Resource Types

Table of contents

1. Introduction

A resource type defines a certain (XML or binary) source format, together with processing options. It typically consists of

- an XML structure definition (e.g., Relax NG), if the resource type is XML-based,
- some presentation pipelines,
- some presentation XSLT stylesheets,
- usecases to manipulate documents.

All of these can be shared between several resource types.

The information describing a resource type is managed by a `ResourceType` (http://lenya.apache.org/apidocs/2.0/org/apache/lenya/cms/publication/ResourceType.html) service. The default implementation is `ResourceTypeImpl`. It implements `ThreadSafe`, which ensures that only a single instance of every resource type is created. It is not possible to declare multiple resource types with the same name.

2. Choose a Unique Resource Type Name

You should choose a reasonable name for your resource type.

Note:

In the examples, we use the name *profile* (page with information about a person).

3. Adding the Resource Type to a Publication

The resource types used by a publication are declared in `publication.xml`, including the assignment of a workflow schema to a resource type. You have to add your resource type to the `<resource-types>` section of this file:

```
<publication>
  ...
  <resource-types>
    <resource-type name="xhtml" workflow="fallback://config/workflow/workflow.xml"/>
    <resource-type name="homepage" workflow="fallback://config/workflow/workflow.xml"/>
    <resource-type name="links" workflow="fallback://config/workflow/workflow.xml"/>
    <resource-type name="profile" workflow="fallback://config/workflow/workflow.xml"/>
    ...
  </resource-types>
  ...
</publication>
```

You can add references to any resource types configured in `<lenya-webapp>/WEB-INF/cocoon.xconf` (see below), containing

- resource types provided by modules,
- resource types of template publications, and

- resource types of the publication itself.

4. Providing One or More Sample XML Documents

If you want to enable users to create new resources belonging to your resource type, it is useful to provide one more more sample XML documents.

The sample document(s) are typically placed in the directory `$MODULE_HOME/samples`. You can choose arbitrary filenames, but for a single sample it is recommended to use the resource type name (e.g., `profile.xml`).

There are two ways of declaring the samples. The first option is very convenient if you don't need custom samples per publication. You just add the samples to the resource type declaration:

```
<component-instance name="profile" ...
  ...
  <sample-name name="Empty Profile" mime-type="application/xml">
    fallback://lenya/modules/profile/samples/empty.xml</sample-name>
  <sample-name name="Example Profile" mime-type="application/xml">
    fallback://lenya/modules/profile/samples/example.xml</sample-name>
  ...
</component-instance>
```

The second option allows you to specify a file where the samples declarations are stored. This enables you to override this file in your publications, providing a different sample list per publication:

```
<component-instance name="profile" ...
  ...
  <samples uri="fallback://lenya/modules/profile/samples/samples.xml"/>
  ...
</component-instance>
```

The `samples.xml` file uses the same syntax:

```
<?xml version="1.0" encoding="utf-8"?>
<samples>
  <sample-name name="Empty Profile" mime-type="application/xml">
    fallback://lenya/modules/profile/samples/empty.xml</sample-name>
  <sample-name name="Example Profile" mime-type="application/xml">
    fallback://lenya/modules/profile/samples/example.xml</sample-name>
</samples>
```

5. Providing an XML Structure Definition

This step is only needed if you want to edit resources with Lenya or validate them after they have been imported or manipulated. The type of the structure definition (XML Schema (http://www.w3.org/XML/Schema) , Relax NG (http://www.relaxng.org/) , ...) depends on the editor or validator you want to use. For instance, the BXE (http://www.bitfluxeditor.org/) WYSIWYG editor requires a Relax NG document.

The structure definition document is typically placed in the directory `$MODULE_HOME/resources/schemas`. The name of the file is arbitrary, but it is recommended to use the resource type name (e.g., `profile.rng`).

