Container List Search, Filter, Sort & Pagination.

Search, Filter, Sort and Pagination on large container list data

Khue Nguyen

Jahia Ltd

version 1.0

Jahia Ltd
45, rue de la Gare
1260 Nyon
Switzerland

DISCLAIMER

Jahia Ltd
45, rue de la Gare
1260 Nyon
Switzerland

support@jahia.org

Document updates

Document Edition	Date	Changes

LICENSE AGREEMENT

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

JAHIA LICENSE Version 2.1 Copyright 2001 by XO3, SA. (http:// www.xo3.com). All rights reserved.

1. *LICENSE TO USE.*
   This is protected software (Jahia). The Jahia software is furnished under license and may only be used or copied in accordance with the terms of such license. Redistribution and use in source or binary forms of Jahia with or without modification, are not permitted without prior written authorization of XO3, SA.
   This statement also applies for all web applications included in the software package : redistribution and use in source or binary forms of web applications included in the software package, with or without modification, are not permitted without prior written authorization of XO3, SA.

2. *NAMES.*
   The names XO3, XO3 S.A., MyComponents, Jahia and any of it's possible derivatives may not be used to endorse or promote products derived from this software without prior written permission of XO3, SA. To obtain written permission to use these names and/or derivatives, please contact support@xo3.com.

3. *DECLARATIONS AND NOTICES.*
   Windows and Windows NT are registered trademarks of the Microsoft Corporation. For more information on these products and/or licenses, please refer to their websites (http://www.msn.com - http://www.microsoft.com).
   Java is a trademark of Sun Microsystems, Inc. For more information on these products and/or licenses, please refer to their website (http:// www.sun.com).
   The Hypersonic SQL Software is an Open Source Java Database with standard SQL syntax and a JDBC interface. For more information on these products and/or licenses, please refer to their websites http:// www.hsql.oron.ch).
   Other trademarks and registered trademarks are the property of their respective owners.

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

5. *LIMITATION OF LIABILITY.*
   THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS UP TO YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
   IN NO EVENT SHALL XO3 S.A. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR

PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6. *GOVERNING LAW.*
   Any action derived from or related to this agreement will be governed by the Swiss Law, specifically by that of the Canton of Geneva and shall be of the competence of the judicial authorities of the Canton of Geneva, Switzerland.
7. *TERMINATION.*
   This agreement is effective until terminated.
8. *LAST VERSION.*
   This is the last version of the Jahia license agreement, which replaces all licenses previously issued.

END OF TERMS AND CONDITIONS

TABLE OF CONTENTS

INTRODUCTION

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

As we are focusing on Large Data Container Lists, a chapter is devoted to explain the new Container List Pagination mechanism too.

You should be used with Jahia Templates design, especially with Jahia Container Model. This document does not explain how to declare Field, Container list nor how to use Taglibs.

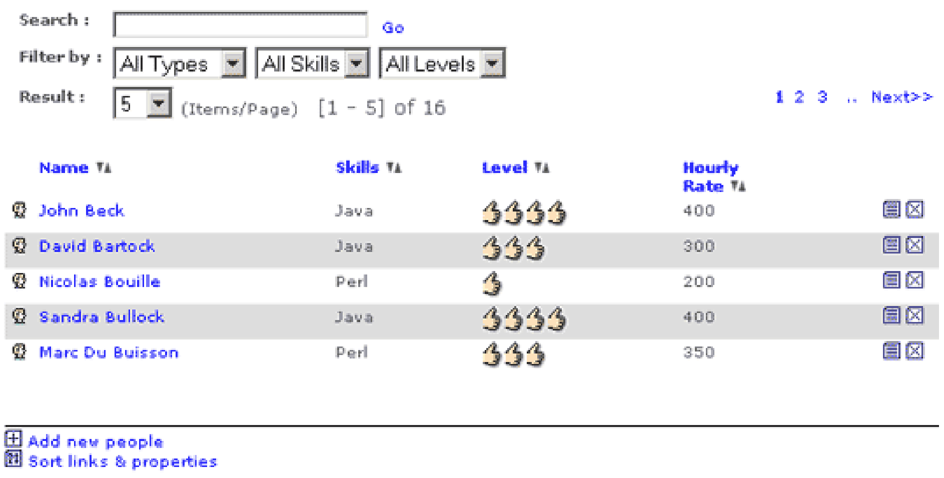
These features are only available with Jahia 3.1 final or higher.

