DOCUMENTATION

How-to use Filters

Jahia's next-generation, open source CMS stems from a widely acknowledged vision of enterprise application convergence – web, document, search, social and portal – unified by the simplicity of web content management.

Jahia Solutions Group SA

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

http://*www.jahia.com*

Summary

1 Introduction

At the hearth of Jahia is the mechanism of filtering. During the rendering of web pages, content grabbed from the JCR goes through several consecutive filters (like a chain) that operates transformations or internal operations (like putting fragments in cache for instance).

As a developer, you can add your own filters to add your own operations. One of the main interests is that the content delivered to the visitors is modified according to the filters it went through but remains clean and untouched in the repository.

Thanks to *prepare*, *execute* and *finalize* methods, the developer can put some information in the rendering context before the HTML generation, modify the generated fragment and start some operations after the finalization of the object rendering.

The goal of this documentation is to explain you how to create a filter "structure" in a Jahia 6.5 module, and then to give you a concrete example.

How to do a filter directly from your module in Jahia 6.5 ?

1.1 Pre-requisites

- Have a running Maven configuration
- Have a Java IDE that supports Maven (Eclipse + M2 / IntelliJ / Netbeans / …)

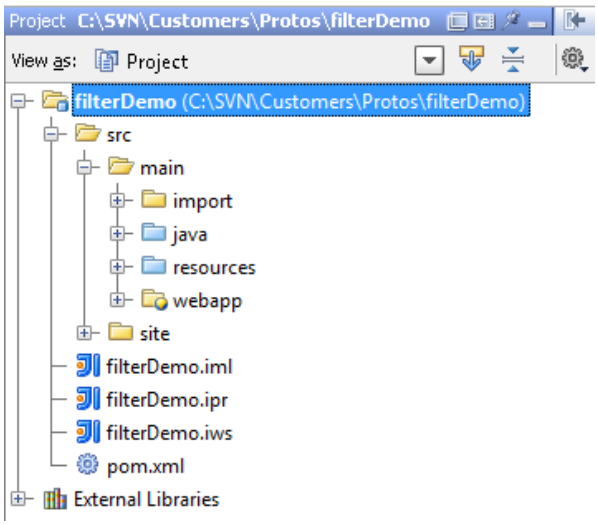
1.2 Generate your module

In command line, enter:

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

When prompted, choose:

3: http://maven.jahia.org/maven2 -> jahia-module-archetype (Jahia archetype for creating a new module (for Jahia version >= 6.5))



Then fulfill the different values :

artifactId : filterDemo

jahiaPackageVersion : 6.5.0

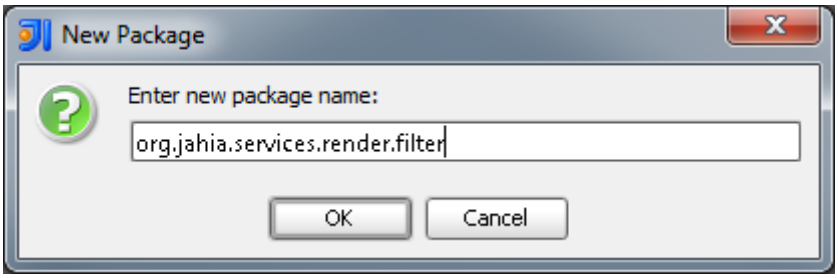
moduleName : Filter Demo

Your module is now generated. It is a maven project than you can open in your IDE.

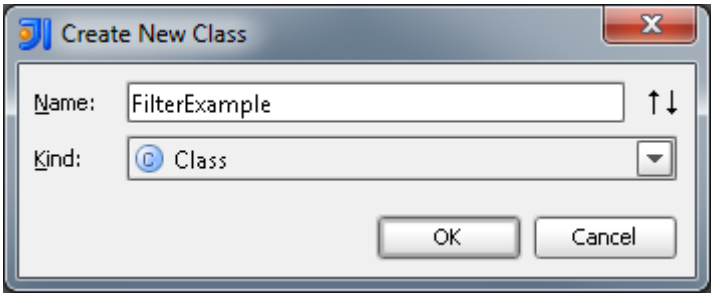
1.3 *Prepare your filter*

Let's go ! First of all, create a new package in the java folder: org.jahia.services.render.filter

/!\ You cannot use your own package name, as Jahia use this to find filters to load.



