (rules.dsrl). The subfolders skins and theme are empty, but they can be filled with new CSS files for your custom skins and themes. In the WEB-INF subfolder there is

a templates.xml, where you will define and describe the different templates of your project (see Defining a set of templates). The web.xml is just a dummy file and is not needed for now.

Your JSP template files can be created directly under the webapp folder or any subfolder structure you create underneath. By default template sets are placed in the following folder

`Webapp/ROOT/templates/`

2.6.1 Template set descriptor

Template files are no longer copied upon site creation, but shared between all the sites using the same set of templates.

When deployed, template sets are now stored in

`/JAHIA_ROOT/templates/templateSetName/`

A set of templates may be applied to several websites.

The templates descriptor file is now based on a XML-Schema. Its purpose is the same as in Jahia 4 and Jahia 5.

Example of templates.xml file. Mandatory lines are in red

```
<?xml version="1.0" encoding="UTF-8"?>
<template-set xmlns:xsi=http://www.w3.org/2001/XMLSchema-
instance xmlns="http://www.jahia.net/jahia/templates"
xsi:schemaLocation="http://www.jahia.net/jahia/templates
http://www.jahia.net/shemas/templates_2_0.xsd">
<package-name> yourTemplateSetName</package-name>
      <root-folder> yourTemplateSetName _templates</root-folder>
      <provider>http://www.jahia.org</provider>
      <thumbnail>images/preview.gif</thumbnail>
      <resource-bundle>resources.yourTemplateSetName </resource-bundle>
      <definitions-file>definitions.cnd</definitions-file>
      <definitions-file>definitions.grp</definitions-file>
      <rules-file>rules.dsrl</rules-file>
      <common-pages>
            <my-settings>mysettings.jsp</my-settings>
```

```
            <search-results>searchresult.jsp</search-results>
      </common-pages>
<templates default="home" homepage="home">
<template name="home" display-name="template.name.home" filename="home.jsp" page-type="
yourTemplateSetName:home"/>
      <template name="news" display-name="template.name.news" filename="news.jsp" page-
type=" yourTemplateSetName:news"/>
      <template name="simple" display-name=" template.name.simple"
filename="simple.jsp" page-type=" yourTemplateSetName:simple"/>
      <template name="double" display-name=" template.name.double"
filename="double.jsp" page-type=" yourTemplateSetName:double"/>
      <template name="faq" display-name="FAQ" filename=" template.name.faq.jsp" page-
type=" yourTemplateSetName:faq"/>
</templates>
</template-set>
```

- package-name: Unique name of the set of templates (identifier)
- root-folder: Root folder of the set of templates
- thumbnail: Relative path to the preview image
- resource-bundle: Default resource bundle to use
- definitions-file: Relative path to the definitions files. Extension .cnd will contain all definitions and extension .grp will declare the order in which the fields should appear in the edition engines.
- rules-file: Relative path to the rules file to use when events are fired.
- common-pages: Defines what JSP files to use for displaying search results and “mySettings”.
- templates: Start defining your templates: the default template to use and the template to use when creating a site’s home page.
- template: defines a template

2.6.2 Template inheritance

One of the new features brought by Jahia EE v6 is the template inheritance mechanism. The template inheritance mechanism implies that a template set can inherit from another template set. By default a template set inherits from Jahia default components. Inheritance offers two major improvements:

- ✓ Ease of maintenance: change the resource once, all templates set will benefit from the change.

- ✓ Customization: if the inherited component doesn't meet the needs, you can override it at a local level.

Usage:

Error pages (400, 401, 403, 503)

By default, error pages sent to visitors (if errors occur) are the one provided by Jahia. You can customize those pages for each template-set, by simply recreating the files at the template-set level. If the error pages are not present in the template-set, Jahia will automatically look back until he found the "default" error pages.

Copy *JAHIA_ROOT/errors/401.jsp*

Into *JAHIA_ROOT/templates/myTemplateSet/errors/401.jsp*

Extensions:

Extensions (see the dedicated section in this guide) like `commentable`, `suscribable`, can also be inherited (default behavior) or customized.

Template inheritances:

If you want to inherit from another template set, you just have to add a directive in the current template.xml file

```
<template-set ...
    <package-name>My new template set</package-name>
    <extends>Parent template set</extends>
```

In Jahia EE v6, you may find an example with the TCK templates, which inherit from the Core Templates.

Once the directive has been added, you can use all definitions and templates located in the parent set. For the jsp pages, the `<template:include/>` tag gives the ability to include an element from the parent set if not present in the local set.

Example: the following instruction

```
<template:include page="common/columnA.jsp"/>
```

Will include in your jsp the local common/columnA.jsp file if it exists, then in the parent template set at the same location.

2.7 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, which 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.

	Since 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 its ancestors is in the “waiting for validation” workflow state.
Multi-language	Content objects may either have different values for each language or share 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 expandable metadata fields attached, which are based on the DublinCore standard (http://en.wikipedia.org/wiki/Dublin_Core), such as the author name, the publisher name or the creation date. Every content modification or validation is then automatically stored and indexed in the Jahia EE v6.
In-context Copy/Paste option	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.
Content Picker (dynamic and static copies)	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).
XML Import/Export feature	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.
Indexation and score boost	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. From Jahia EE v6 onwards you can configure term analyzers on a per field level.

2.8 Metadata

Jahia defines by default a set of metadata fields (Dublin core: http://dublincore.org/) for each page, container list and container.

- Creator
- Last contributor
- Creation date
- Last modification date
- Last Publication date
- Last validator
- Keywords
- Description
- Categories

These are defined in the applicationcontext-metadata.xml file. If you need additional metadata fields **in all your sites** you may extend the configurations there. If you only need to add some metadata for a particular set of templates, then you should declare those fields in the definitions.cnd file of the template set (see Declaring Content Objects section in this document).

To display a metadata value, you can use the `template:metadata` tag. You can use this tag, as any other field tag to declare a bean and manipulate the given object as needed. For instance, in the files box declared in the Web-templates, the creation date is retrieved and formatted using a pattern:

```
<template:metadata contentBean="${fileContainer}" metadataName="created" asDate="true"
var="creationDate"/><fmt:formatDate pattern="dd/MM/yyyy" value="${creationDate}"/>
```

A field of type `category` is directly mapped to the category tree. You can have several different category fields per content object linking to a tree of categories (e.g. classification, category, etc... fields). Template developers are now also capable of defining the entry point in the category tree for each category field. So you are not forced anymore 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.

2.9 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 use 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 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.

3 Declaring content objects

It is no longer possible to use the declareContainerList, declareContainer and declareField tags, neither the related scriptlet code in order to define content objects.

You will need to define a new file (i.e. *definitions.cnd*) which will contain all the definitions related to the set of templates.

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

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

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

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

3.1 Field types

All Jahia fields must have one of the following type :

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

3.2 Field selectors

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

- text - is a simple text field used for entering unformatted text. By default, the field is displayed on one single line.
- richtext - fck editor, mainly used or bigtexts.
- choicelist - a drop down menu with the list of constraints is built. Can take the "sort" option to sort the entries of the drop down menu.
- datetimepicker - date and time selector, can be used with date type only. "Format" option specifies the date format.
- datepicker - date only selector, can be used with date type only. "Format" option specifies the date format of the date.
- category - displays the category picker. Can take the "root" option to specify the root category.
- file - displays the file picker. Can take the "mime" option to specify a mime type, and the "filters" option for name filtering.
- portlet - displays the portlet picker.

- page - Jahia page selector is an engine that allows to create a subpage, link to an existing page or create an external link. This selector is only used for the Jahia page type. The "type" option with "direct", "external", and "internal" values can limit the type of link to create. The "templates" option can define the list of templates the user can use for this field (in case of "direct" or "internal" links).
- color - simple color picker.
- checkbox - checkbox field.

3.3 Container definitions

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

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

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

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

A field declaration consists at least a field type and a name:

```
bigText newsDesc
```

The template developer can then choose an optional selector for the field. The name of the selector is put between parenthesis, and can have a list of option between brackets. An optional default value can then be specified here, with a '=' symbol :

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

The template developer can also restrict the type of pages that can be created using a page Field.

```
page newsPage (page[type="direct",templates="news"])
```

3.4 Page definitions

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

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

Container lists are defined exactly the same way as sub container lists in a container.

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

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

Or, if you want to use resource bundles for the display name (you should, in fact).

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

3.5 Mixin types

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

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

For example, web templates define two mixin types:

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

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

Those two types are reused in all page types of the template-set:

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

3.6 Metadata definitions

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

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

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

For example, the following new metadata type is created:

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

And can be assigned to a specific container definition:

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

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

3.7 Default values

Default values can be specified in the definitions file.