CHAPTER 1 SEARCH, FILTER & SORT

Search, Filter and Sort on large data Container List.

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

Figure 1-1 : Search, Filter and Sort Options.


Search : Go

Filter by : All Types | All Skills | All Levels

Result : 5 (Items/Page) [1 - 5] of 16 1 2 3 .. Next>>

Name	Skills	Level	Hourly Rate
John Beck	Java	👍👍👍👍	400
David Bartock	Java	👍👍👍	300
Nicolas Bouille	Perl	👍	200
Sandra Bullock	Java	👍👍👍👍	400
Marc Du Buisson	Perl	👍👍👍	350

Add new people

Sort links & properties

1.1. JAHIA PAGE FORM

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

More important is that these form values need to be send to the currently displayed Jahia page. That is why the form action is an URL requesting the Jahia current page.

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

1.1.1 Jahia page Form Tag

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

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

Table 1-1 : jahiaPageForm Tag Attributes

Attributes	Description	Mandatory
name	The form name. If not set, the default name is "**jahiapageform**".	no
method	Should be "Post" or "Get". The default value is "**Post**".	no

1.2. Search Option

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

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

1.2.1 Search Engine

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

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

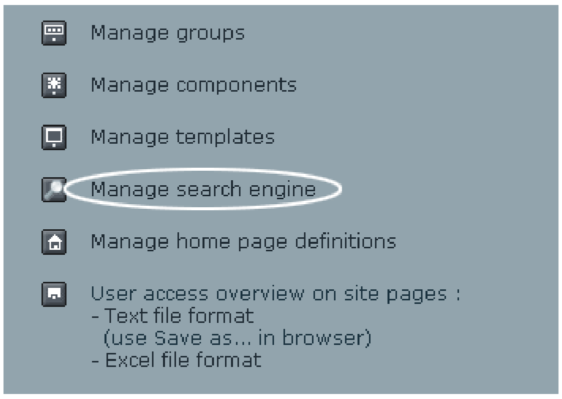


Figure 1-2 : Manage Search Engine.

1.2.2 Linking A Search Option to a Container List.

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

"clistsquery_" + container_list_name

In example, for the container list named "directoryPeopleContainer":

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

1.2.3 Search Option Tags

There are two Tags that can help you adding the search option within the template:

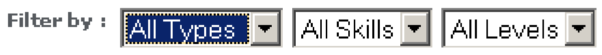
```
<jahia:containerList name="directoryPeopleContainer">
        <input type="text" name="<jahia:ctnListSQueryInputName/>"
                              value="<jahia:ctnListSQueryInputValue/>" />
</jahia:containerList>
```

The **ctnListSQueryinputName** Tag automatically generates the correct Search option input name accordingly to the enclosing container list, while the **ctnListSQueryInputValue** Tag is used to populate the input value with the previous Search query entered by the user.

As you can see, these Tags must be enclosed within the Container List Tag **containerList**.

1.3. Filters Options

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



There are three filters in the figure above:

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

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

But filters can be more complex, i.e :

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

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

For example, the combination of these values,

Filter by : Employee Java Senior

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

1.3.1 Container Filter Bean and Container Filter Handler.

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

Note: **The code below must be placed before the Container List declaration!**

A : Init the vector of filter beans.

