Meta Data

Table of contents

1. Introduction

Meta data are organized in *element sets*. An element set is identified using a namespace URI. Each element set can supply a fixed set of elements. An element is identified using a name. An element can be *editable*, and it can support *multiple values*.

2. Registering Meta Data Element Sets

Element sets are declared using patch files for `cocoon.xconf`. When the application starts up, they are registered with the `MetaDataRegistry`. Here's an example:

```
<xconf xpath="/cocoon/meta-data"
    unless="/cocoon/meta-data/component-instance
      [@name = 'http://apache.org/lenya/metadata/media/1.0']">
  <component-instance name="http://apache.org/lenya/metadata/media/1.0"
    class="org.apache.lenya.cms.metadata.ConfigurableElementSet">
    <element name="filename" multiple="false"/>
    <element name="format" multiple="false"/>
    <element name="extent" multiple="false"/>
    <element name="width" multiple="false"/>
    <element name="height" multiple="false"/>
    <element name="caption" multiple="false" editable="true"/>
  </component-instance>
</xconf>
```

3. Accessing Meta Data

Here's an example for accessing the meta data of a document:

```
MetaData meta = document.getMetaData("http://myproject.org/metadata/1.0");
String description = meta.getFirstValue("description");
String[] references = meta.getValues("references");
```

To find out which element sets are registered, you can access the `MetaDataRegistry`:

```
MetaDataRegistry registry = null;
try {
    registry = (MetaDataRegistry) this.manager.lookup(MetaDataRegistry.ROLE);
    String[] namespaces = registry.getNamespaceUris();
    ...
}
finally {
    if (registry != null) {
        this.manager.release(registry);
    }
}
```

4. The Meta Data Input Module

You can use the `MetaDataModule` to make an element set accessible in Cocoon sitemaps. To declare it, use a patch file for `cocoon.xconf`:

```
<xconf xpath="/cocoon/input-modules"
  unless="/cocoon/input-modules/component-instance[@name = 'mymeta']">
  <component-instance logger="sitemap.modules.input.mymeta" name="mymeta"
    class="org.apache.lenya.cms.cocoon.components.modules.input.MetaDataModule"
    namespace="http://myproject.org/metadata/1.0"/>
</xconf>
```

Now you can access the meta data in your pipelines:

```
<map:transform src="...">
  <map:parameter name="description" value="{mymeta:description}"/>
</map:transform>
```

5. Storage

In 1.4.x meta data is stored separately from the document content but in the same directory (index_{lang}.meta). A typical sample for a meta data XML document may be the following:

```
<metadata xmlns="http://apache.org/lenya/metadata/1.0">
  <element-set namespace="http://apache.org/lenya/metadata/media/1.0">
    <element key="width">
      <value>300</value>
    </element>
    <element key="height">
      <value>374</value>
    </element>
    <element key="extent">
      <value>30291</value>
    </element>
    <element key="filename">
      <value>hello-world.jpg</value>
    </element>
    <element key="format">
      <value>image/jpeg</value>
    </element>
  </element-set>
  <element-set namespace="http://purl.org/dc/elements/1.1/">
    <element key="creator">
      <value>lenya</value>
    </element>
    <element key="title">
      <value>Hello World</value>
    </element>
    <element key="date">
      <value>2006-07-20 22:44:37</value>
    </element>
    <element key="language">
      <value>en</value>
    </element>
  </element-set>
  <element-set namespace="http://apache.org/lenya/metadata/document/1.0">
    <element key="extension">
      <value>jpg</value>
    </element>
    <element key="resourceType">
      <value>resource</value>
    </element>
    <element key="contentType">
      <value>xml</value>
    </element>
  </element-set>
</metadata>
```

6. Implementation

Like nearly all new modules/functionality the meta data usecases are following the new **fallback** concept. Meaning you are using the core contracts as long you are **not** overriding them with your own implementation. To override a core implementation you just need to place your custom implementation to the right path in you pub and lenya will try to pick it up from there.

6.1. Create meta data

Upon creation of a document a set of sample meta data is presented in the creation form. This values are partially filled in by the user (subject, desciption, etc.) and partly by the system (creator, creation data). This is done with the `site.create` usecase (lenya.usecase=site.create).

To tell lenya that you want as well create a set of custom meta data, you need to extend the usecase handler and modify your implementation of the `create.jx` form.

Custom implementation of create.jx

An **example** of an implementation can be found in `{$default-pub}/lenya/usecases/site/create.jx`. Just change it and see what comes out. BTW if you need it in your custom pub just mind the path. ;-)

6.2. Display/modify meta data

The display of meta data is handled by the usecase `tab.meta`. All editable meta data are presented by the form.