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

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

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

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

3.8 Constraints, validators, choice lists

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

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

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

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

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

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

smallText and sharedSmallText fields can be initialized with value lists;

The following will draw a drop-down list in engines

```
sharedSmallText myField (choicelist) <'val1','val2','val3'
```

The following will draw a multiline input form field:

```
sharedSmallText myField (choicelist) multiple <'val1','val2','val3'
```

The following will draw a drop-down list in engines, ‘option2’ will be selected by default:

```
sharedSmallText dropDown (choicelist) = 'option2' < 'option1', 'option2', 'option3'
```

The following will draw a multiline input form field with option1 and option2 selected by default:

```
sharedSmallText selectBox (choicelist) = option1,option2 multiple < 'option1',
'option2', 'option3'
```

The following will draw a drop-down list sorted (option 2, Pim7, toption4, zoption9):

```
sharedSmallText dropDownSorted (choicelist[sort='alpha']) < ‘Pim7’, 'option2',
'zoption9',’toption4’
```

The following will draw a drop down with all values in the interval:

```
integer longDropdown (choicelist) < '[1,10]', '[21,30]
```

The following will draw a drop-down list with values from the ‘eventsType’ resource bundle file:

```
sharedSmallText eventsType (choicelist) < resourceBundle(resources.eventsType)
```

3.9 Read only properties

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

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

3.10 Indexation configuration

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

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

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

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

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

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

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

Example:

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

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

3.11 Titles and labels

Titles and labels of the fields and containers are taken from a resource bundle file that comes along with the cnd file. Standard jahia types use the « JahiaTypesResources » bundle file. Types defined in template packages use the bundle provided within the template package.

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

```
jnt_mainContentContainer = Main content
```

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

```
jnt_mainContentContainer.mainContentTitle = Title
```

3.12 Jahia embedded content objects

Some content objects already exist in Jahia when installing the application, detailed definitions are stored in

WEB-INF/etc/repository/nodetypes/05-standard-types.cnd

Those objects (containers) may be used directly in your own template set definitions.cnd

For instance, if you want to use the newsContainer object as defined by Jahia, you just have to type:

```
[mytemplates:news] > jnt:page
containerList mynews (jnt:newsContainer)
```

You can even extend a Jahia definition inherited from Jahia for your template set. In that case, you will have to call the news declaration from Jahia then add the required fields you want to add, before using that definition in your templates.

Let's add 3 fields to the Jahia news definition in our site names "mytemplates":

```
[mytemplates:newsContainer] > jnt:newsContainer
smallText          newsAgency
integer            numberOfPhotos
sharedSmallText    country
```

Let's use that extended definition in our template "mynews":

```
[mytemplates:news] > jnt:page,mytemplates:navigation
containerList mynews (mytemplates:newsContainer)
```

Finally, let's say that my news are commentable by users:

```
[mytemplates:news] > jnt:page,mytemplates:navigation
containerList mynews (mytemplates:newsContainer [addMixin="jmix:commentable"])
```

3.13 List of build-in objects

```
[jnt:navLink] > jnt:container
 page navLink primary

[jnt:box] > jnt:container abstract
 smallText boxTitle primary mandatory

[jnt:fileContainer] > jnt:container
 file file primary
 bigText fileDesc
 boolean fileDisplayDetails

[jnt:filesBox] > jnt:box
 containerList fileContainer (jnt:fileContainer)

[jnt:lastNewsContainer] > jnt:container
 integer maxNews = 10 indexed=no

[jnt:lastNewsBox] > jnt:box
 containerList lastNewsContainer (jnt:lastNewsContainer)

[jnt:linkContainer] > jnt:container
 page link (page[type="external,internal"])
 bigText linkDesc

[jnt:linksBox] > jnt:box
 containerList linkContainer (jnt:linkContainer)

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

[jnt:mainContentsBox] > jnt:box
 containerList mainContentContainer (jnt:mainContentContainer)
```

```
[jnt:newsContainer] > jnt:container
 smallText newsTitle primary
 bigText newsDesc
 file newsImage (file[mime='image/*'])
 page newslink (page[type="external,internal"])
 date newsDate = now()

[jnt:newsBox] > jnt:box
 containerList newsContainer (jnt:newsContainer)

[jnt:pageContainer] > jnt:container
 page page (page[type="direct"])

[jnt:pagesBox] > jnt:box
 containerList pageContainer (jnt:pageContainer)

[jnt:savedSearchContainer] > jnt:container
 smallText boxSavedSearchTitle
 sharedSmallText boxSavedSearchQuery indexed=no
 sharedSmallText boxSavedSearchView (choicelist) = resourceKey(list) indexed=no <
resourceKey(list),resourceKey(table)
 integer boxSavedSearchMaxs (choicelist) = 10 indexed=no < 1,5,10,20,50,100,200,500

[jnt:savedSearchBox] > jnt:box
 containerList savedSearchContainer (jnt:savedSearchContainer)

[jnt:textContainer] > jnt:container
 bigText text

[jnt:textsBox] > jnt:box
 containerList textContainer (jnt:textContainer)

[jnt:groupsContainer] > jnt:container
 sharedSmallText groupDisplayLimit = 5 indexed=no
 sharedSmallText groupQuery = '*' indexed=no

[jnt:groupsBox] > jnt:box
 containerList groupsContainer (jnt:groupsContainer)
```

```
[jnt:usersContainer] > jnt:container
 integer userDisplayLimit = 5 indexed=no
 sharedSmallText userQuery = '*' indexed=no

[jnt:usersBox] > jnt:box
 containerList usersContainer (jnt:usersContainer)

[jnt:portletContainer] > jnt:container
 portlet portlet

[jnt:portletsBox] > jnt:box
 containerList portletContainer (jnt:portletContainer)

[jnt:peopleContainer] > jnt:container
 sharedSmallText peopleFirstname
 sharedSmallText peopleLastname sortable
 date peopleBirthdate
 sharedSmallText peopleCivility (choicelist) analyzer='keyword' < 'M.','Mme.','Mlle.'
 sharedSmallText peopleGender (choicelist) analyzer='keyword' < 'male','female'
 smallText peopleTitle
 smallText peopleNationality facetable
 file peoplePicture (file[mime='image/*'])

[jnt:peopleBox] > jnt:box
 containerList peopleContainer (jnt:peopleContainer)

[jnt:locationContainer] > jnt:container
 sharedSmallText locationStreet
 sharedSmallText locationZipCode analyzer='keyword'
 smallText locationTown facetable
 smallText locationCountry facetable

[jnt:locationBox] > jnt:box
 containerList locationContainer (jnt:locationContainer)

[jnt:organizationContainer] > jnt:container
 sharedSmallText organizationName facetable
 sharedSmallText organizationAcronym analyzer='keyword' facetable
 file organizationLogo (file[mime='image/*'])
```

```
 sharedSmallText organizationReference
```

```
[jnt:organizationBox] > jnt:box
 containerList organizationContainer (jnt:organizationContainer)

[jnt:mediaContainer] > jnt:container
 file mediaFile
 sharedSmallText mediaAuthor analyzer='simple'
 date mediaDate
 smallText mediaNote
 sharedSmallText mediaCredit

[jnt:mediaBox] > jnt:box
 containerList mediaContainer (jnt:mediaContainer)

[jnt:eventContainer] > jnt:container
 smallText eventTitle
 date eventDateBegin
 date eventDateEnd
 bigText eventContent
 singleContainer locationContainer (jnt:locationContainer)

[jnt:eventBox] > jnt:box
 containerList eventContainer (jnt:eventContainer)

[jnt:IframeContainer] > jnt:container
 sharedSmallText IframeSource indexed=no
 sharedSmallText IframeName indexed=no
 integer IframeWidth indexed=no
 integer IframeHeight indexed=no
 integer IframeFrameborder = '0' indexed=no
 integer IframeMarginwidth = '0' indexed=no
 integer IframeMarginheight = '0' indexed=no
 sharedSmallText IframeScrolling (choicelist) indexed=no < 'yes','no','auto'
 smallText IframeAlt indexed=no

[jnt:IframeBox] > jnt:box
 containerList IframeContainer (jnt:IframeContainer)
```

```
[jnt:videoContainer] > jnt:container
 sharedSmallText videoName
 file videoSource
 integer videoWidth = '0'
 integer videoHeight = '0'
 integer videoHspace = '0'
 integer videoVspace = '0'
 sharedSmallText videoAutostart (choicelist) analyzer='keyword' fulltextsearchable=no <
'true','false'
 sharedSmallText videoInvokeURLs (choicelist) analyzer='keyword' fulltextsearchable=no
< 'true','false'
 sharedSmallText videoEnablecontextmenu (choicelist) analyzer='keyword'
fulltextsearchable=no < '0','1'
 sharedSmallText videoShowstatusbar (choicelist) analyzer='keyword'
fulltextsearchable=no < '0','1'
 sharedSmallText videoShowcontrols (choicelist) analyzer='keyword'
fulltextsearchable=no < '0','1'
 sharedSmallText videoAutosize (choicelist) analyzer='keyword' fulltextsearchable=no <
'true','false'
 integer videoDisplaysize = '0' analyzer='keyword' fulltextsearchable=no
 sharedSmallText videoLoop (choicelist) analyzer='keyword' fulltextsearchable=no <
'true','false'

[jnt:videoBox] > jnt:box
 containerList videoContainer (jnt:videoContainer)

[jnt:fileContentContainer] > jnt:container
  file fileContentSource

[jnt:fileContentBox] > jnt:box
 containerList fileContentContainer (jnt:fileContentContainer)

[jnt:flashContainer] > jnt:container
 file flashSourceFlashContainer
 sharedSmallText widthFlashContainer analyzer='keyword' fulltextsearchable=no
 sharedSmallText heightFlashContainer analyzer='keyword' fulltextsearchable=no
 sharedSmallText flashPlayerFlashContainer = '9' analyzer='keyword'
fulltextsearchable=no
 sharedSmallText idFlashContainer analyzer='keyword' fulltextsearchable=no
```

```
 sharedSmallText nameFlashContainer
```

```
 sharedSmallText swliveconnectFlashContainer (choicelist) analyzer='keyword'
fulltextsearchable=no < 'true', 'false'
 sharedSmallText playFlashContainer (choicelist) indexed=no < 'true','false'
 sharedSmallText loopFlashContainer (choicelist) indexed=no < 'true','false'
 sharedSmallText menuFlashContainer (choicelist) indexed=no < 'true','false'
 sharedSmallText qualityFlashContainer (choicelist) analyzer='keyword'
fulltextsearchable=no < 'low','high','autolow','autohigh','best'
 sharedSmallText scaleFlashContainer (choicelist) analyzer='keyword'
fulltextsearchable=no < 'default','noborder','exactfit'
 sharedSmallText alignFlashContainer (choicelist) indexed=no < 'l','r','t','b'
 sharedSmallText salignFlashContainer (choicelist) indexed=no <
'l','r','t','b','tl','tr','bl','br'
 sharedSmallText wmodeFlashContainer (choicelist) indexed=no <
'window','opaque','transparent'
 color bgcolorFlashContainer
 sharedSmallText baseFlashContainer indexed=no
 sharedSmallText flashvarsFlashContainer indexed=no

[jnt:flashBox] > jnt:box
 containerList flashContainer (jnt:flashContainer)

[jnt:rssContainer] > jnt:container
 sharedSmallText url
 integer entriesCount

[jnt:rssBox] > jnt:box
 singleContainer rssContainer (jnt:rssContainer) mandatory
```