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

```

B : Retrieve user Skill choice. Here, the parameter name refers to the select box named **`directoryPeopleSkill_filter`** :

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

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

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

```
17 :
18 : if ( containerFilter != null ){
19 :     // Add the Filter Bean to the vector
20 :     cFilterBeans.add(containerFilter);
21 : }
22 :
23 : // reset the Container Filter Bean.
24 : containerFilter = null;
25 :
```

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

```
26 : // Now we create the Filters Handler, only if there is at least one
27 : // available Filter Bean.
28 : if ( cFilterBeans.size()>0 )
29 : {
30 :     ContainerFilters containerFilters =
31 :     new ContainerFilters( "directoryPeopleContainer",
32 :                           jParams,cFilterBeans);
33 :     // Store the list of filters in the request object.
34 :     // It will be used later by the container list loader.
35 :     request.setAttribute( "directoryPeopleContainer_filter_handler",
36 :                           containerFilters);
37 : }
```

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

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

1.3.2 Linking a Filters Handler to a Container List

In the listing above (line 35), we store the Filters Handler in the request object as an Attribute named: **directoryPeopleContainer_filter_handler**.

This name respect the format used to link a Filters Handler with a Container List.

Container_List_Name + "_filter_handler"

This Handler will be detected by Jahia and it will be used to apply filtering when loading the corresponding Container List.

1.4. Sort Options

Containers can be sorted on one Field at a time in Ascending or Descending order.
As with filters, you need to instantiate a Sort Handler, then store it in the request object.

Two information are required to init the Sort Handler:

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

1.4.1 Sort Handler

The listing below illustrates how to create the Sort Handler.

Note : **The code below must be placed before the Container List declaration!**

A : Retrieve the name of the Field on which to sort.
Here, this parameter is named **directoryPeopleContainer_sort** .

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

```

B : Retrieve the Sort Order.
Here, this parameter is named **directoryPeopleContainer_sort_order**.

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

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

```
if ( !peopleSort.equals("none") ){
     ContainerSorterBean sorter =
     new ContainerSorterBean("directoryPeopleContainer",
                             jParams,peopleSort);

     if ( !peopleSortOrder.equals("asc") ){
         sorter.setDescOrdering();
     }
     // Store the sort handler in the request object.
     // It will be used later by the container list loader.
     request.setAttribute("directoryPeopleContainer_sort_handler",
                          sorter);
}
```

1.4.2 Linking a Sort Handler to a Container List

In the listing above (line 60), we store the Sort Handler in the request object as an Attribute named : **directoryPeopleContainer_sort_handler**.

This name respects the format used to link a Sort Handler with a Container List.

Container_List_Name + "_sort_handler"

This Handler will be detected by Jahia and it will be used to apply ordering when loading the corresponding Container List.

1.4.3 Sort values as number

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

In example, when sorting people on the Field "People Rate", we want to apply a Number ordering. In the Sort Handler Constructor, we set the fourth parameter to **true**.

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

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

sorter.setNumberOrdering(**true**);

CHAPTER 2 CONTAINER LIST PAGINATION

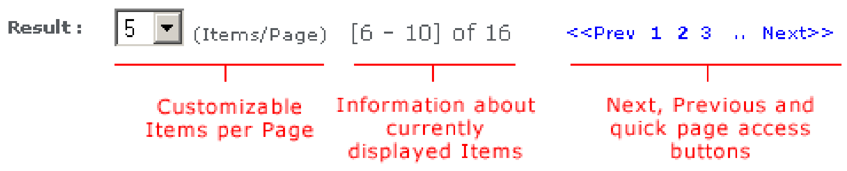
The Container List Pagination is an improvement of the initial Container List Scrolling mechanism done by Serge Huber that was available in Jahia 2.1.

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

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

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

Figure 2-1 : Elements of Container List Pagination.


Result :
5
(Items/Page)
[6 - 10] of 16
<<Prev 1 2 3 .. Next>>
Customizable Items per Page
Information about currently displayed Items
Next, Previous and quick page access buttons

2.1. GENERAL CONSIDERATIONS

2.1.1 Enclosing Page Form Tag and Container List Tags.

These elements are implemented using Tags that must be enclosed inside a Container List Tag :

<jahia:**containerList** name="directoryPeopleContainer">

... // Elements of Pagination go here.

</jahia:**containerList**>

If you want these elements to be implemented for a Post request (form submission), you need to enclose the previous code inside a Jahia Page Form (HTML form which action is an URL requesting the current page, see Chapter 1 , topic 1) :

```
<jahia:jahiaPageForm name="jahiapageform">
    <jahia:containerList name="directoryPeopleContainer">

        ... // Elements of Pagination go here.

    </jahia:containerList>
</jahia:jahiaPageForm>
```

You need to include the Jahia default javascript file **jahia.js** too.

