Digester FAQ

The Apache Commons Digester Frequently Asked Questions Page

Introduction

This page is intended to gather answers to questions that are regularly asked on the digester email lists.

If you discovered something about the Digester that you think other people may find useful, please edit this page to add that information.

Is there documentation on using Digester?

Yep. As the website says:

> User documentation is available as package descriptions within the JavaDoc API documents.

This means clicking on the javadoc link, then clicking on the "org.apache.commons.digester" package name in the main window (not the package nav bar) then scrolling down below the table of interfaces and classes.

Are there any examples of how to use Digester?

Definitely.

Download the source distribution and look in src/examples. Don't forget to read the readme.txt files!

Alternatively, you can browse the examples directly from the subversion source control system:
http://svn.apache.org/repos/asf/commons/proper/digester/trunk/src/examples/

Should I use the xmlrules module or the java API to configure rules?

The rules that Digester applies to input XML can be configured from java code, by instantiating various rule object types, and adding them to the digester. Alternatively, Digester can be asked to read the rules from an external configuration file (itself in xml format) that is called an xmlrules file.

It is a matter of debate as to which is superior. See the XmlRules Info Page for more information.

Why doesn't Digester throw SAXParseException when processing bad XML?

The Digester class extends org.xml.sax.helpers.DefaultHandler, and the default implementation it inherits from there is to completely ignore errors reported during parsing.

You need to implement your own subclass of org.xml.sax.ErrorHandler, and ensure the class looks something like the following:

```
public class MyErrorHandler implements org.xml.sax.ErrorHandler {
  void warning(SAXParseException ex) {
     // stop processing on warnings
    throw ex;
  }

  void error(SAXParseException ex) {
     // stop processing on errors
    throw ex;
  }

  void fatalError(SAXParseException ex) {
     // stop processing on fatal errors
    throw ex;
  }
}
```

You then need to tell Digester to use your errorhandler class, as follows:

```
  MyErrorHandler eh = new MyErrorHandler();
  digester.setErrorHandler(eh);
```

Can I reuse a Digester instance to parse more than one xml document?

Well, some people do. However this is not currently recommended.

The problems with reusing a Digester instance aren't related to multi-thread synchronization. The problem is that the Digester class has quite a few member variables whose values "evolve" as SAX events are received during a parse.

When reusing the Digester instance, you therefore need to ensure that all those members are reset back to their initial states before the second parse begins.

The "clear()" method makes a stab at resetting these, but it is actually rather a difficult problem.

If you are determined to reuse Digester instances, then at the least you should call the clear() method before each parse, and *must* call it if the Digester parse terminates due to an exception during a parse. This method is called automatically when the SAX event "endDocument" is received; however when a parse fails part-way through, this event is never generated so the clear() method is never invoked.

A much cleaner solution is simply to create a new Digester instance for each XML document parsed. If you are concerned about performance then you might want to create a RulesBase object, add rule instances to it, then reuse that object but create a separate Digester object each time. Re-using rule instances can be unsafe, however. The majority of Rule classes are stateless, but some are not. Of the ones that do retain state, they are *probably* safe to reuse if the parse was successful. When an error occurred during a digester parse, however, these rules may well have internal state that will cause them to work incorrectly on a future parse. Sorry, there is no list indicating which Rule classes are stateless and which are not; you'll have to look at the source code for each Rule class to figure that out.

You can also create a parser object and pass it to the digester constructor, to save having to instantiate a new parser object when new Digester objects are instantiated. You will of course need to call any necessary methods on the parser to reset it before each parse.

How do I enable debugging output for Digester?

Digester uses the commons-logging package. Commons-logging can be configured in many different ways, but the simplest mechanism is to run Java with the following options:

```
java -Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.SimpleLog \
     -Dorg.apache.commons.logging.simplelog.defaultlog=debug \
      your.main.class.name
```

An alternative is to ensure that log4j.jar is in the classpath, then create a log4j.xml or log4j.properties file with the appropriate logging categories enabled, and place that configuration file in a directory which is in the classpath. See http://logging.apache.org/log4j/docs for more information on log4j.