4 Template mechanisms

4.1 Template

A template is a jsp page used by Jahia to generate an html page upon visitor request. The relation between pages and templates is a n – 1 type, meaning that a template may be used on several pages, but a page can and must have only one template at a time.

Templates are written in .jsp files, stored at the root level of the template set directory.

By convention, as there can be many other jsp files in a template set, we advise you to use a prefix or a suffix in templates files names to recognize them at first time during development or maintenance. In the Web-templates set we have chosen to prefix all our templates with tpl.

```
tpl.templateName.jsp
```

4.1.1 Template structure

When executed, a template must use the following tags that defines its structure:

- template
- templateHead
- templateBody

Thus a minimal template looks like this:

```
<template:template>
   <template:templateHead>
      ...
      </template:templateHead>
   <template:templateBody>
      ...
      </template:templateBody>
</template:template>
```

These tags include all the necessary resources for a valid template.

For advanced templating, you may use strategies such as shown in the web-templates where in the template file itself we only declare areas as variables, and send those variables to execute a file name positioning.jsp.

At the end, template files are just “descriptors” but when Jahia executes those templates there are still the template, templateHead and templateBody tags. The difference is there are not directly placed in the template file itself but in an externalized file.

4.2 Containers, container lists and fields

Reminder: A field must always be enclosed in a container, and a container must be placed in a container list to be displayed in a page. A standalone field, or a container not enclosed in a container list will not work at all.

4.2.1 Displaying fields: basic method

In order to display a field, there are **only four tags to learn**.

```
<template:field>
<template:image>
<template:file>
<template:link>
```

There is no longer a tag per field type as it used to be in Jahia 5, with specific parameters for each tag.

Here is an example of code to display a container “portrait” containing 6 fields:

firstName (sharedSmallText)

lastName (sharedSmallText)

age (int)

email (sharedSmallText)

jobTitle (smallText)

picture

```
<template:containerList name="portraitContainerList" id=" portraitContainerList ">
```

```
<template:container id="portrait">

<div class="portrait">
<template:field name="picture" align=”left” alt=”<template:field name=lastName/>
portrait”/>
<h3><template:field name=firstName/> </h3><h4><template:field name=’jobTitle’></h4>
<ul>
<li>Age: <template:field name=”age” /></li>
<li>Email: <template:field file="email" /></li>
</ul>
</div>

</template:container>

</template:containerList>
```

As you can see, the Jahia EE v6 display code is as simple as possible.

The main differences are:

- No more action menu declarations. They are implicit with the template:container and template:containerList tags.
- No more logic tests within the code (Empty / notEmpty / present), the template:field tag handles the tests internally.
- No more content:containerDiffHighlight declarations.
- No more “bean.define” methods for managing images, links, pages

Fields rendering

By default, each type of field is now be returned by Jahia in the most common usage method.

smallText, sharedSmallText, bigText, integer, float: are returned as raw values

categories: string composed of all selected categories, separated by a coma

color: hexadecimal value

file: filename.ext (the css class uses the extension file type)

image: <img src=”url” alt=”file name”>

page: page name

For pages, files and images, if the default rendering of the field tag does not suit your needs, you can use one of the following ones:

- image
- link
- file

Field tag has lots of interesting attributes:

- beanID
- var
- maxChar
- maxWord
- inlineEditingActivated
- removeHtmlTags
- cssClassName

4.2.2 Displaying fields: advanced method

If the default rendering of a `template:field` tag does not meet your needs, then you may use JSTL expression language to retrieve field value and manipulate it : the web-templates offer dozens of examples.

The principle is always the same: use the tag with a display parameter set to false, then declares a `var` variable, with the name you want. For instance if you want to test a value in order to create a conditional treatment.

Example 1: in the Web-templates set (/common/box/display/filesBoxDisplay.jsp) we retrieve the value of a Boolean field. If the value is true, then we display some details about the file in the container.

```
<template:field name="fileDisplayDetails" display="false" var="fileDisplayDetails"/>
…
<c:if test="${fileDisplayDetails.boolean}">
<span class="docsize"><fmt:formatNumber var="num" pattern="### ### ###.##"
type="number" value="${(file.file.size/1024)}"/>(${num} Ko)   <template:metadata
contentBean="${fileContainer}" metadataName="created" asDate="true"
var="creationDate"/><fmt:formatDate pattern="dd/MM/yyyy"
value="${creationDate}"/></span>
</c:if>
```

Example2: the Web-templates set (/modules/teasers/largeTeasers.jspf), we create a var on the field image, because the default display returned by Jahia is <img src="url"> but we want to insert only the image url in a css instruction.

```
<template:field name="image" var="imageBean" display="false"/>
<div class="box2-topright"></div><div class="box2-topleft"></div>
      <h3 class="box2-header"><span><template:field name='title'/></span></h3>
     <div class="box2-illustration" style="background-
image:url(${imageBean.file.downloadUrl})"></div>
```

Example 3: page field

```
<template:field name="myPage" var="myPageBean" display="false"/>
<c:if test="${! empty myPageBean.object}">
    <a href="${myPageBean.object.url}">${myPageBean.object.title}</a>
</c:if>
```

4.2.3 Pagination and other attributes

With some of the attributes you can control how many list items should be fetched and displayed at all and per page.

Jahia EE v6 now implicitly creates the UI controls to navigate through a list. In Jahia 5 you had to explicitly use some pagination tags to do that.

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

4.2.4 Absolute container lists

An absolute container list defines a page level where the container list is filled; all occurrences of the same container list in other page levels will use the content of this absolute container list.

