CocoonRefDocProject

First specification for the cocoon-refdoc project of the GoogleSummerOfCode2005.

The project mentors are [BertrandDelacretaz] and Ross Gardler (who doesn't have a page here but is part of the Forrest team).

Overview

The goal of the project is to 'Generate reference documentation by extracting, indexing and assembling snippets of information from annotated java, XML, XSLT and other source code'.

The main goal is to generate reference documentation for Cocoon itself, in XML form which will be converted to HTML for web publication, either using Forrest, Daisy or a specific mechanism. The system is not specific to Cocoon code, however, and could be used to any code, but the focus is on Cocoon for the first phase.

Current situation and goals

Although there are quite a lot of documents, the Cocoon documentation is not well organized and information is hard to find.

Realizing that the many existing examples are often the best source of precise information, this project aims to extract the reference information directly from the source code, by annotating the code and extracting snippets of information from the various files (java code, sitemaps, configuration files, XML input files, XSLT stylesheets, etc) which compose a Cocoon example or application.

This will be applied mainly to the Cocoon sitemap components (Generators, Transformers, Serializers, etc.), and to the many examples provided with Cocoon.

Use-cases

Annotating XML code

Here's an example of annotating a sitemap.

Markers in XML comments are used to mark sections of code, and to give them "doktor attributes" which are used to organize and sort the code snippets (doktor is the current codename for the DOKumentation extracTOR, the component which will extract and organize the snippets).

```
<!-- @doktor-start type:pipeline key:FirstExample -->
<map:match pattern="dummy-sitemap">
    <map:read src="index.html"/>
</map:match>

<map:match pattern="*/dummy/**">
  <map:mount check-reload="yes" src="{1}/" uri-prefix="{1}"/>
</map:match>
<!-- @doktor-end -->
```

In that case, the snippet includes the next two <map:match> elements, and will be attached to the FirstExample document that the system will generate by aggregating all snippets having the same key.

```
<!-- @doktor-element type:pipeline -->
<map:match pattern="snippets/**">
  ...
</map:match>
```

The '@doktor-element' marker selects the following element only as a snippet.

By default, the 'key:' property is the last one declared in the same document, so this snippet would also be attached to the FirstExample document.

Annotating java code

Here's an example of annotated java code:

```
/** This class shows example of doktor annotations, which the
 *  doktor extractor uses to generate documentation snippets.
 *
 *  key: sets the current snippet key
 *  @doktor key:firstExample
 *
 *  @doktor-start
 *      Everything between -start and -end annotations is collected
 *      as part of the snippet content.
 *  @doktor-end
 *
 *  @doktor-start type:warning
 *      Here a name-value pair after doktor-start is used to set the
 *      type of the annotation. This will be published as a warning
 *  @doktor-end
  */
public class AnnotatedClass {

    /** @doktor-start type:code
     *  To include code in a doktor annotation, surround it with
     *  -start and -end doktor tags
     * */
    public void someFunction() {
        for(int i=0; i < 10; i++) {
            final int nothingUseful = i;
        }
    }
    /** @doktor-end */
}
```

In the first version, the snippet extractor won't know about the java syntax, as it is not meant to replace javadoc, but more to provide an overview of the application and data flow.

Annotating other source code

Other text source files (.txt, .properties, etc.) are annotated as in the java code example, and the same parser is used to extract them.

Extracting and indexing documentation snippets

To extract the documents snippets, a viewer is created with the usual Cocoon tools (DirectoryGenerator, XSLT, probably a custom Java transformer) to navigate the source code directories and access the documentation snippets in XML.

Using XSLT transforms, the XML produced as adapted to be easily indexable by the standard Lucene crawler provided by the Lucene block of Cocoon.

Then, the Lucene block crawler uses the viewer to collect and index all documentation snippets, including named fields like 'key' which are used later to assemble the snippets into documents.

Accessing a single document

To access a single document, the Lucene index is queried for a specific *key*, the snippets are sorted according to their *type*, and an XML document is generated, containing the aggregation of all snippets having the given key.

Publishing

The usual Cocoon/Forrest/Daisy mechanisms can then be used for publishing the XML documents generated in the above use case.

For the Summer of Code project, the goal is to create a publication, which is ready for inclusion in the Cocoon website.