Create a new aspect (Manakin)

(This is a work in progress. As I learn more I add it.)

A Manakin aspect contributes a self-contained set of features to some set of the pages generated by XMLUI. An aspect is embodied in a Cocoon pipeline. Its implementation consists of a sitemap, and zero or more classes to implement unique components used in the sitemap. (It is possible that all of the components you need are already provided by Cocoon or Manakin. Some aspects, for example, require only XSL transforms, an an XSL transformer is already implemented for you.)

You need a working knowledge of Cocoon to fully understand or to create a Manakin aspect.

The Sitemap

This is a normal Cocoon sub-sitemap. List the aspect's pipelines and the components that those pipelines use.

Component elements (`transformer`, `action`, `matcher`, etc) are grouped by type. Each one gives a short name to a Java class which implements its function.

A pipeline expresses a sequence of actions carried out on the DRI document being constructed. Conceptually, the document passes through each action on its way to the end of the sitemap. Begin with a `generator` (which accepts the developing DRI document from an earlier processing stage) and end with a `serializer` (which passes the document to a later stage). Sandwiched between these are `matches`, `selectors`, `transformers`, `actions`, and the like, which act on the content of the document.

Typically a pipeline will need to choose whether it is interested in processing this particular document, and perhaps how it should treat the document based on its content. A variety of matchers are available for this purpose. Common matches are "wildcard" (which matches wildcard patterns against the application-specific portion of the URL), `HandleTypeMatcher` (which tests handles as to whether they represent communities, collections, items, bitstreams, etc.), and `HandleAuthorizedMatcher` (which tests the requesting user's authorization for types of access to the object represented by a handle). If none of your pipeline's components matches the current document, the document should be serialized as-is.

Match operations can be nested to any reasonable depth.

Eventually you reach a point at which you have discovered what you want to do to the document. `transform` stages apply transformers to the document, to add to or alter some of its content. You may find existing transformers that implement some of the operations you require, but this is the most likely place for you to identify some of your own code which augments or edits the document.

A start on a list of the transformers provided by DSpace is available by browsing the DSpace javadocs.
See the direct known subclasses of `AbstractDSpaceTransformer`.

You also have the use of the stock transformers provided by Cocoon – see the Cocoon 2.1 documentation for details.

Writing a Transformer

A transformer is a subclass of `AbstractDSpaceTransformer`.
See the javadoc for `AbstractDSpaceTransformer` for the methods you can override.

Augmenting the DRI document

In particular, you'll want to implement some of `addBody()`, `addOptions()`, `addPageMeta()`, and/or `addUserMeta()`. Each is passed an object representing the appropriate segment of the DRI document so that your code can make required additions to it. For example, to augment the body you would override `addBody`:

```
public void addBody(Body body) throws ...
{
  String foo = Bar.getFoo();

  body.addDivision("myDivision")
    .addPara("myStuff", null)
    .addContent(foo);
}
```

This would add a division named "myDivision" containing a paragraph named "myStuff", whose content is whatever was returned by `Bar.getFoo()`.

What elements you add is up to you. Keep in mind that you'll need to organize the data so that your themes can easily pick them out for placement in the final page.

Knowing what you're working with

The class `org.dspace.app.xmlui.utils.HandleUtil` is quite useful for discovering what sort of request you are fulfilling and what repository object is in focus.

```
DSpaceObject theObject = HandleUtil.obtainHandle(objectModel);
if (null != theObject)
{
  int objectType = theObject.getType();
  int objectId = theObject.getID();
}
```

If `obtainHandle` returns `null` then the current request does not refer to a specific repository object handle.

`objectModel` is an attribute of `AbstractDSpaceTransformer`, filled in by its `setup` method, and thus available to you since you are extending that class. The type of a `DSpaceObject` should be one of the small integers defined in `org.dspace.core.Constants` (ITEM, COLLECTION, COMMUNITY, etc.).

Complete Examples

Emetsger__ExampleAspects

Additional Documentation

- Manakin Developer's Guide
- DRI Schema Reference