For instance, in Web-templates, we have denied the logo container list as an absolute one. Once this container List is filled, all pages which use the same definition will display the same logo as the one defined in the home page.

```
<template:absoluteContainerList name="logo" id="logoContainerList" pageLevel="1">
<template:container id="logoContainer">
        <template:field name="logo"/>
    </template:container>
</template:absoluteContainerList>
```

4.2.5 Unique containers

Sometimes it is necessary to have a unique container in a page, and not a list. The most common case is the site logo, which is unique, or a banner, or even and editorial. Whatever the functional case, to meet that requirement, you must specify this uniqueness in the definitions.cnd file, using the singleContainer expression.

First, we define the container logo:

```
[web_templates:logoContainer] > jnt:container
file logo (file[mime='image/*'])
```

Then, in the template(s) using this logo, we specify that this container is unique:

```
[mytemplates:home] > jnt:page
singleContainer logo (my:logoContainer)
```

In the template jsp code, we call the logo, enclosed in a container list (most of the time, the logo is defined in the home page and stored in an absolute container list, for the example we've deleted all references to the absolute property).

```
<template:ContainerList name="logo" id="logoContainerList"
actionMenuNameLabelKey="logo.update">
    <template:container id="logoContainer">
        <a href='<template:composePageURL
pageID="${requestScope.currentSite.homepageID}"/>'><template:image file="logo"/></a>
    </template:container>
</template:absoluteContainerList>
```

Note that, another syntax for:

```
<a href='<template:composePageURL
pageID="${requestScope.currentSite.homepageID}"/>'><template:image file="logo"/></a>
```

Could be:

```
<a href="${currentSite.homePage.url}"><template:image file="logo"/></a>
```

4.2.6 Boxes

Boxes are special content objects that can embed other content objects, whatever their type.

Boxes are very convenient for developers when they want to add some flexibility in templates or, basically, when they don't know in advance what type of content editors may wish to put in their pages. This is frequently the case with side columns, where editors usually want to be able to put… "whatever I want".

To manage boxes, as for any other content object, you need:

1/ Make declarations in the CND file:

There is a built-in Jahia content, named jnt :box

This object contains the list of containers available in boxes.

To create a new type of box, you must first define the content object you want.

In your definition file, create a new content of jnt:container type. For instance:

```
[web_templates:lastNewsContainer] > jnt:container
integer          maxNews = 10 indexed=no
sharedSmallText display (choicelist) = medium indexed=no < large,medium,small
category         filter (category[autoSelectParent=false]) indexed=no
```

Then, add this container to the box list:

```
[web_templates:lastNewsBox] > jnt:box
 singleContainer lastNewsContainer (web_templates:lastNewsContainer)
```

Then, in your page definition you have specify that this specific type of box is allowed in the containerList:

```
containerList   columnA_box (jnt:box
[availableTypes="jnt:IframeBox,web_templates:lastNewsBox,jnt:textsBox,web_templates:filesBox"])
```

2 / Create jsp display files:

Display files for boxes must be stored in the `template_set/common/box/display/` folder, this point is mandatory. Each box must have its own display file.

The file name must respect the following syntax: *BoxName*Display.jsp

Where *BoxName* is the name of the jnt:box type you have created. In our case this will be `lastNewsBox`

This file works exactly the same as any other display files in Jahia, the name syntax is the only particularity.

Finally, if you want to display understandable labels in Jahia engines (webmasters GUI) you might add resource bundles keys and values for boxes names. The syntax is the following:

template-set-name_boxName = value

web-templates_lastNewsBox = Last news retrieving

4.3 Menus

Menus used to be one of the most complex things to deal within Jahia 5, as they needed both recursions and iterations. It has also been a rather frequent performance bottleneck for many integrators who created non-optimized menus with direct API calls.

One of the main improvements about templating in Jahia EE v6 is the presence of a very simple tag dedicated to build menus.

4.3.1 Displaying menus: basic method

A new tag has been added to insert a menu in a template.

```
ui:navigationMenu
```

This tag refers to a Jahia buit-in content object (navLink)

To use this menu tag in a template, as for any other content, you have to declare first in the definitions.cnd file that this object is allowed.

```
[templateSetName:pageTypeName] > jnt:page
containerList navLink (jnt:navLink)
```

Several parameters may be defined in the menu tag

- displayActionMenuBeforeLink (true / false)
- cssClassName (string)
- mockupClass (string)
- kind (topTabs / sideMenu / dropDownMenu)
- requiredTitle (true / false)
- usePageIdForCacheKey (true / false)

From this tag, Jahia will iterate over items (and sub-levels if any) rendering a very classical XHTML structure:

```
<ul>                                for each depth level
<li><a href="…">item</a></li>    for each item in a depth level
</ul>
```

Jahia also add the following classes an identifiers:

```
<ul id="#navigationNx">   where x is the depth level (#navigationN1, #navigationN2…)
<li class="xxx xxx">              first, last, inpath
<a class="selected">item</a>   selected
</li>
</ul>
```

The way you choose to display the menus depends only on the CSS. With the following structures and classes, it is possible to create horizontal and vertical menus, change the fonts and colors at any depth level or depending on a specific condition (first item in a list, last item, selected, inpath, etc.)

Example:

Web-templates main menu simple method:

```
<ui:navigationMenu displayActionMenuBeforeLink="true" cssClassName=""
mockupClass="mockup-hmenu" kind="topTabs" requiredTitle="true"
usePageIdForCacheKey="true"/>
```

Web-templates side menu simple method:

```
<ui:navigationMenu labelKey="pages.add" kind="sideMenu" mockupClass="mockup-vmenu"
display="true" usePageIdForCacheKey="true" requiredTitle="true">
```

If the provided structure (mode) and the classes (first, last, inpatch, selected, navigationNx) does not feet your needs, or if you want to control every aspects of your menu behavior, add javascript or ajax easily, you may still use the ui:navigationMenu tag to retrieve pagesn but define your own rendering.

To do so, you must set the *display* parameter to false and set the "var" attribute, it contains a collection of *NavigationMenuTag.NavMenuBean* type on which we will iterate to build and display the menu.

Let's do the exercise with the side menu of the Web-templates modifying the declaration like this:

```
<ui:navigationMenu labelKey="pages.add" kind="sideMenu" mockupClass="mockup-vmenu"
display="false" var="navMenuBeans" usePageIdForCacheKey="true" requiredTitle="true">
```

NavMenuBean object has the following properties

- title : titre long de la page
- url : url de la page
- actionMenu : action menu (sous forme de) de la page (edition, suppression)
- level : niveau de la page firstInLevel : booleen à oui si la page est la première de son niveau
- lastInLevel : booleen à oui si la page est la dernière de son niveau
- actionMenuList : action menu (sous forme de) de la gestion du niveau actuel de menu (ajout, tri)
- inPath : Boolean set to true if the page is in the tree path to the current page
- selected : Boolean set to true if the page is selected (e.g. the current page)
- markedForDelete : Boolean set to true if the element is waiting for deletion
- displayLink : formatted title

- actionMenuOnly : Boolean set to true if the NavMenuBean contains only the list action menu (which is the case if there is no further element in the list)

```
<ui:navigationMenu labelKey="pages.add" kind="sideMenu" mockupClass="mockup-vmenu"
display="false" var="navMenuBeans" usePageIdForCacheKey="true" requiredTitle="true">
  <%--test if any items in menu (under 2 items because the first is the manage menu
box)--%>
    <c:forEach var="navMenuBean" items="${navMenuBeans}">
        <c:if test="${fn:length(navMenuBeans) == 1 && navMenuBean.actionMenuOnly &&
requestScope.currentRequest.editMode}">
          <div class="mockup-vmenu">
       <c:set var="liCssNavItem" value=""/>
        <c:set var="aCssNavItem" value=""/>
        <c:if test="${navMenuBean.firstInLevel}"><c:set
var="liCssNavItem" value="${liCssNavItem} first"/></c:if>
        <c:if test="${navMenuBean.lastInLevel}"><c:set
var="liCssNavItem" value="${liCssNavItem} last"/></c:if>
        <c:if test="${navMenuBean.inPath}"><c:set var="aCssNavItem"
value="${aCssNavItem} inpath"/></c:if>
        <c:if test="${navMenuBean.selected}"><c:set var="aCssNavItem"
value="${aCssNavItem} selected"/></c:if>
        <c:if test="${navMenuBean.markedForDelete}"><c:set
var="aCssNavItem" value="${aCssNavItem} markedForDelete"/></c:if>
        <li class="${liCssNavItem}">
        <c:if test="${!navMenuBean.actionMenuOnly}">
            <div class="level_${navMenuBean.level} ${liCssNavItem}"><a
class="${aCssNavItem}" href="${navMenuBean.url}"
alt="${navMenuBean.title}"><span>${navMenuBean.displayLink}</span></a>${navMenuBean.act
ionMenu}</div>
        </c:if>
        <c:if test="${requestScope.currentRequest.editMode &&
navMenuBean.lastInLevel}">
                <div>${navMenuBean.actionMenuList}</div>
         </c:if>
      <c:if test="${fn:length(navMenuBeans) == 1 && navMenuBean.actionMenuOnly &&
requestScope.currentRequest.editMode}">
            </div>
        </c:if>
```

```
    </c:forEach>
    </ui:navigationMenu>
```