For more options, see the commons-logging project documentation.

How do I get CallMethodRule to fire before SetNextRule?

Often, when using SetNextRule to add a newly created object to a parent object, it is necessary to ensure that the child object's members are all fully initialised from the input xml before calling the parent object's add method. However the intuitive approach does not work.

Given these rules:

```
digester.addObjectCreate("parent", Parent.class);
digester.addObjectCreate("parent/child", Child.class);
digester.addCallMethod("parent/child", "setName", 1);
digester.addCallParam("parent/child/name", 0);
digester.addSetNext("parent/child", "addChild");
```

and the xml

```
<parent>
  <child>
    <name>child1</name>
  </child>
</parent>
```

the sequence of operations is:

- create Parent instance
- create Child instance
- call parent.addChild(child)
- call child.setName("child1")

The reason is a side-effect of the stack-based nature of Digester.

Rule objects can perform their work when the matching start tag is found, or when the matching end tag is found. The CallMethodRule has no choice: it has to fire when the end tag is found, because it has to wait until all its parameters are available. And SetNextRule also performs its operations in the end method (ie at the end tag).

However end methods for rules are fired in *reverse* order from the order they are added to a digester instance, in order to ensure that if the rule manipulates a stack then proper stack LIFO ordering is preserved. Because the CallMethodRule was added before the SetNextRule, its begin method fires before the SetNextRule begin method but the CallMethodRule end method (where the child object's setName method is called) fires after the SetNextRule end method (where the addChild method is called).

The solution is simple, though not entirely intuitive; put the SetNextRule first, resulting in its end method being called last. The following code results in the child's name being fully set before the addChild method is called:

```
  digester.addObjectCreate("parent", Parent.class);
  digester.addObjectCreate("parent/child", Child.class);
  digester.addSetNext("parent/child", "addChild");
  digester.addCallMethod("parent/child", "setName", 1);
  digester.addCallParam("parent/child/name", 0);
```

Note that this problem does *not* occur when using the SetPropertiesRule. This rule does all its work when the matching start tag is encountered (because it only requires the attributes which are all present on the start tag) and so does not experience the "reverse call" order that applies to the "end" method of Rules.

How do I add literal elements to a List object?

Assume the following input:

```
<list>
  <entry>value1</entry>
  <entry>value2</entry>
  <entry>value3</entry>
  <entry>value4</entry>
</list>
```

The following rules will generate and populate an ArrayList with the literal strings:

```
  digester.addObjectCreate("list", ArrayList.class);
  digester.addCallMethod("list/entry", "add", 1);
  digester.addCallParam("list/entry", 0);
```

How do I add object elements to a List object?

Assume the following input:

```
<list>
  <entry>value1</entry>
  <entry>value2</entry>
  <entry>value3</entry>
  <entry>value4</entry>
</list>
```

The following rules will generate and populate an ArrayList with an object representing each entry. In this case, a StringBuffer is generated to represent each entry, but of course the entry xml could be more complex, and mapped into some appropriate more complex object:

```
  digester.addObjectCreate("list", ArrayList.class);

  // rules to create/initialise an object for each entry
  digester.addObjectCreate("list/entry", StringBuffer.class);
  digester.addCallMethod("list/entry", "append", 1);
  digester.addCallParam("list/entry", 0);

  // rule to add the Entry object (top object on the stack)
  // to the list object (top-1 object on the stack)
  digester.addSetNext("list/entry", "add");
```

How do I add literal elements to a Map object?

Assume the following input:

```
<map>
  <entry key='key1'>value1</entry>
  <entry key='key2'>value2</entry>
  <entry key='key3'>value3</entry>
  <entry key='key4'>value4</entry>
</map>
```

The following rules will generate and populate a HashMap:

```
  digester.addObjectCreate("map", HashMap.class);

  // call the put method on the top object on the digester stack
  // passing the key attribute as the 0th parameter
  // and the element body text as the 1th parameter..
  digester.addCallMethod("map/entry", "put", 2);
  digester.addCallParam("map/entry", 0, "key");
  digester.addCallParam("map/entry", 1);
```

How do I add object elements to a Map object?

