Anteater tags

by Ovidiu Predescu, Jeff Turner

1. Anteater tags

Since Anteater is based on Ant, understanding of the latter is helpful in understanding how Anteater works. Anteater extends Ant by supplying its own set of tasks, which have no equivalent in Ant.

1.1. Tag overview

Anteater extends Ant with a number of new tags: HTTP "action" tasks, "matcher" tasks for checking returned content, plus structural, configuration and metadata tags.

1.1.1. Action tasks

FIXME (JT):

Need to get across the idea that httpRequest is a noun, with the property that it matches certain criteria. Nice declarative stuff.

These are tasks that perform some testing operation. httpRequest and soapRequest issue HTTP requests, and invoke *matcher* tasks to perform matching (validating) on the HTTP response they get back. The listener task does the opposite. It waits for a HTTP request, invokes its matchers to perform matching on it, and then sends back a HTTP response. fileRequest is the same as httpRequest, but it tests files on the local filesystem, making it useful for prototyping tests.

1.1.2. Matcher (validator) tasks

These tasks check that the result of test operations "matches" some criterion. HTTP-related matchers include:

parameter
Checks for a HTTP parameter, eg in a query string (?foo=bar)
header
Checks for a HTTP header, like a Content-Type.
responseCode
Checks for a response code, eg 200 (OK)
method
Checks for a HTTP method (GET or POST).

These HTTP matchers also double as property setters, and when arranged with match tags, allow basic flow control.

FIXME (JT):
Add examples of this

The general content-checking matchers are:

regexp
Check HTTP body with a regular expression
contentEquals
Check HTTP body for specific contents
image
Checks if the HTTP body contains an image of specified type

There are also some XML-specific matchers:

xpath
Check that an XPath expression is true in the returned XML
relaxng
Validate returned XML against a Relax NG schema

1.1.3. Structural, configuration and metadata objects

Everything that isn't an action task or a matcher is lumped in this category.

Configuration objects include: loggers (log testing actions), sessions (client-side statefulness), and namespaces (for namespace-aware matchers). Loggers are usually used in conjunction with groups. Sessions are used in advanced scenarios when you want to override the default session.

The main structural task is match, which lets one group matcher tests. The group element is the core of the Grouping system, which becomes important when structuring larger scripts. In the future, there will be test metadata objects like testdescription, specref

1.2. How it works

Each action task can contain one or more match tasks. An action task corresponds to a particular message you expect to receive from a Web service or client.

The match tasks associated with an action task describe the HTTP message you expect to receive. A match task is considered to be successful if all the associated matcher tests succeed. If one test fails, the match task that contains it is fails.

If multiple match tasks are specified for an action task, each match task is executed in turn, until one of them succeeds, at which point the matching process stops. If a match task succeeds, the action task that contains it succeeds. If none of the match tasks succeed, the action task is considered to fail, and it will be reported as such. In other words, if we had:

```
<httpRequest path="/foo.xml">
  <match>
    <A ../>
    <B ../>
  </match>
  <match>
    <C ../>
    <D ../>
    <E ../>
</match>
```

Then the httpRequest task would succeed if A and B succeeded, or if C, D and E all succeeded. In boolean logic, this is (A and B) or (C and D and E).

Anteater implements a 'shortcut boolean evaluation' policy. As soon as a <match> succeeds, the action task concludes. Likewise, as soon as a matcher test fails (eg `<C ../>`, none of the others (`D, E`) are processed.

If no match tasks are specified for an action task, the action task is considered successful as soon as it finishes. An action task that sends an HTTP request is considered finished as soon as the response is read from the Web server. An action task that listens for an incoming request is considered successful as soon as a request is received on the listening URL and the response is sent back to the client.

Let's consider the following simple example:

```
<target name="simple">
  <soapRequest
    description="Post a simple SOAP request">
    href="http://services.xmethods.net:80/soap"
```

```
    content="test/requests/get-quote">
    <namespace prefix="soap" uri="http://schemas.xmlsoap.org/soap/envelope/"/>
    <namespace prefix="n" uri="urn:xmethods-delayed-quotes"/>
    <match>
      <responseCode value="200"/>
      <xpath select="/soap:Envelope/soap:Body/n:getQuoteResponse/Result"/>
    </match>
  </soapRequest>
</target>
```

In this example we can identify the following Anteater tasks:

- soapRequest: the action task
- namespace: namespace declaration
- match: the match task
- responseCode and xpath: the test tasks