Simply add the JSToolsTag at the top of your template file :

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

2.1.2 Working with paginatable Container List.

These features are only usefull if they are used with paginatable Container Lists.

You define a default Item per page value (called Window Size) at the same time you declare these Container List.

Declaration using the API :

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

Declaration using Tag lib :

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

2.1.3 Tracking the container scrolling value of the currently displayed Page.

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

Add the Tag **cListPaginationCurrentPageScrollingValue** with the attribute valueOnly set to false as below :

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

This Tag need to be enclosed inside a **containerList** Tag and a Jahia Page Form.

It will generate a hidden input used by Jahia to keep track of the current position when scrolling through the list of Containers:

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

2.2. Next, Previous Buttons

There are two Tags that help you implementing these buttons:

- previousWindowButton Tag

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

- nextWindowButton Tag

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

These Tags used as above will generate respectively these URLs :

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

and

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

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

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

You will have URLs that look like :

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

and

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

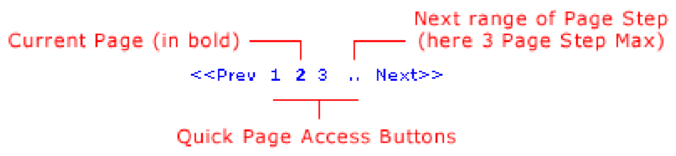
The attributes of these Tags are given below :

Table 2-1 : previousWindowButton & nextWindowButton Tags' Attributes

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

2.3. Quick Page Access Buttons

Figure 2-2 : Quick Page Access Buttons Details.



Current Page (in bold)
Next range of Page Step (here 3 Page Step Max)
<<Prev 1 2 3 .. Next>>
Quick Page Access Buttons

The figure above shows some interesting elements :

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

There are 5 Container List Pagination Tags.

As with the Next and Previous Tags, they must be enclosed inside a **containerList** Tag.

cListPagination Tag :

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

Table 2-2 : cListPagination Tag Attributes

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

cListPaginationPreviousRangeOfPages,cListPaginationNextRangeOf-Pages Tags :

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

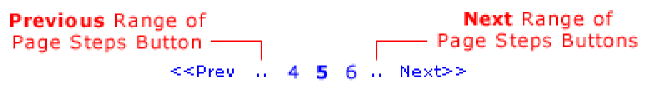

Previous Range of Page Steps Button
Next Range of Page Steps Buttons
<<Prev .. 4 5 6 .. Next>>

Figure 2-3 : Previous & Next Range of page Steps Button

In the figure above, the maximun of Quick Page Access Buttons to display per page has been set to 3 in the enclosing **cListPagination Tag.**

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

    <jahia:cListPaginationNextRangeOfPages method="post"
              formName="jahiapageform" title="&#160;..&#160;"/>
</jahia:cListPagination>
```

Table 2-3 : cListPaginationPreviousRangeOfPages & cListPaginationNextRangeOfpages Tag Attributes

Attributes	Description	Mandatory
title	The title of the button.	no
method	If you want to implement a Post (form submission) request version, you need to set this attribute to “**post**”. By default the method is “**get**”.	no
formName	The form name needed to generate the form submit Javascript code. The value must refers to the corrent enclosing Jahia Page Form Name. **It is mandatory when the method attribute is set to “post”**.	no

cListPaginationPageUrl Tag :

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

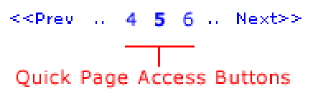


Figure 2-4 : Quick Page Access Buttons.

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

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

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

        <jahia:cListPaginationPageUrl method="post"
             formName="jahiapageform" />&#160;

    <jahia:cListPaginationNextRangeOfPages method="post"
              formName="jahiapageform" title="&#160;..&#160;"/>
</jahia:cListPagination>
```

Table 2-4 : cListPaginationPageUrl Tag Attributes

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

ifCListPaginationCurrentPage Tag:

When iterating through the Quick Page Access Buttons list, this Tag can be used to check if the current Quick Page Access Button to display refers to the **currently Displayed page**. If so, we can highlight this button using Bold caracters, etc...