Assume the following input:

```
<map>
  <entry key='key1'>value1</entry>
  <entry key='key2'>value2</entry>
  <entry key='key3'>value3</entry>
  <entry key='key4'>value4</entry>
</map>
```

The following rules will generate and populate a HashMap with an object representing each entry. In this case, a StringBuffer is generated to represent each entry, but of course the entry xml could be more complex, and mapped into some appropriate more complex object:

```
  digester.addObjectCreate("map", HashMap.class);

  // create an object to represent the entry and initialise it
  digester.addObjectCreate("map/entry", StringBuffer.class);
  digester.addCallMethod("map/entry", "append", 1);
  digester.addCallParam("map/entry", 0);

  // call the put method on the second-to-top object on the digester stack
  // passing the key attribute as the 0th parameter
  // and the top element on the stack as the 1th parameter..
  //
  // Note that we can't use SetNextRule as that only ever passes one parameter.
  // Note also that this variant of CallMethodRule doesn't have a convenience
  // factory method on the Digester class, so we need to create it directly.
  Rule r = new CallMethodRule(1, "put", 2);
  digester.addRule("map/entry", r);
  digester.addCallParam("map/entry", 0, "key");
  digester.addCallParam("map/entry", 1, true);
```

Why does Digester read the DTD even when validation is disabled?

A DTD can affect the meaning of a document, so an XML parser still needs to read it even when validation is disabled. Note that this is a fundamental feature of XML parsing, and nothing to do with Digester really.

For example, the DTD may define default values for xml attributes:

```
  <!ATTLIST some-element some-attribute CDATA "some-default-value">
```

When the DTD is present, and the user specifies

```
<some-element/>
```

the xml parser will report that the element has an attribute "some-attribute" with value "some-default-value". But if the DTD is ignored (not read) then the element would be reported as having no attributes.

The DTD can also define entities that can be referenced from the document. Without the DTD, these won't work.

How can I use a local version of a DTD referenced from an xml document?

When an xml document contains `<!DOCTYPE rootelement PUBLIC xxxx SYSTEM yyyy>` the xml parser used by Digester will try to load file yyyy in order to process the DTD. As noted in the previous FAQ entry, this occurs even when validation is disabled.

SYSTEM is a totally non-portable identifier. Usually it is a reference to a local file that is really only useful on the same machine the document was created on. Even when it is an http reference, it is not really wise for the receiver to download the specified file each time the document needs to be parsed (if it's accessable at all).

PUBLIC is a portable identifier that is essentially a key used to look up the real location of the corresponding resource. An application receiving a document from a remote source is expected to register local copies of the relevant document by public id, so the lookup returns a local copy. This solves the problem of passing XML documents between host machines.

In order to support this, method Digester.register(String publicId, String entityURL) can be used to specify what local file (or http url) should be read instead. Digester acts as an EntityResolver for the XML parser it creates, and uses the registered mappings in the EntityResolver.resolveEntity method. This mapping applies to all "external entities" referenced by the xml document being parsed, not just the DTD (though xml documents don't typically use external entities other than the DTD).

Note that this method takes a *PUBLIC* id only, not a *SYSTEM* id. A document that is meant to be used across machines which omits the PUBLIC identifier is broken.

If you do have to deal with a broken XML document that only has a SYSTEM id and no PUBLIC id then you will need to create an EntityResolver and pass it to the Digester.setEntityResolver method. If you really do want to ignore the DTD, you can roll your own EntityResolver class in about 10 lines; it just needs to return an empty stream. Before doing this, however, re-read the FAQ entry describing why the DTD is read even when validation is disabled.

