ValidationTransformer

Note that a validation transformers block will be available in Cocoon 2.1.8, see http://svn.apache.org/repos/asf/cocoon/branches/BRANCH_2_1_X/src/blocks/validation/

General

In this page I present two Transformers that ease the validation of XML documents:

- `ValidatorTransformer`
- `ValidatorReportTransformer`

that we will cover in more detail below.

ValidatorTransformer

Description

This transformer can be inserted in any stage of an XML pipeline, in order to validate its contents. This can be a powerful tool for debugging and conformance testing. All detected errors and warnings are dumped to the logs. The transformer uses directly SAX events in order to validate. It operates in a non-intrusive way, it can be commented out and the pipeline will work in the same fashion.

Declaration

In order to use it first you have to declare the transformer as a sitemap component:

```
<map:transformers>
        ....
        <map:transformer logger='transf.validate' name='validate' src='fcc.ima.cocoon.ValidatorTransformer'/>
        ....
</map:transformers>
```

Use

Here is an excerpt of a pipeline that uses this transformer, to check that we are generating valid HTML.

```
<map:resource name='html'>
        <map:generate src='my-file.xml'/>
        <map:transform type='my-transform'/>
        <map:transform type='validate' src='validation/xhtml1-transitional.dtd'>
                <map:parameter name='active' value='true'/>
        </map:transform>
        <map:serialize type='html'/>
</map:resource>
```

As you can see in the `src` attribute we pass a URL that points to a DTD, Relax NG Schema or XML Schema. Any valid protocol is valid, even the `cocoon:` one. Relax NG Schemas are much more powerful and easy to generate.

Configuration

Normally, we will want to deactivate the checking in production sites, in order to gain speed, so I have created a parameter called `active` that allows easily deactivating it.

You can achieve this also globally:

```
<map:transformers>
        ....
        <map:transformer logger='transf.validate' name='validate' src='fcc.ima.cocoon.ValidatorTransformer'>
                <active>false</active>
                <defaultFactory>com.thaiopensource.relaxng.jarv.VerifierFactoryImpl</defaultFactory>
        </map:transformer>
        ....
</map:transformers>
```

The `<factory>` element specifies the default validator factory. It can be overriden with the parameter of the same name. Its default value is `com.sun.msv.verifier.jarv.TheFactoryImpl`. This means to use the `Sun Multi-Schema XML Validator`. See later under the pre-requisites section.

The following boolean parameters control if processing must be aborted in case of a warning, an error or a fatal error:

- stop-for-errors, default value `false`.
- stop-for-warnings, default value `false`.
- stop-for-fatal, default value `true`.

And here is an example of its use:

```
<map:transform type="validation" src="my-schema.rng">
        <map:parameter name="stop-for-warnings" value="false"/>
        <map:parameter name="stop-for-errors" value="false"/>
        <map:parameter name="stop-for-fatal" value="false"/>
</map:transform>
```

If you'd like to change the default values for all invocations of the transformer, you can also use configuration parameters at component declaration:

```
<map:transformers>
        ....
        <map:transformer logger='transf.validate' name='validate' src='fcc.ima.cocoon.ValidatorTransformer'>
                <stop-for-warnings>false</stop-for-warnings>
                <stop-for-errors>true</stop-for-errors>
                <stop-for-fatal>true</stop-for-fatal>
        </map:transformer>
        ....
</map:transformers>
```

ValidatorReportTransformer

Description

It is very similar to the `ValidatorTransformer`. Instead of letting pass through all the SAX events and generate the report on the logs, it generates the report directly as an output XML stream. This report informs about the conformance of the XML input data with any schema, DTD supplied to the transformer. It can be useful for debugging or quality control purposes

It generates an output XML document whose root element is called `report` and may have child elements `fatal-error`, `error`, `warning`. This report can easily be transformed to a HTML report through a simple XSTL sheet.

```
<report>
       <error>Detected error 1</error>
       <warning>Detected warning 1</warning>
       <error>Detected error 2</error>
       <fatal-error>Detected fatal error 1</fatal-error>
       <warning>Detected warning 2</warning>
</report>
```

In case the input XML document is valid an empty report will be generated.

```
<report>
</report>
```

Locating errors

If the generator provides a document locator, the line and column number are provided. This allows to easily locate the error. In any other case a pseudo-XPath syntax is used like this `rootElement/subElement[number]/subElement[number]/.....`

Example:

If an error is reported in `html/body[2]/div[5]` it means that the error appears in a div element, that is the fifth child element (not the fifth `<div>` element!) of a `<body>` element. This `<body>` element is the second child of the `<html>` root tag.

Declaration

In order to use it first you have to declare the transformer as a sitemap component:

```
<map:transformers>
        ....
        <map:transformer name='report-validate' src='fcc.ima.cocoon.ValidatorReportTransformer'/>
        ....
</map:transformers>
```

Use

Here is an excerpt of a pipeline that uses this transformer, to check that we are generating valid HTML.

```
<map:resource name='html'>
        <map:generate src='my-file.xml'/>
        <map:transform type='my-transform'/>
        <map:transform type='report-validate' src='validation/xhtml1-transitional.dtd'>
                <map:parameter name='active' value='true'/>
        </map:transform>
        <map:transform src='report-to-html.xsl'/>
        <map:serialize type='html'/>
</map:resource>
```

As you can see in the `src` attribute we pass a URL that points to a DTD, Relax NG Schema or XML Schema. Any valid protocol is valid, even the `cocoon:` one.

Configuration

Nothing is needed.

Pre-requisites

Both transformers use a product called Sun Multi-Schema XML Validator. It has been tested with version `20030225`. You need to place the file `msv.jar` and other ones this product needs like `xsdlib.jar`, `relaxngDatatype.jar` and `isorelax.jar` in `WEB-INF/lib` of your `Cocoon` installation. All these JARs are included in `msv.zip`. The Multi-Schema XML Validator page appears to have moved to here.

An alternative to `MSV` is Jing and isorelax. I have tested both and personally prefer `MSV` because it gives clearer validation messages.

In order to use Jing, you have to specify this verification factory: `com.thaiopensource.relaxng.jarv.VerifierFactoryImpl`.

Note that MSV cannot currently be redistributed with Cocoon due to a license incompatiblity (nuclear restriction clause), see

- http://marc.theaimsgroup.com/?t=108557516400003&r=1&w=2
- http://marc.theaimsgroup.com/?t=111720853300003&r=1&w=2

Contact the author

I can be reached through `craquerpro (at) yahoo.es`. Comments are welcome.

Source code

See attached files.

History of changes

2005-8-26: improved the reporting of error messages. Default factory is configurable.

MiXMLPipe.java

AbstractValidatorTransformer.java

Elem.java

LocationTracker.java

ValidatorReportTransformer.java

ValidatorTransformer.java