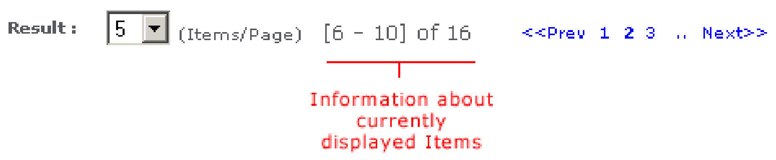
The code below show where this Tag take place:

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

        <jahia:ifCListPaginationCurrentPage><b>
        </jahia:ifCListPaginationCurrentPage>
            <jahia:cListPaginationPageUrl method="post"
                formName="jahiapageform" />&#160;
        <jahia:ifCListPaginationCurrentPage></b>
        </jahia:ifCListPaginationCurrentPage>

    <jahia:cListPaginationNextRangeOfPages method="post"
              formName="jahiapageform" title="&#160;..&#160;"/>
</jahia:cListPagination>
```

2.4. Container List Pagination Information Tags

There are 3 Tags used to give some information about the navigation through the paginated Container List.

Figure 2-5 : Pagination Information Elements



Their usages are quite simple. They only need to be enclosed inside the containerList Tag.

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

cListPaginationFirstItemIndex Tag

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

cListPaginationLastItemIndex Tag

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

cListPaginationTotalSize Tag

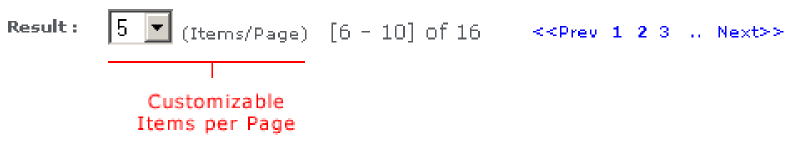
This Tag draws the total number of Containers of the enclosing Container List.

2.5. Customizable Items Per page Option

It is possible to allow the user to change the number of items per page used in the Pagination of a given Container List.

This optional Window Size will override the one set when declaring the Container List.

Figure 2-6 : Customizable Items Per Page Option.



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

Container_List_Name + "_windowsize"

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

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

CHAPTER 3 APPENDIX A - TEST MATERIALS

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

There are two ways to deploy this template set:

Deploying as Shared Template Set

1. Copy the file filter_demo.jar in the following directory :

   <jahia-home>/WEB-INF/var/shared_templates

2. Restart Jahia
3. Create a new Virtual Site with the template set named Filter_Demo.

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

Deploying as New Template Set

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

1. Copy the file filter_demo.jar in the following directory :

   <jahia-home>/WEB-INF/var/new_templates/**myjahiasite**

   Jahia will detect the new template set and automatically deploy it for the given site.

2. Now a new Template named "Filters" should appear in the available templates list. Create a new Page using this template.

CHAPTER 4 APPENDIX B - CONTAINERFILTERBEAN API

The Javadoc of the class ContainerFilterBean is given here. You should only care of all the "**AddClause**" methods that allow you to implement different type of Filter.

The listing below illustrates the "Rate" filter used to filter people whose rate are between 180 and 300 .

```
//-----------------------------------------------------------
// Rate filter (template people)
//-----------------------------------------------------------
// Handle filtering on field "directoryPeopleRate"

containerFilter = new ContainerFilterBean("directoryPeopleRate",true);//
Adding filtering clause
//containerFilter.addClause(ContainerFilterBean.COMP_BIGGER,"180");
containerFilter.addRangeClause(ContainerFilterBean.COMP_BIGGER,
   ContainerFilterBean.COMP_SMALLER_OR_EQUAL,
   "180",
   "300");
```

Note : the second attribute in the Constructor is "**true**" to force number filtering.

org.jahia.data.containers

Class ContainerFilterBean