In this new package, create a new java class, with the name of your choice: FilterExample



This class must extends AbstractFilter :

```
public class FilterExample extends AbstractFilter{

}
```

Then there are three interesting method to override: prepare, execute, finalize.

```
public class FilterExample extends AbstractFilter{

    public String prepare(RenderContext renderContext, Resource resource, RenderChain chain) throws Exception {
        return null;
    }

    public String execute(String previousOut, RenderContext renderContext, Resource resource, RenderChain chain)       throws Exception {

        return previousOut;
    }

    public void finalize(RenderContext renderContext, Resource resource, RenderChain renderChain) {
    }

}
```

- When a resource is called by the end user, we enter in the prepare method. This method allows to put some information in the scope of the request before the generation of the HTML output.

- After the resource rendering (HTML output generation), we enter in the execute method, it allows to modify the generated HTML fragment before to return it to the end user.

- Finally, when the fragment is finalized, we enter in the finalize method, it allows to reset some things in the context or to reinitialize some counters / ... on server side. At this step it is no more possible to interact with the generated HTML fragment.

Finally you need to reference your filter by creating a spring bean configuration. For that, rename ./src/main/webapp/META-INF/spring/filterDemo.xml.disabled to filterDemo.xml and edit the file.

The minimum “things” to set are of course the bean class:

org.jahia.services.render.filter.FilterExample

And also the priority, which defines the order of execution.

```
<bean class="org.jahia.services.render.filter.FilterExample">
    <property name="priority" value="21" />
</bean>
```

/!\ Be careful, in most of cases the priority will be > to 16. Why ? Because the cache filter has a priority of 16. So if your filter priority is < to 16, you will be evaluated each time a page is called. At the opposite, if you have a priority > to 16, your filter is executed only the first time the page is called and then the result of the treatment is cached. Impact of performances can be very important, so please set a priority > to 16 when it is possible.

Usually, we also define additional properties to reduce the scope of application of the filter; here is the list of available properties:

- applyOnMainResource - the filter will be applied only on the main resource
- applyOnModules - comma-separated list of module names this filter will be executed for (all others are skipped)
- applyOnNodeTypes - comma-separated list of node type names this filter will be executed for (all others are skipped)
- applyOnTemplates - comma-separated list of template names this filter will be executed for (all others are skipped)
- applyOnTemplateTypes - comma-separated list of template type names this filter will be executed for (all others are skipped)
- applyOnConfigurations – comma-separated list of configuration this filter will be executed for (all others are skipped)

- skipOnModules- comma-separated list of module names this filter won't be executed for
- skipOnNodeTypes - comma-separated list of node type names this filter won't be executed for
- skipOnTemplates - comma-separated list of template names this filter won't be executed for
- skipOnTemplateTypes - comma-separated list of template type names this filter won't be executed for
- skipOnConfigurations – comma-separated list of configuration this filter won't be executed for (all others are skipped)

You can find examples of this parameters in applicationcontext-renderer.xml.

Here are the only things to do to have a working filter in Jahia 6.5 module. You can use this filter initialization to implement your own filter, or continue the tutorial and do the exercise.

1.4 Filter examples

Now that we have a ready to go filter structure, let's use it. The first example shows you an example that uses the prepare method, the second example shows you an example that uses the execute method.

1.4.1 "Who is on this page" module

We want to do a module that is able to list all the users who are looking the current page. For that, we will use a filter to generate a list of users who are looking the current page. The idea is at each HTTP call the filter updates a map which associates a user with a page, and put the list of users associated with the current page in the request . Then a very simple module displays this list in the page. Let's do it …

The first step is to implement the code in the filter …

```
public class FilterExample extends AbstractFilter{

    private Map<String,String> userMap = new HashMap<String, String>();

    public String prepare(RenderContext renderContext, Resource resource, RenderChain chain) throws Exception {

        //Associate the currentPage and the currentUser
        userMap.put(renderContext.getUser().getUsername(),resource.getNode().getIdentifier());

        //Put in the request the list of user that read the current page
        List<String> userList = getKeysFromValue(userMap, resource.getNode().getIdentifier());
        renderContext.getRequest().setAttribute("userList",userList);


        return null;
    }

    public List<String> getKeysFromValue(Map<String, String> hm, String value){
    List <String>list = new ArrayList<String>();
    for(String o:hm.keySet()){
        if(hm.get(o).equals(value)) {
            list.add(o);
        }
    }
    return list;
  }

}
```

Nothing complicated, first we declare a map in our class:

```
private Map<String,String> userMap = new HashMap<String, String>();
```