6. Creating a Workflow Schema

If your resources should have a workflow, you have to add a workflow schema for your resource type as described in Workflow Configuration (../../../docs/1_2_x/components/workflow/configuration.html) . A workflow schema can be shared between multiple resource types. The workflow schema is assigned to a resource type in `$PUB_HOME/config/publication.xml` (see section *Adding Resource Types to a Publication*).

7. Cache Expiration Time

For each resource type, you can configure how long the documents shall be cached by the client, for instance:

```
<expires seconds="3600" />
```

This value can be accessed via the `getExpires()` methods of the `ResourceType` and `Document` interfaces. You can use the header action to set the corresponding HTTP response header in your publication:

```
<map:act type="set-header">
  <map:parameter name="Expires"
    value="{date-iso8601-rfc822:{doc-info:{pubId}:{area}:{uuid}:{language}:expires}}"/>
</map:act>
```

8. The Resource Type Definition

To declare a custom resource type and assign the creator, schema etc. to it, add the component instance to an XPatch file (e.g., `$MODULE_HOME/config/cocoon-xconf/resourcetype-profile.xconf`):

```
<xconf xpath="/cocoon/resource-types"
       unless="/cocoon/resource-types/component-instance[@name = 'profile']">

  <component-instance name="profile"
                      logger="lenya.resourcetypes.profile"
                      class="org.apache.lenya.cms.publication.ResourceTypeImpl">

    <schema src="fallback://lenya/modules/profile/resources/schemas/profile.rng"
            language="http://relaxng.org/ns/structure/0.9"/>

    <sample-name name="Empty Profile" mime-type="application/xml">
        fallback://lenya/modules/profile/samples/empty.xml</sample-name>

    <sample-name name="Example Profile" mime-type="application/xml">
        fallback://lenya/modules/profile/samples/example.xml</sample-name>

    <link-attribute xpath="//*[namespace-uri() = 'http://foo.bar.org/profile']/@href"/>

    <format name="xhtml" uri="cocoon://modules/profile/profile.xml"/>

    <expires seconds="3600" />

  </component-instance>

</xconf>
```

This XPatch file will be used when the publication is deployed (e.g., when you issue a `./build`

command). Its contents will be patched into `<lenya-webapp>/WEB-INF/cocoon.xconf`.

9. I18n for the Resource Type Label

There is a convention that the message key *resourceType-{name}* is used for a human-readable, translated resource type label:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogue xml:lang="en" xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <message key="resourceType-profile">Profile</message>
</catalogue>
```

10. Define a Custom Menubar

If you want to use a custom menubar for your resource type, follow the guidelines on the page The Lenya Menubar (../../../docs/1_2_x/components/layout/lenya-menubar.html) . Typically, a menubar is shared between multiple resource types. Small customizations can be achieved with Java code in the menubar XSP.

To let the user create new resources using the `DefaultBranchCreator`, you have to add the following menu item:

```
<item uc:usecase="sitemanagement.create" href="?doctype=profile">
  <i18n:translate>
    <i18n:text>New ... Document</i18n:text>
    <i18n:param><i18n:text>resourceType-profile</i18n:text></i18n:param>
  </i18n:translate>
</item>
```

11. Formats

A resource type provides a set of *formats* to provide different ways of presenting content documents. The formats are defined in the resource type declaration in `$MODULE_HOME/config/cocoon-xconf/resourcetype-profile.xconf`:

```
  <format name="xhtml" uri="cocoon://modules/profile/xhtml.xml"/>
  <format name="include" uri="cocoon://modules/profile/xhtml-inline.xml"/>
```

The following figure illustrates some usage examples of formats:

- Example **A** shows the most typical case - a *msg* (message) document is rendered using the format *xhtml* to be displayed as page content.
- In example **B**, the format *teaser* is used to render teaser versions of the *msg* documents to be displayed in the sidebar. This is usually done using an include mechanism (see below).
- In example **C**, a *news* document includes the *teaser* versions of message documents. The *news* document in turn provides the format *xhtml* to be rendered as page content.

The `uri` attribute of the `format` element may refer to an arbitrary URL, which is typically a request into the module itself. This URI is matched inside the module sitemap (in our case, `modules/profile/sitemap.xmap`).

There are two ways in which the format can be requested:

- for the current document - without parameters
- for a specific document - with the parameters `/{pubId}/{area}/{uuid}/{language}`

Typically, an XSLT is applied to the content document to transform it into another format (XHTML, XSL-FO, ...). In the following example, the name of the XSLT stylesheet is supposed to end with the format name (e.g., `profile2xhtml.xsl`):