```
java.lang.Object
  |
  +--org.jahia.data.containers.ContainerFilterBean
```

public final class **ContainerFilterBean**
extends Object

Jahia Standard container filtering for a given container list.

Author:
Khue Nguyen khue@jahia.org

See Also:
`FilterClause`, `ContainerFilters`, `JahiaContainerSet`

Field Summary

static String	**COMP_BIGGER** ">" Comparator
static String	**COMP_BIGGER_OR_EQUAL** ">=" Comparator
static String	**COMP_EQUAL** "=" Comparator
static String	**COMP_SMALLER** "<" Comparator
static String	**COMP_SMALLER_OR_EQUAL** "<=" Comparator

Constructor Summary

ContainerFilterBean(String fieldName)   Constructor
ContainerFilterBean(String fieldName, boolean numberFiltering)   Constructor

Method Summary

void	**addClause**(String comparator, String value) Add a simple comparison clause with a single value I.E : comparator = ContainerFilterBean.COMP_BIGGER (>) value = '1' will be used to generate the WHERE clause : WHERE (fieldvalue>'1')
void	**addClause**(String comparator, String[] values) Add a simple comparison clause with multiple values An OR comparison is added between each clause.

void	**addDateClause**(String lowerComp, String upperComp, Date lowerVal, Date upperVal) Constructs a range clause matching date values between `lower` and `upper`.
void	**addDateClause**(String lowerComp, String upperComp, long lowerVal, long upperVal) Constructs a range clause matching date values between `lower` and `upper`.
void	**addEqualClause**(String value) Add a simple equality comparison clause with a single value I.E : value = '1' will be used to generate the WHERE clause : WHERE (fieldvalue>'1')
void	**addEqualClause**(String[] values) Add a simple equality comparison clause with multiple value I.E : values = {'1','3','1000'} will be used to generate the WHERE clause : WHERE (fieldvalue='1' OR fieldvalue='3' OR fieldvalue='1000')
void	**addRangeClause**(String lowerComp, String upperComp, String lowerVal, String upperVal) Constructs a range clause matching values between `lowerVal` and `upperVal`.
void	**addTodayDateClause**() Constructs a range clause matching date values that are in Today date.
void	**addXDayMaxDateClause**(int nbDays) Constructs a range clause matching date for X day ago.
BitSet	**doFilter**(int ctnListID) Perform filtering.
Vector	**getClauses**() Return the vector of clauses.
String	**getFieldName**() Return the field name
boolean	**getNumberFiltering**() Return the number filtering status.
String	**getSelect**(int ctnListID) Return the select statement, build with the clauses for a given container list id.
void	**setNumberFiltering**(boolean val) You can force field values to be corverted to long representation for filtering comparison Prefer String comparison when possible, because it is faster (only one DB query needed).

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

COMP_EQUAL

`public static final String` **`COMP_EQUAL`**

"=" Comparator

COMP_SMALLER

`public static final String` **`COMP_SMALLER`**

"<" Comparator

COMP_SMALLER_OR_EQUAL

```
public static final String COMP_SMALLER_OR_EQUAL
```

"<=" Comparator

COMP_BIGGER_OR_EQUAL

```
public static final String COMP_BIGGER_OR_EQUAL
```

">=" Comparator

COMP_BIGGER

```
public static final String COMP_BIGGER
```

">" Comparator

Constructor Detail

ContainerFilterBean

```
public ContainerFilterBean(String fieldName)
```

Constructor
Parameters:
`String` - fieldName, the field name of the field on which to apply filtering.

ContainerFilterBean

```
public ContainerFilterBean(String fieldName,
                           boolean numberFiltering)
```

Constructor
Parameters:
`String` - fieldName, the field name of the field on which to apply filtering.
`boolean` - numberFiltering, if true force to convert filed value to long representation

Method Detail

addClause

```
public void addClause(String comparator,
                      String value)
```

Add a simple comparison clause with a single value

```
                    I.E : comparator = ContainerFilterBean.COMP_BIGGER (>)
```

```
                                value          = '1'

                                will be used to generate the WHERE clause :

                                      WHERE (fieldvalue>'1')
```

Parameters:
`String` - comparator, the comparator used to compare the field value.
`String` - value, a single value

---

addClause

```
public void addClause(String comparator,
                      String[] values)
```

Add a simple comparison clause with multiple values An OR comparison is added between each clause.