Then we override the prepare method and put the logic inside:

```
//Associate the currentPage and the currentUser
userMap.put(renderContext.getUser().getUsername(),resource.getNode().getIdentifier());

//Put in the request the list of user that read the current page
List<String> userList = getKeysFromValue(userMap, resource.getNode().getIdentifier());
renderContext.getRequest().setAttribute("userList",userList);

//Nothing to put in the html out
return null;
```

Our filter is now ready. We now need to adapt the spring configuration … Our filter must be executed each time the page is displayed and never cached. So we must put a priority < to 16. Moreover, we want to apply this filter on all “page” rendering and not on all nodes rendering. But we want to include real page (jnt:page) and also virtual page (generated by content template). To catch all these cases, we will use the applyOnConfigurations > page parameter. Finally, this filter is only useful in live mode, but it could be nice to have a a preview. So let apply the filter in live and preview mode only :

```
<bean class="org.jahia.services.render.filter.FilterExample">
    <property name="priority" value="14" />
    <property name="applyOnConfigurations" value="page" />
    <property name="applyOnModes" value="live,preview" />
</bean>
```

The filter is now called at the right time, on the right object, so the latest thing to do is to implement a small component that retrieve the userList attribute from the request and display it …

Edit : ./src/webapp/META-INF/spring/definitions.cnd

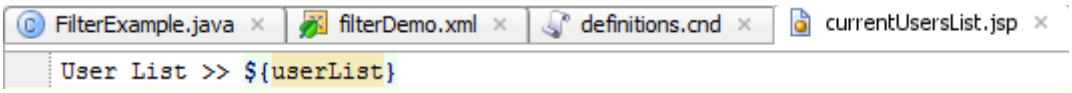
Put : [jnt:currentUsersList] > jnt:content, jmix:siteComponent

```
<jnt = 'http://www.jahia.org/jahia/nt/1.0'>
<jmix = 'http://www.jahia.org/jahia/mix/1.0'>

[jnt:currentUsersList] > jnt:content, jmix:siteComponent
```

Create folder : ./src/webapp/jnt_currentUsersList/html/

Create and edit the file : ./src/webapp/jnt_currentUsersList/html.currentUsersList.jsp



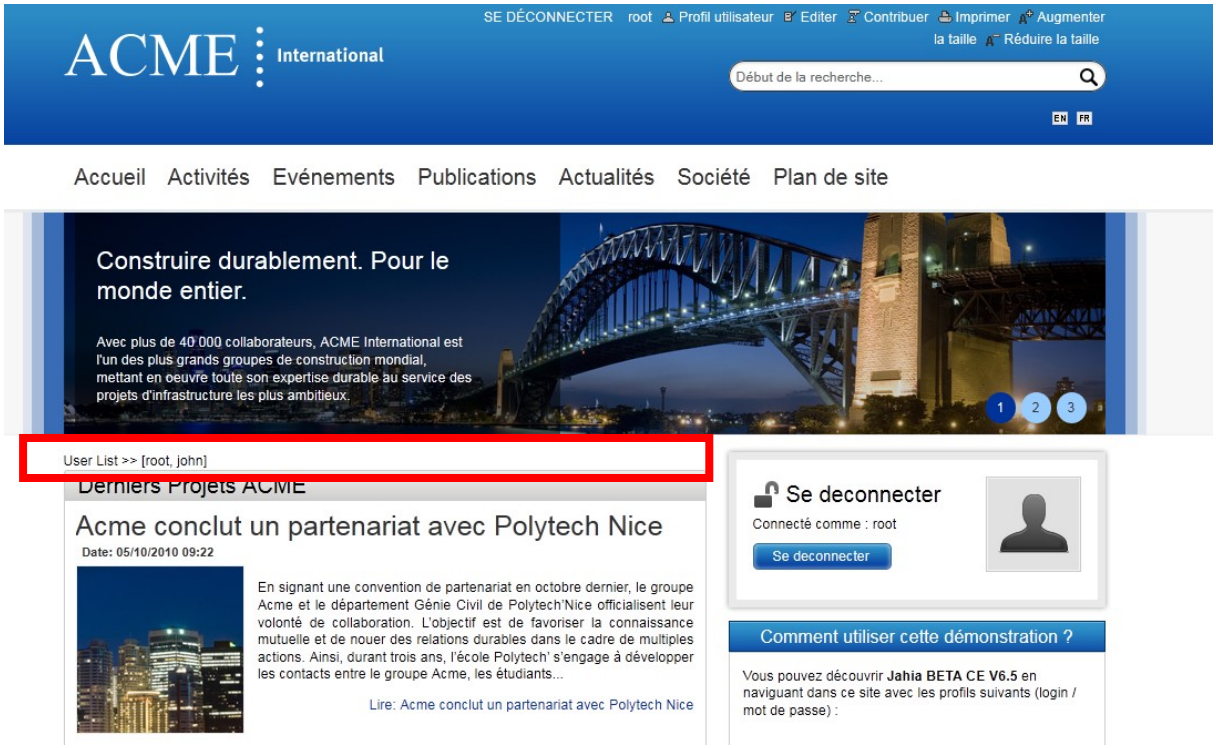
Nothing special to do, just do an output of the userList variable … ${urerList} No cosmetic stuff in this tutorial ☺ !