```
<!-- apply a format -->
<!-- {format}.xml (current document) -->
<map:match pattern="*.xml">
  <map:generate src="cocoon:/{1}/{page-envelope:publication-id}/ \
                                {page-envelope:area}/ \
                                {page-envelope:document-uuid}/ \
                                {page-envelope:document-language}"/>
  <map:serialize type="xml"/>
</map:match>

<!-- {format}.xml/{pubId}/{area}/{uuid}/{language} (specific document) -->
<map:match pattern="*.xml/*/*/*/*">
  <map:generate src="lenya-document:{4},lang={5}{link:rev}"/>
  <map:transform src="fallback://lenya/modules/profile/xslt/profile2{1}.xsl">
    <map:parameter name="rendertype" value="{request-param:rendertype}"/>
    <map:parameter name="nodeid" value="{doc-info:{2}:{3}:{4}:{5}:nodeName}"/>
    <map:parameter name="language" value="{4}"/>
  </map:transform>
  <map:serialize type="xml"/>
</map:match>
```

To request a formatted document, you can use the *format* parameter of the `site:` and `lenya-document:` protocols. You can issue such a request from an arbitrary sitemap (e.g., from your publication sitemap or from a different module sitemap).

```
<!-- aggregate navigation components and XHTML-formatted content -->
<map:aggregate element="cmsbody">
  <map:part src="cocoon://modules/sitetree/{2}/{3}/breadcrumb/{5}.xml"/>
  <map:part src="cocoon://modules/sitetree/{2}/{3}/tabs/{5}.xml"/>
  <map:part src="cocoon://modules/sitetree/{2}/{3}/menu/{5}.xml"/>
  <map:part src="cocoon://modules/sitetree/{2}/{3}/search/{5}.xml"/>
  <map:part src="lenya-document:{page-envelope:documente-uuid}?format=xhtml"/>
</map:aggregate>
```

Another usage scenario is to include formatted documents in other documents, e.g. as a teaser or summary (see the figure above):

```
<ci:include src="site:/{$language}{$path}?format=teaser"/>
```

```
<ci:include src="lenya-document:{$uuid}?format=teaser"/>
```

Since formats allow you to select the presentation style of a document on demand, you can use them as layout templates. But unfortunately formats have to be declared in `cocoon.xconf`, so you can't add new templates at runtime with this approach at the moment. Here's an example how to use formats to select templates:

- Set the Dublin Core element `format` when you create the document, e.g. using a combo box. We use the Dublin Core for simplicity here, for a cleaner approach you should define your own meta

data element set.

- In the resource type module, provide an XSLT stylesheet `{resource-type}2{format}.xsl` for each format (see above)
- Add the format parameter to the `lenya-document` call:

```
<map:generate src="lenya-document:{page-envelope:uuid}?format={dublincore:format}"/>
```

12. Presentation

To make your resources available as HTTP pages, you have to add the appropriate pipelines and XSLT stylesheets. In general, there are no restrictions.

The pipelines have to be placed in `$MODULE_HOME/sitemap.xmap`. The stylesheets are typically located in `$MODULE_HOME/xslt/` and named `{resource-type}2{format}.xsl` (e.g., `profile2xhtml.xsl`).

In the case of the *xhtml* format, the stylesheet is supposed to generate a valid XHTML document (in the XHTML namespace). The output of other formats depends on the purpose of the respective format.