An alternative to writing your own EntityResolver is to use a real EntityResolver such as the one available from:
http://xml.apache.org/commons/components/resolver/index.html

More information on EntityResolver behaviour can be found here:
http://xml.apache.org/commons/components/resolver/resolver-article.html

How can I validate the input document against a schema?

Unfortunately, validation of xml documents against schemas is not currently standardised, ie the procedure for configuring this is parser-specific.

Some digester maintainers feel that Digester should provide the ability to turn on document validation, and therefore the following methods have been defined on the Digester class:

```
Digester.setValidating
Digester.setSchema
Digester.setSchemaLanguage
```

If you get validation working using these, then please document the procedure here. In particular, you may wish to look at the source for the ParserFeatureSetterFactory and the classes in the org.apache.commons.digester.parser package.

However other digester maintainers feel that as this functionality is complex to set up and not currently portable the best approach is for users to configure the parser first, then pass the parser to the Digester object. This makes it possible for users to write test code to get the document reading and validating working first without involving digester. This approach will result in something like the following:

```
SAXParserFactory f = SAXParserFactory.newInstance();
f.setValidating(true);
SAXParser p = f.newSAXParser();
p.setErrorHandler(new MyErrorHandler());
// do parser-specific stuff to set up schema validation here

// here you can test whether you have got the schema validation correctly set up by
// just trying to parse the document. once that is working correctly, add the code
// below to hook a digester instance up to that parser...

Digester d = new Digester(p);
// add digester rules here

// tell the explicitly created parser to parse the input, using the
// digester object as the content handler.
p.setContentHandler(d);
p.parse(inputDocument);
```

How do I match elements based on attribute values?

Occasionally, people ask how they can fire a rule for an element based on the value of an attribute, eg

```
<item value="enabled">
```

but not for

```
<item value="disabled">
```

There is no simple way to do this with Digester; the built-in rule-matching engines only provide the ability to match on element name. There is no support available for XPath expressions like "item[@value='enabled']".

It might be possible to create a custom "filtering" rule that has a child rule, and fires that child rule only when the appropriate conditions are set. There are no examples of such a solution, however.

A possible solution is to perform an XSLT transform to filter out or rename elements that match certain conditions before using Digester to process the document.

SAX "filters" may also be useful; see class org.xml.sax.XMLFilter in the standard java javadocs. Transforming the name of matching elements should then allow rules to match based on the element name only.

If you are aware of other solutions to this issue, please add that information here.

How do I get some xml nested within a tag as a literal string?

It is frequently asked how some XML (esp. XHTML) nested within a document can be extracted as a string, eg to extract the contents of the "body" tag below as a string:

```
 <article>
   <title>An article about something</title>
   <body>
     <p>Some xhtml data</p>
     <p>Some more xhtml data</p>
   </body>
  </article>
```

If you can modify the above to wrap the desired text as a CDATA section then things are easy; Digester will simply treat that CDATA block as a single string:

```
<article>
  <title>An article about something</title>
  <body>
    <![CDATA[
    <p>Some xhtml data</p>
    <p>Some more xhtml data</p>
    ]]>
  </body>
 </article>
```

If this can't be done then you need to use a NodeCreateRule to create a DOM node representing the body tag and its children, then serialise that DOM node back to text.

Remember that Digester is just a layer on top of a standard XML parser, and standard XML parsers have no option to just stop parsing input at a specific element - unless it knows that the contents of that element is a block of characters (CDATA).

How do I get some HTML (or other non-xml data) nested within a tag as a literal string?

If you have something like:

```
<article>
  <title>An article about something</title>
  <body>
    Some html (not XHTML) data here
    <br>
    And some more text.
  </body>
 </article>
```

then this simply 'cannot' be processed by digester. Digester is a layer on top of a standard XML parser, and as this is not valid XML the underlying parser will not allow it.

Your best option is to wrap the non-xml content in a CDATA section (see the preceding FAQ entry). If you absolutely cannot change the input format (despite it not being valid XML at all) then you may be able to use something like the cybernecko HTML-Parser library (which converts HTML into XHTML) to first pre-process the data.