Also create a file : ./src/webapp/jnt_currentUsersList/html.currentUsersList.properties

And put it inside : cache.expiration=0

It avoids caching this component, because it changes at each page generation.

Deploy your module, drag & drop currentUsersList in the page … And here is the result:



Of course it is a very “basic” implementation … We should handle user disconnection and things like that, but the idea here is to focus on filters.

1.4.2 “E-mail obfuscator” module

We want to protect e-mail of our web-sites from robots. All modules should be protected including rich text. The easiest way to do that, is to do a filter that evaluates the ouput HTML buffer to detect e-mail and replaces them by obfuscated version.

Let’s do it …

The first step is to implement the code in the filter …

A “simple” execute method that parses the code, detects e-mail and replaces it by an entity version. This code is based on http://obfuscatortool.sourceforge.net . You can get the source file in the zip package associated to this how to tutorial.

```
public String execute(String previousOut, RenderContext renderContext, Resource resource, RenderChain chain)
        throws Exception {
    StringBuffer wholeHtml = new StringBuffer(previousOut);

    StringTokenizer st = new StringTokenizer(previousOut);

    while (st.hasMoreTokens()) {
        String current = st.nextToken();
        if (containsAddress(current)) {
            String[] split = current.split(addrSpec, 2);
            // separate the email out
            String email = current.substring(split[0].length(), current.length() - split[1].length());

            // now go through all occurances of the found email in the document
            int index = wholeHtml.indexOf(email);
            int lastIndex = index;

            // as long as we still find one, keep going
            while (index != -1) {

                // index to search from next time
                lastIndex = index + 1;

                String entityVersion;

                // check for mailto:
                if (index > 7 && wholeHtml.substring(index - 7, index).equals("mailto:")) {
                    entityVersion = convertToHtmlEntity("mailto:" + email);
                    wholeHtml.replace(index - 7, index + email.length(), entityVersion);
                } else {
                    entityVersion = convertToHtmlEntity(email);
                    wholeHtml.replace(index, index + email.length(), entityVersion);
                }

                // get the next index of the email address!
                index = wholeHtml.indexOf(email, lastIndex);
            }

        }
    }
    return wholeHtml.toString();
```

Then, when the method is implemented, we now need to adapt the Spring configuration. We don't want to re-evaluate the filter each time we generate the page. We want to cache the crypted version of the email. So we will use a priority > to 16. Moreover, we want to apply this filter on all "page" to treat the HTML buffer in "one shot". We want to include real page (jnt:page) and also virtual page (generated by content template). To catch all these cases, we will use the applyOnConfigurations > page parameter. Finally, this filter is only useful in live mode, but it could be nice to have a preview. So let apply the filter in live and preview mode only:

```
<bean class="org.jahia.services.render.filter.FilterExample">
    <property name="priority" value="21" />
    <property name="applyOnConfigurations" value="page" />
    <property name="applyOnModes" value="live,preview" />
</bean>
```

That's all ! Deploy your module, and it will not be called each time a page is generated for the first time to convert e-mail to entity crypted version :

Corresponding code source :

```
    <h4 class="box5-title box5-titleblue">Comment utiliser cette démonstration ?</h4>
<div class="box5">
    <div class="box5 box5padding box5marginbottom">
        <div class="box5-content">
                    <p>
        <a href="&#109;&#97;&#105;&#108;&#116;&#111;&#58;&#99;&#102;&#108;&#111;&#110;&#100;&#64;&#106
<p>
```

Of course, entity is maybe not the better encrypting algorithm to protect e-mail, but one more time the goal is to show you quickly what you can do with filters and how it works.

Jahia Solutions Group SA

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

http://*www.jahia.com*