4.3.2 Advanced Menus: step by step explanation

```
<ui:navigationMenu labelKey="pages.add" kind="sideMenu" mockupClass="mockup-vmenu"
display="false" var="navMenuBeans" usePageIdForCacheKey="true" requiredTitle="true"> On
défini une collection  dont le nom est : navMenuBeans
<c:forEach var="navMenuBean" items="${navMenuBeans}"> On itere sur les elements de la
collection que l'on place dans une variable navMenuBean
<c:if test="${fn:length(navMenuBeans) == 1 && navMenuBean.actionMenuOnly &&
requestScope.currentRequest.editMode}">
<div class="mockup-vmenu">
</c:if>
```

This part tests in edit mode, if the collection has more than one item and this item is an action menu only. If true, this is an empty menu, we display the mockup of the menu. (This mockup is set with a and be closed at the end of this menu)

```
       <c:set var="liCssNavItem" value=""/>
        <c:set var="aCssNavItem" value=""/>
        <c:if test="${navMenuBean.firstInLevel}"><c:set
var="liCssNavItem" value="${liCssNavItem} first"/></c:if>
        <c:if test="${navMenuBean.lastInLevel}"><c:set
var="liCssNavItem" value="${liCssNavItem} last"/></c:if>
        <c:if test="${navMenuBean.inPath}"><c:set var="aCssNavItem"
value="${aCssNavItem} inpath"/></c:if>
        <c:if test="${navMenuBean.selected}"><c:set var="aCssNavItem"
value="${aCssNavItem} selected"/></c:if>
        <c:if test="${navMenuBean.markedForDelete}"><c:set
var="aCssNavItem" value="${aCssNavItem} markedForDelete"/></c:if>
```

Two css variables are defined liCssNavItem and aCssNavItem. These variables will contains css names to be display in or <a> tag.

```
        <c:if test="${!navMenuBean.actionMenuOnly}">
```

```
            <div class="level_${navMenuBean.level} ${liCssNavItem}"><a
class="${aCssNavItem}" href="${navMenuBean.url}"
alt="${navMenuBean.title}"><span>${navMenuBean.displayLink}</span></a>${navMenuBean.act
ionMenu}</div>
        </c:if>
```

If a link is defined (not actionMenuOnly), we display the menu:.

```
<c:if test="${requestScope.currentRequest.editMode && navMenuBean.lastInLevel}">
<div class="level_${navMenuBean.level}">${navMenuBean.actionMenuList}</div>
</c:if>
```

On the last item, we display the action menu to manage the list:

```
<c:if test="${fn:length(navMenuBeans) == 1 && navMenuBean.actionMenuOnly &&
requestScope.currentRequest.editMode}">
</div>
</c:if>
```

end of mockup tag.

```
</c:forEach>
```

Close forEach tag:

```
</ui:navigationMenu>
```

Close navigationMenu tag

5 Advanced templating

5.1 Resource bundles

Resource bundle in Jahia EE v6 has been rewritten to be JSP 2.0 and JSTL compliant and to ease integrators' work. More than changes, some new capabilities have been added to use smartly the resource bundle with Jahia fields.

5.1.1 Main changes

In Jahia 5, resource bundle files were all located in

/webapps/Jahia/WEB-INF/classes/jahiatemplates

and not in the template set folder, which used to cause a real lack of productivity.

We introduced a new `utility:setBundle` tag which is fully compliant with the `fmt:setBundle` defined by the standard. The purpose of this tag is to allow the localization of your templates resource bundle files inside your templates directory, summarizing it your resource bundles files could now be moved from

/webapps/jahia/WEB-INF/classes/jahiatemplates/

to

/webapps/jahia/templates/my_template_set/

In the templates.xml file you may now specify which resource bundle is associated with your template set. This resource bundle will be available by default in every request, so you won't need to redefine a bundle using `fmt:setBundle` for example (as long as you want to use your template resource bundle).

The template resource bundle benefits from the template inheritance mechanism. It means that if your template inherits from another template, there is no need to override or redefine its resource bundle (unless you really want to) through a `setBundle` tag. Resource bundle defined in the inherited template will be used locally in a transparent way.

If, for some particular reasons, you need to use another resource bundle than the one defined in the template.xml file or through inheritance mechanism, then you can define it with a `setBundle` tag.

5.1.2 New capabilities

You can now also use a full resource bundle file to initialize a choice list for your fields instead of defining all possible values on a key per key basis with resourceKey keyword inside the cnd file.

You will find an example of such usages inside the events templates where we want to use some resource bundle file to initialize a choice list of values.

Example of initializing a choice list from a resource bundle file:

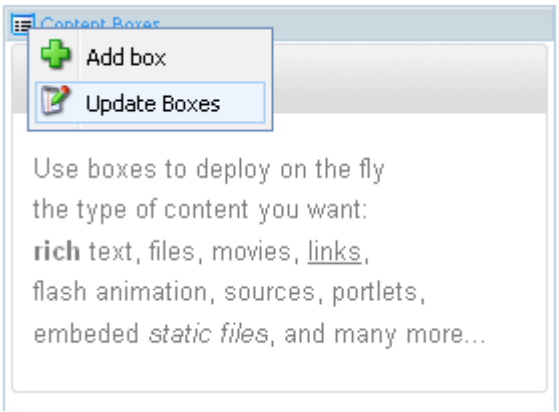
```
[web_templates:eventContainer] > jnt:container, web_templates:title
date             startDate
date             endDate
smallText        location facetable sortable
sharedSmallText eventsType (choicelist) facetable sortable <
resourceBundle(resources.eventsType)
bigText          body
```

5.2 Action menus

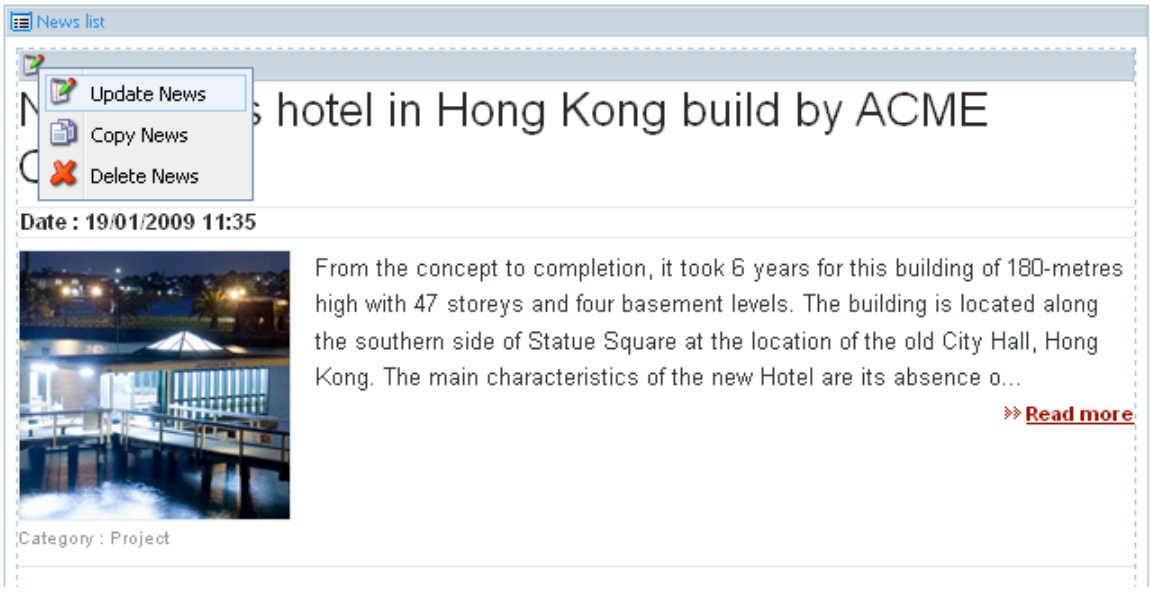
Action menus are the GUI components enabling Editors to interact with content structures available in pages (add, update, delete, sort…).

