# The Usecase Framework

## Table of contents

Copyright © 2002-2005 The Apache Software Foundation. All rights reserved.

## 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 simple 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.event=publish&leny`

The `lenya.event` parameter belongs to the Lenya workflow framework and can be ignored here. I can even be left out and the usecase will still work fine, but workflow actions will not be triggered.

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 recognonize 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

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

Copyright © 2002-2005 The Apache Software Foundation. All rights reserved.

### 2.2. Your Publication

```
$PUB_HOME
  /lenya/usecases                     usecase-related files
                 /editHeadline.jx     JX templates for usecase views
  /java/src/...                       usecase handler classes
```

## 3. Architecture

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.

The flowscript `usecases.js` determines the usecase handler class using the `org.apache.lenya.cms.usecase.UsecaseResolver`. All business code operations are delegated to the usecase handler class.

Usecase framework architecture

## 4. The Contract Between Flowscript And Usecase Handler

The usecase handler class has to implement the interface `org.apache.lenya.cms.usecase.Usecase`. The methods of this interface are called in a certain order when the usecase is invoked:

1. `setup(String sourceUrl, Situation situation);` Initializes the handler.
2. `isInteractive()` Asks is 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. Implementing a Custom Usecase

### 5.1. Prerequisites

1. Choose a name to identify the usecase, e.g. editHeadline. It is possible to group usecases using "." as delimiter, for instance *article.editHeadline*.

### 5.2. Add a Menu Item

**Note:**
This step is necessary if you want to call the usecase from the Lenya menubar.

1. Add the corresponding menu item:
   ```
   <item uc:usecase="article.editHeadline">Edit Headline</item>
   ```

### 5.3. Implement the Usecase Handler Class

1. Choose a name for your business logic class, e.g. `org.myproject.lenya.usecases.EditHeadline`.
2. The class must implement the interface `org.apache.lenya.cms.usecase.Usecase`.
3. To simplify development, you can extend one of the following classes:
   - `org.apache.lenya.cms.usecase.AbstractUsecase`
   - `org.apache.lenya.cms.usecase.DocumentUsecase` (only for usecases invoked on document pages)
   - `org.apache.lenya.cms.usecase.SiteUsecase`

   They have built-in support for the unit-of-work pattern (which will evolve into an ACID transaction someday) as well as functionality specific to the area they are supposed to be used with, e.g. the site area.
4. Add the usecase handler class declaration to an XPatch file, e.g. `$PUB_HOME/config/usecases.xconf`:
   ```
   <xconf xpath="/cocoon/usecases" unless="/cocoon/usecases/component-instance[@name =
   'article.editHeadline']">
   ```

Copyright © 2002-2005 The Apache Software Foundation. All rights reserved.

```
  <component-instance name="article.editHeadline"
                      logger="lenya.usecases.editHeadline"
                      class="org.myproject.lenya.usecases.EditHeadline"/>
</xconf>
```

### 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 (../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`.

Copyright © 2002-2005 The Apache Software Foundation. All rights reserved.