```
                      I.E : comparator = ContainerFilterBean.COMP_EQUAL (=)
                                values           = {'1','3','1000'}

                                will be used to generate the WHERE clause :

                                      WHERE (fieldvalue='1' OR fieldvalue='3' OR fieldvalue
```

Parameters:
`String` - comparator, the comparator used to compare the field value with each value of the values array.
`String[]` - values, an array of values as String

---

addEqualClause

```
public void addEqualClause(String value)
```

Add a simple equality comparison clause with a single value

```
                      I.E :
                                value          = '1'

                                will be used to generate the WHERE clause :

                                      WHERE (fieldvalue>'1')
```

Parameters:
`String` - value, a single value

---

addEqualClause

```
public void addEqualClause(String[] values)
```

Add a simple equality comparison clause with multiple value

```
                      I.E :
```

```
                     values           = {'1','3','1000'}

                     will be used to generate the WHERE clause :

                           WHERE (fieldvalue='1' OR fieldvalue='3' OR fieldvalue
```

Parameters:

`String` - value, a single value

---

addRangeClause

```
public void addRangeClause(String lowerComp,
                           String upperComp,
                           String lowerVal,
                           String upperVal)
```

Constructs a range clause matching values between `lowerVal` and `upperVal`.

```
           I.E : lowerComp = ContainerFilterBean.COMP_BIGGER_OR_EQUAL (>=)
                 upperComp = ContainerFilterBean.COMP_SMALLER (<)
                 lowerVal      = '1'
                 upperVal      = '1000'

                 will be used to generate the WHERE clause :

                       WHERE (fieldvalue>='1' AND fielValue<'10001')
```

Parameters:

`String` - lowerComp, the lower comparator
`String` - upperComp, the upper comparator
`String` - lowerVal, the lower value
`String` - upperVal, the upper value

---

addDateClause

```
public void addDateClause(String lowerComp,
                          String upperComp,
                          Date lowerVal,
                          Date upperVal)
```

Constructs a range clause matching date values between `lower` and `upper`. Available only with field of type JahiaDateField (date field).

```
           I.E : lowerComp = ContainerFilterBean.COMP_SMALLER (<)
                 upperComp = ContainerFilterBean.COMP_BIGGER_OR_EQUAL (>=)
                 lowerVal      = '1020038400100' ( long representation )
                 upperVal      = '1020038400000' ( long representation )

                 will be used to generate the WHERE clause :

                       WHERE (fieldvalue<'1020038400100' AND fielValue>='102
```

Parameters:

`String` - lowerComp, the lower comparator
`String` - upperComp, the upper comparator
`Date` - lowerVal, the lower date

`Date` - upperVal, the upper date

---

addDateClause

```
public void addDateClause(String lowerComp,
                          String upperComp,
                          long lowerVal,
                          long upperVal)
```

Constructs a range clause matching date values between `lower` and `upper`. Available only with field of type JahiaDateField (date field).

```
                     I.E : lowerComp = ContainerFilterBean.COMP_SMALLER (<)
                           upperComp = ContainerFilterBean.COMP_BIGGER_OR_EQUAL (>=)
                           lowerVal      = '1020038400100'  ( long representation )
                           upperVal      = '1020038400000'  ( long representation )

                           will be used to generate the WHERE clause :

                                 WHERE (fieldvalue<'1020038400100' AND fielValue>='102
```

Parameters:
`String` - lowerComp, the lower comparator
`String` - upperComp, the upper comparator
`long` - lowerVal, the lower date
`long` - upperVal, the upper date

---

addXDayMaxDateClause

```
public void addXDayMaxDateClause(int nbDays)
```

Constructs a range clause matching date for X day ago. Available only with field of type JahiaDateField (date field).
Parameters:
`int` - nbDays

---

addTodayDateClause

```
public void addTodayDateClause()
```

Constructs a range clause matching date values that are in Today date. Available only with field of type JahiaDateField (date field).