Since Jahia EE v6, action menus are positioned automatically by Jahia when using the containerList and container tags, therefore integrators do not have to manage them anymore.

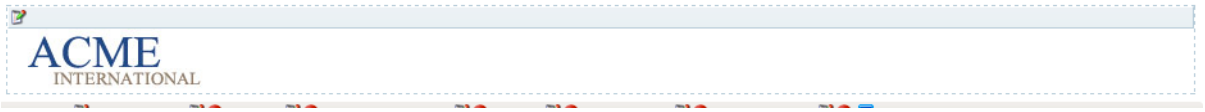
When a containerList is empty, there is only one Action Menu available on the list



Once content has been added, Jahia draw the action menus, on each container



When a containerList is specified as simple, the containerList action menu is not displayed and the action menu of the container is only displayed (to avoid redundant actions and because sorting properties are not relevant when there can be only one item).



In some cases, developers may want to position themselves the action menus. If you want to do so, first you have to disable in `containerList` tag and/or `containerTag` the display of the action menu by setting displayActionMenu to false

```
<template:container id="bannerContainer" displayActionMenu="false">
```

Then to position you action menu you will use the `<ui:actionMenu>` tag. The mandatory attribute contentObjectName is the object ID (ID is set on the containerList or container you want to display)

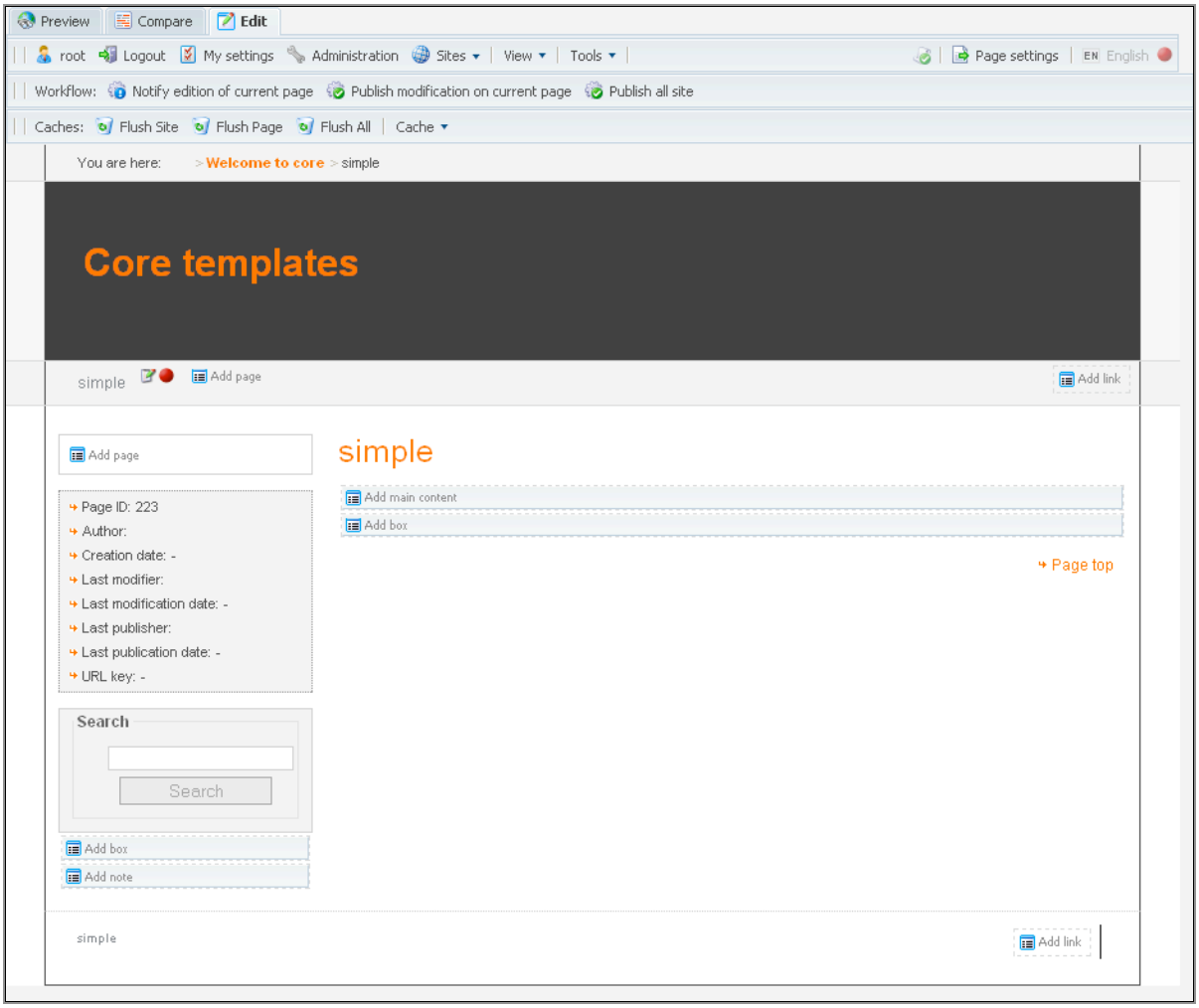
```
<ui:actionMenu contentObjectName="bannerContainer"/>
```

You do not have to specify if your action menu is set for a container or a container list, Jahia will automatically draw the right component using the `contentObjectName` to retrieve the type of object.

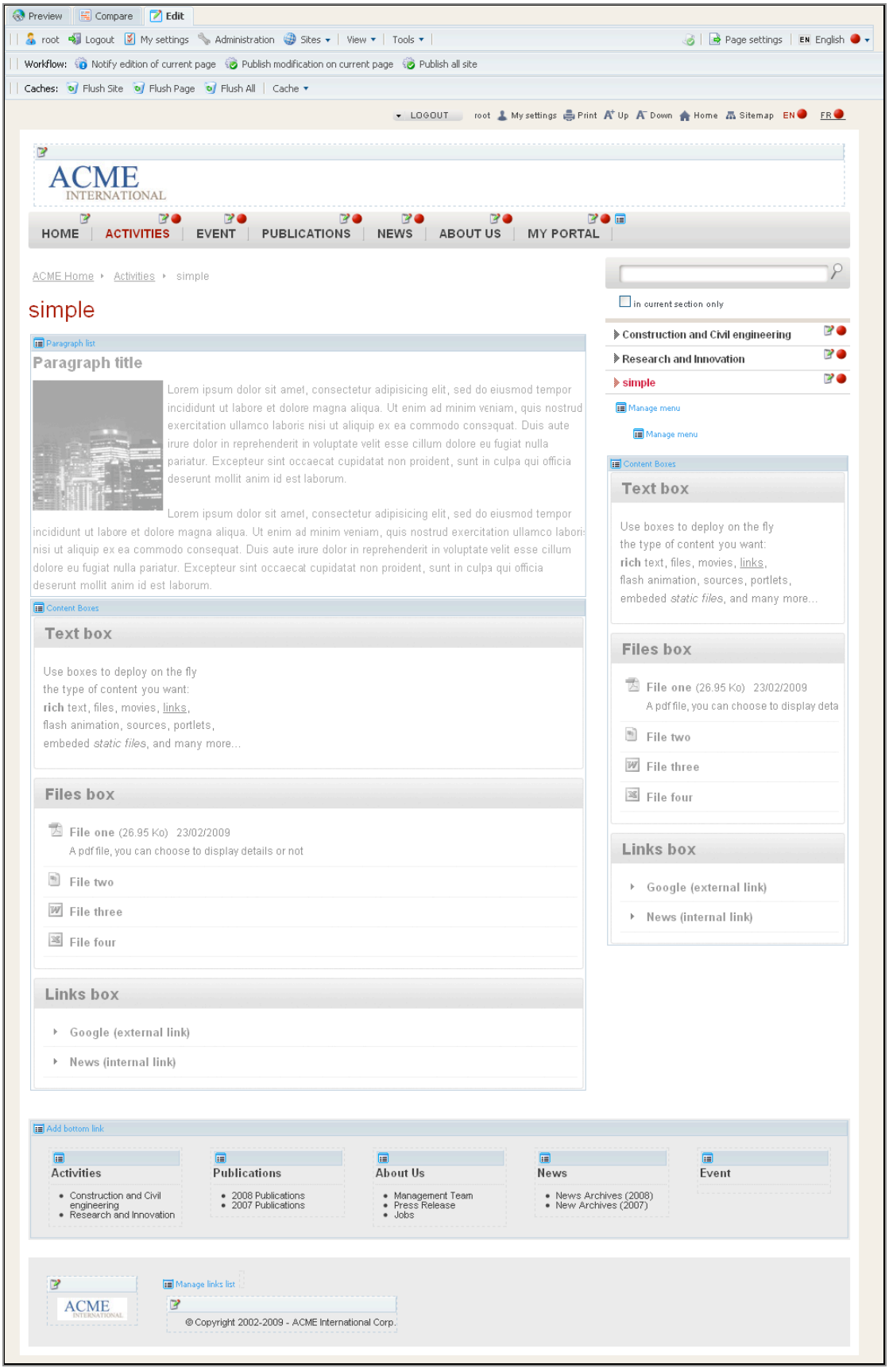
5.3 Mockups

We call mockups the preview images displayed in edit mode to ease editor's work. Thanks to those preview images, editors can see how the page will look, if content areas are filled, and can understand immediately what type of content may appear in an area, which is not so self-speaking with action menus alone.

