

The Usecase Framework

Table of contents

1 Introduction.....	2
2 Directory Structure.....	2
2.1 The Lenya Core.....	2
2.2 Your Publication.....	2
3 Architecture.....	3
4 The Contract Between Flowscript And Usecase Handler.....	3
5 Implementing a Custom Usecase.....	3
5.1 Prerequisites.....	3
5.2 Add a Menu Item.....	3
5.3 Implement the Usecase Handler Class.....	3
5.4 Implement the View.....	4
6 Overriding Core Usecases in Publications.....	4
6.1 Overriding Usecase Handler Classes.....	4
6.2 Overriding JX Templates.....	4

1. Introduction

A usecase in Lenya means a user triggered action. In most cases, a usecase is triggered by a CMS menu option on a specific document of the publication. This document is the object of the usecases' action (such as edit, delete, publish, ...).

There are usecases which are independent of a specific document, such as the `ac.logout` usecase. In that case it does not matter on what document the usecase is triggered. The part of the request which specifies the document is simply ignored by usecases that are document independent.

The CMS menus trigger usecases by setting the `lenya.usecase` request parameter on the current document. If for example the user selects the *Publish* option from the *Workflow* menu, a request will be triggered such as: GET `http://www.server.com/lenya/default/authoring/tutorial.html?&lenya.usecase=publish`

The `Lenya.global-sitemap.xmap` will redirect requests with a `lenya.usecase` request parameter to the `$LENYA_WEBAPP/lenya/usecase.xmap` sub-sitemap. From version 1.4 on, the following pipeline in this sitemap is used to recognize usecases which are implemented in Java using the new 1.4 usecase framework:

```
<map:pipeline>
  <map:match type="registered-usecase">
    <map:mount src="usecases/usecase.xmap" uri-prefix="" check-reload="yes" reload-method="synchron"/>
  </map:match>
</map:pipeline>
```

The registered-usecase matcher's default implementation (`org.apache.lenya.cms.cocoon.matching.UsecaseRegistrationMatcher`) will use the Avalon component resolver mechanism to resolve the name of the usecase to a an Avalon component. In case it cannot resolve the usecase to an Avalon component, sitemap processing will continue and the usecase is treated in the traditional way using the usecase and step matchers (`org.apache.cocoon.matching.WildcardRequestParameterMatcher`). In order for this to work correctly, there should be a `lenya.step` parameter in the request.

If the usecase could be resolved successfully into an Avalon component, processing will continue in the `$LENYA_WEBAPP/lenya/usecases/usecase.xmap` (as opposed to `$LENYA_WEBAPP/lenya/usecase.xmap`) with the new JX and Java based 1.4 usecase framework.

The *usecase framework* in Lenya 1.4 is a simple framework to implement usecases using JX templates and Java. This approach is an "85% solution". It enables the user to implement a big range of common usecases.

Note:

Some special complex usecases might require a custom flowscript, in this case you can't use this framework.

2. Directory Structure

2.1. The Lenya Core

<code>\$LENYA_WEBAPP</code>	
<code>/lenya/usecases</code>	usecase-related files
<code>/usecase.xmap</code>	usecase dispatching sitemap
<code>/usecases.js</code>	flowscript for usecase control flow
<code>/admin</code>	Lenya admin usecases
<code>/addUser.jx</code>	JX templates for usecase views
<code>...</code>	more Lenya core usecases

2.2. Your Publication

`$PUB_HOME`

A usecase request - denoted by the request parameter `lenya.usecase` - is dispatched by `$LENYA_WEBAPP/lenya/usecases/usecase.xmap`. All usecases are handled by a single flowscript `$LENYA_WEBAPP/lenya/usecases/usecases.js`. This keeps javascript maintenance costs at a minimum.

Usecase framework architecture

1. `setup(String sourceUrl, Situation situation);` Initializes the handler.
2. `isInteractive()` Asks if the usecase is interactive, i.e. if a confirmation screen should be presented to the user.
3. `checkPreconditions()` This allows the handler to check if the usecase may be invoked in this situation.

5.4. Implement the View

1. The view for a usecase is implemented using a JX template.
2. The output of the view has to be a Lenya page:

```
<page:page
  xmlns:jx="http://apache.org/cocoon/templates/jx/1.0"
  xmlns:page="http://apache.org/cocoon/lenya/cms-page/1.0"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:i18n="http://apache.org/cocoon/i18n/2.1"
>

  <page:title><i18n:text>Edit Headline</i18n:text></page:title>
  <page:body>

    <form>
      <input type="hidden" name="lenya.continuation" value="{continuation.id}"/>
      <input type="hidden" name="lenya.usecase" value="{request.getParameter('lenya.usecase')}" />

      ...

    </form>

  </page:body>
</page:page>
```

3. Take care of adding the hidden `lenya.usecase` and `lenya.continuation` fields as shown above.
4. The template has to be located at `$PUB_HOME/lenya/usecases/<usecase>.jx` (replace `.` with `/`), for instance

6. Overriding Core Usecases in Publications

6.1. Overriding Usecase Handler Classes

The usecase resolver, which is responsible for obtaining the handler class for a usecase, looks first if the current publication overrides the core usecase handler. This can be done by declaring a usecase called `<pub-id>/<usecase-name>`, for instance `mypub/admin.addUser`. To implement a core usecase using a custom handler class, you need to

1. Implement the handler class and put it in `$PUB_HOME/java/src`. In most cases, you will extend the core usecase handler class to inherit the basic functionality.
2. Declare it in an *xpatch* file, for instance `$PUB_HOME/config/usecases.xconf`:

```
<xconf xpath="/cocoon/usecases" unless="/cocoon/usecases/component-instance[@name =
'mypub/admin.addUser']">
  <component-instance name="mypub/admin.addUser"
    logger="lenya.usecases.editHeadline"
    class="org.myproject.lenya.usecases.AddUser"/>
</xconf>
```

Now, when the usecase is invoked from inside the publication `mypub`, the custom handler class will be used.

6.2. Overriding JX Templates

Overriding the JX template of a usecase follows the [publication templating](http://cocoon.apache.org/lenya/publication-templating/index.html) (`../publication-templating/index.html`) principle. You just have to put a JX template with the same name in `$PUB_HOME/lenya/usecases`, for instance `$PUB_HOME/lenya/usecases/admin/addUser.jx`.