An empty page without mockups:



A page with mockups:



Mockups are optional,

Mockups are declared at the container level using a simple parameter

```
emptyContainerDivCssClassName
```

If a container list is empty, then in edit mode Jahia will use this parameter to search the css class and add this class to the defining the area.

For instance in *webapp/Jahia/templates/web-templates/modules/introductionDisplay.jsp*

```
<div class="intro">
<template:containerList name="introduction" id="introductionContainerList"
actionMenuNamePostFix="introduction"actionMenuNameLabelKey="introduction">
        <template:container id="mainContent"
emptyContainerDivCssClassName="mockup-introduction">
            <template:field name='introduction' />
        </template:container>
    </template:containerList>
</div>
```

The class mockup-introduction will be applied in edit mode if the container list is empty.

The class source is stored in edit.css stylesheet (the sheet which is loaded only in edit mode)

```
.mockup-introduction {
display:block;
width:638px;
height:226px;
background: url( ../img/mockup/bk_introduction.gif) top left no-repeat;
}
```

Note: in the edit.css style sheet, all mockups classes defined by Jahia start by '.mockup- ', hence it is very easy to remove them if you do not want them to be loaded.

5.4 Skins

A skin is a piece of code surrounding a **container list**, which can be chosen and applied by editors, from a limited list of choices provided by integrators.

The first usage of skins is to give a more granular control on the final look and feel of web-pages to content editors, but in a limited and well mastered manner. For instance the Web-templates use the skin mechanism to propose different layouts like those ones on boxes.

How-to implement skins:

- Create a skins directory in your template set
- Create one folder per skin. Each folder must have a name without spaces, accents or accentuated characters.
- Create one jsp file per skin with your desired html surrounding the instruction <template:includeContent/>
- In the bundle resource files add your skins folder names as keys, and fill with more human-speaking values. (I.e. skins2 = round corners with icon, skin3= square corners without icon…)
- Create a preview image of the final rendering in png format, name it exactly as your skin folder.
- In the definitions.cnd file, add the property to the containerList declaration
    - addMixin="jmix:skinnable"

5.5 Themes

The theme mechanism is new in Jahia EE v6. Switching from a theme to another one allows changing the complete layout of a page or the whole site. The theme property is checked when a page is requested and, if the property is empty, a recursive mechanism will check for parent pages until a theme is found. It means that if you change a page theme at any point of the tree, all sub-pages where no theme has been specified will be impacted.

The most obvious usage is to change the look and feel for a limited period of time (feast, sales, special event, announce or partnership) in a well mastered process. Using themes, it will be possible to prepare the second layout without any perturbations or code changes, and then switch from one layout to another, the roll-back to the first one when necessary, without any risks.

In fact, a theme is nothing more than a collection of graphic resources (css and images), which are loaded by Jahia and used to generate final HTML pages depending on a variable which is the current theme selected.

The theme is applied to the complete website, and cannot be set for a particular section or depending on the page level for instance.

Usage of themes is absolutely not mandatory, in the provided template sets, both TCK and Core templates don't use it. Therefore the template set directory will look very familiar to Jahia 5 developers.

To use themes, there are no declarations to do anywhere, but a more structured construction of your template directory is required.

- In the *template_set_root_directory/*, create a *theme* directory
- Inside the *theme* directory, create at least on folder named default, then as much folders as themes you want to deploy; for instance red_theme, christmas_theme, (in fact the naming must respect the resource bundle key naming standard restrictions).
- Inside your template resource bundle, add the theme name as a key and the value that will be displayed as title in Jahia engines. (for instance christmas_theme = Theme for Christmas sales)
- Inside the folder of your theme, create a css folder and put all your style-sheets in this folder.
- Other resources like images may be stored where you want in the folder of your theme, if your links in the css style-sheet are correct
- In the *template_set_root_directory/images/* you can put a preview image for each theme. The naming is strictly defined: preview_[name-of-the-theme].gif ('preview_ christmas_theme.gif' is the theme is named 'christmas_theme' for instance).

Once those operations are done, Jahia will automatically propose in the administration interface to select a theme between all available themes.

Jahia also loads automatically all the css style sheets stored in /theme/theme_name/css/ directories, so you do not have to declare what style-sheets have to be loaded.

Until a theme is changed by a site administrator, the 'default' theme is used by Jahia.

When installing Jahia, theme management is by default accessible to site administrators only (Administration > Site management > Default theme).

To change the theme and a page level, use the Page properties engine.

The theme selected by the administrator will be applied by default for all users. As the theme selection is just a variable, changing the theme has not negative impacts on caches and on performances.

Final users may have the right to change the theme also, if you agree so. Using the `<ui:themeSelector scope="user"/>` tag in your template will draw a drop-down list filled with the complete list of themes.

If you want editors to be able to change the theme, without the site administration rights, you may use the `<ui:themeSelector scope="user"/>` tag and set it visible only in edit mode by security.

When a user changes the selected theme, this modification is stored only for him, and will not impact the whole site default theme (set by the site administrators) or the other users changes.

If the user is logged, the selected theme is stored as a property (and therefore maintain from one session to another). If the user is not logged (guest account) then the theme selected will be used in the current session (then lost after when session ends).

For sure themes are not used in all web sites, nevertheless, even if you have a single design, we advise you to store your resources as if themes may be used one day.

5.6 Querying content (Query Taglib)

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

This tag is used to declare an abstract Query Object Model:

```
definition
```

This tag is used to build a Concrete ContainerQueryBean from a query model

```
containerQuery
createContainerQueryBean
```

The Selector tag is used to define the Node Type Name of Nodes that will appear in the query result

```
selector
```

Those tags are used to create query constraints within an abstract or concrete query definition:

```
comparison
childNode
descendantNode
equalTo
greaterThan
greaterThanOrEqualTo
lessThan
lessThanOrEqualTo
like
notEqualTo
fullTextSearch
```

Those tags are used to create logical connections on query constraints within an abstract or concrete query definition:

```
and
or
not
```

This tag is used to declare ordering of query results:

```
sortBy
```

Example of a query retrieving all news of the current site:

```
<query:containerQuery>
      <query:selector nodeTypeName="jnt:newsContainer" selectorName="newsList"/>
      <query:descendantNode selectorName="newsList" path="${currentSite.JCRPath}"/>
      <query:sortBy propertyName="newsDate"
order="${queryConstants.ORDER_DESCENDING}"/>
</query:containerQuery>
```

After building the query, the integrator just have to define the treatment he wants to operate on the result set, usually displaying data.

You can find an example of a box using a query to retrieve news in the Web-templates. In this example, three options have been added to give more control to the editor about the final result:

- ✓ a filter upon categories (retrieving only news sharing the same categories)
- ✓ an integer to define the maximum number of news to display
- ✓ the display format. Three formats have been defined. The rendering code for each format is located in a specific .jspf file. Choosing one or the other rendering, the editor in fact indicates Jahia which code segment should be used to display the result set after the query.

```
<template:containerList maxSize="${maxNews.integer}" id="newsList"
displayActionMenu="false">
    <query:containerQuery>
        <query:selector nodeTypeName="web_templates:newsContainer"
selectorName="newsList"/>
        <c:if test="${!empty categoriesFilter}">
            <query:equalTo propertyName="${queryConstants.CATEGORY_LINKS}"
value="${categoriesFilter}" multiValue="true" metadata="true"/>
        </c:if>
        <query:descendantNode selectorName="newsList" path="${currentSite.JCRPath}"/>
        <query:setProperty name="${queryConstants.SEARCH_MAX_HITS}"
value="${maxNews.integer}" />
        <query:sortBy propertyName="newsDate"
order="${queryConstants.ORDER_DESCENDING}"/>
    </query:containerQuery>
    <c:if test="${display == 'small'}">
<%@ include file="../../../modules/news/smallNewsDisplay.jspf" %>
    </c:if>
    <c:if test="${display == 'medium'}">
        <%@ include file="../../../modules/news/mediumNewsDisplay.jspf" %>
    </c:if>
    <c:if test="${display == 'large'}">
        <%@ include file="../../../modules/news/largeNewsDisplay.jspf" %>
    </c:if>
</template:containerList>
```

If the editor chooses the “medium” rendering, then the following code will be executed after the query.

```
<ul class="summary">
    <template:container id="newsContainer" cacheKey="shortNews"
displayActionMenu="false" displayExtensions="false" displayContainerAnchor="false">
        <li class="summary">
            <template:field name="newsImage" var="newsImage" display="false"/>
            <div class="summaryImg"><img src="${newsImage.file.downloadUrl}"
alt='<template:field name="newsTitle" inlineEditingActivated="false"/>'/></div>
            <h4><a
href="${currentPage.url}/template/tpl.newsDetail?queryPath=${newsContainer.JCRPath}"><t
emplate:field
                    name='newsTitle' inlineEditingActivated="false"/></a></h4>
```

```
            <p class="summaryresume"><template:field name="newsDesc" maxChar="120"
inlineEditingActivated="false"/></p>
            <div class="clear"> </div>
        </li>
    </template:container>
</ul>
```

5.7 Search (Taglib)

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

Tags to create a form for entering search input to create simple or advanced search forms on Jahia pages or the file repository:

- form
- created
- createdBy
- date
- documentProperty
- documentType
- fileLocation
- fileType
- language
- lastModified
- lastModifiedBy
- pagePath
- rawQuery
- term
- itemsPerPage

Tag to define the URL of the results page:

- resultsPageUrl

Tags to execute the search defined in the form and to iterate over the results:

- results
- resultIterator

Used to display the results in a table (e.g. for saved search):

- resultTableSettings
- resultTable

Exposes a descriptor for the specified document property into the page scope:

- documentPropertyDescriptor
- Create html structure for an open search:
- opensearchBox

5.8 Names Jahia variables for JSP Expression language

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

Attribute name	Class	Description
`currentPage`	`org.jahia.data.beans.PageBean`	Provides access to an object that contains information relative to the current page, as well as all the objects in the page
`currentSite`	`org.jahia.data.beans.SiteBean`	Provides access to an object that contains information relative to the current site, as well as all the pages in the site.
`currentJahia`	`org.jahia.data.beans.JahiaBean`	Provides access to an object that contains

		information relative to the current Jahia installation, as well as all the sites
`currentRequest`	`org.jahia.data.beans.RequestBean`	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 `isEditMode()`, `isLogged()`, etc..
`currentUser`	`org.jahia.services.usermanager.JahiaUser`	This object is the user currently logged in JahiaUser (would be guest if the user is not logged in), and provides access to user properties
`dateBean`	`org.jahia.services.expressions.DateBean`	Contains some formatted dates based on currenttdate

Here are some expressions you may find useful:

`${currentRequest.normalMode}`
`${currentRequest.compareMode}`
`${currentRequest.previewMode}`
`${currentRequest.editMode}`
	Returns `true`/`false` depending on the mode of the current request

`${currentRequest.IE}`
`${currentRequest.IE4}`
`${currentRequest.IE5}`
`${currentRequest.IE6}`
	Returns `true`/`false` depending on the user agent of the current request (Internet Explorer, Mozilla)

`${currentRequest.IE7}`
`${currentRequest.NS}`
`${currentRequest.NS4}`
`${currentRequest.NS6}`	

`${currentRequest.mac}`
`${currentRequest.unix}`
`${currentRequest.windows}`
	Returns `true`/`false` depending on the user agent's operating system

`${currentRequest.logged}`	Returns `true`/`false` whether user for current request is logged on or not
`${currentRequest.admin}` `${currentRequest.root}`	Returns `true`/`false` if user for current request is administrator or even root user
`${currentRequest.adminAccess}` `${currentRequest.writeAccess}`	Returns `true`/`false` if user for current request has admin or write access
`${currentUser.name}`	Returns name of the current user
`${currentUser.userProperties['somePropName']}`	Returns a property of current user
`${currentSite.homepageID}`	Returns homepage ID of current site
`${currentSite.homePage}`	Returns `PageBean` of homepage of current site
`${currentSite.id}`	Returns ID of current site
`${currentSite.templatePackageName}`	Returns template set name of current site
`${currentSite.siteName}`	Returns current site name

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

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

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

5.8.1 Constants

For some taglibs you will need to pass constants as attributes, which are defined in some Jahia classes. If you just hardcode the constant in your template, it may be for instance just a meaningless magic number. So it would be better to directly use the defined constant variable name. You can use the `<utility:useConstants>` tag to be able to access such static variables from JSTL.

For instance with this definition:

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

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

you could then access the static variables like this:

```
${queryConstants.CATEGORY_LINKS}
or
${fieldTypeConstants.SMALLTEXT}
```

5.9 Extensions

Jahia offers some extensions to its definitions. (rssable, commentable,.. etc.)

The display of these extensions can be found in the directory:

`JAHIA_ROOT/templates/default/extensions/`

5.9.1 Setting:

In your definitions, you have to add the extension to the container definition

example:

```
      containerList news (web_templates:newsContainer [addMixin="jmix:commentable"])
```

5.9.2 Display:

If you want to custom the displaying of the extension, create an extensions directory in your template root folder, copy and custom the file of the extension you need. (in our example, extensions/commentable/commentable.jsp)

All the resources needed for these extensions are created by Jahia. You can create very simply a commentable container with Jahia definitions (title, body, name, author), but if you need to customize this part, you will have to use the form handler for that.

Actually, you can use:

commentable : offer the ability to add a form under a containerList to let users add comments.

rssable : add an rss icon on the top of the containerlist with an URL can be set as rss file.

subscribable : add a form to subscribe on modifications in the current containerList.

For more informations about extensions, please refer to JahiaPedia.

5.10 Form Handler

You can associate a container to a form in order to fill a containerList.

First you need to set the definition (the example is from the job template in web templates):

```
[web_templates:answerJobContainer] > jnt:container
sharedSmallText firstname
sharedSmallText lastname primary sortable
sharedSmallText email
bigText         text
```

Next you will display the form in your template:

```
<template:gwtJahiaModule isTemplate="true" jahiaType="form" id="form"
nodeType="web_templates:answerJobContainer"
captcha="${pageContext.request.contextPath}/jcaptcha" action="createNode"
target="${jobContainer.JCRPath}/jobAnswers" cssClassName="answer"/>
```

This is a GWT module, you have to set the attribute gwtForGuest at true in <template:template> tag

You have also to set the display of this container if you want to see its content:

```
<template:containerList name="jobAnswers" id="jobAnswers" displayActionMenu="false">
                    <table width="100%" style="border:1px solid #DDDDDD">
                    <template:container id="jobAnswer" displayActionMenu="false"
displayContainerAnchor="false">
                        <tr>
                            <td><template:field name="firstname"/></td>
                            <td><template:field name="lastname"/></td>
                            <td><template:field name="email"/></td>
                            <td><template:field name="text"/></td>
                            <td><ui:actionMenu contentObjectName="jobAnswers"
labelKey="update"></ui:actionMenu></td>
                        </tr>
                    </template:container>
                    </table>
</template:containerList>
```

You have to set this container as “write for guest” if you want guest to post in this form.

5.11 Using Ajax

Jahia has chosen Gwt/Gxt framework to build all Ajax GUI components used in edit mode, in engines and in administration panels. In Jahia EE v6, both Zimbra and Prototype have been removed in order to have a more coherent and rational approach.

Some ready to use components built with Gwt/Gxt can be reused in templates by integrators. Nevertheless, we know that many integrators are not familiar with Gwt/Gxt framework and may prefer a lighter framework in their templates.

You are free to use your preferred framework within templates, but we strongly recommend JQuery 1.3 (http://jquery.com/) a very complete framework with hundreds of scripts available over the internet. We

have tested it and most scripts work “as is” with Jahia without any problem. Sometimes light adaptations are required, but it still seems to us the number one choice.

In the Web-templates, we have chosen to integrate JQuery as an example, and use it twice:

- ✓ In the events template, to display a calendar,
- ✓ In the managers template, to display the full size picture.

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

www.jahia.com
The Company website

www.jahia.org
The Community website