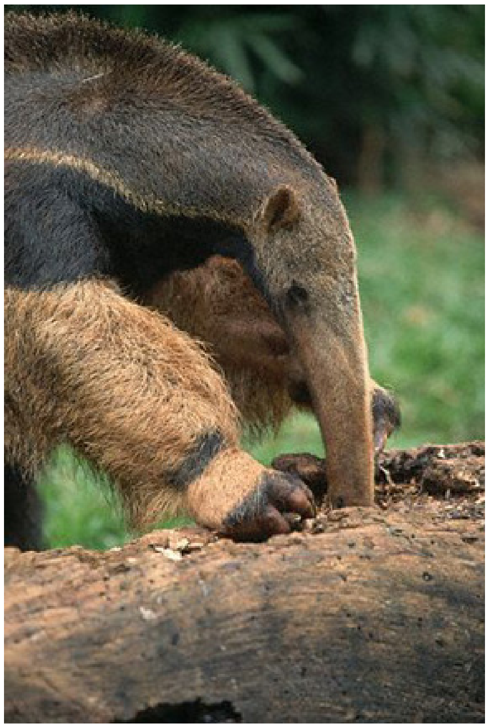
Anteater User Manual

for version 0.9.16
April 2003



by Ovidiu Predescu and Jeff Turner

1 Introduction

Anteater is a testing framework designed around Ant , from the Apache Jakarta Project. It provides an easy way to write tests for checking the functionality of a Web application or of an XML Web service.

The type of tests you can write using Anteater are:

- Send a HTTP/HTTPS request to a Web server. When the response comes back, test that it meets certain criteria. You can check for HTTP headers and response codes, and validate the response body with regexp, XPath, Relax NG, or contentEquals tests, plus some binary formats. New tests can be easily added.
- Listen for incoming HTTP requests at an given URL on the local machine. When a request comes on that URL, you can check its parameters and/or content, and send a response accordingly.

The ability to wait for incoming HTTP messages is something unique to Anteater, which makes it especially useful when building tests for applications that use high level SOAP-based communication, like ebXML or BizTalk. Applications written using these protocols usually receive SOAP messages, and send back a meaningless response. It is only later when they inform the client, using an HTTP request on the client, about the results of the processing. These are the so-called *asynchronous SOAP messages*, and are the heart of many high-level protocols based on SOAP or XML messages.

Here is a simple example written using Anteater:

```
<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>
```

Anteater provides XML and text logging support, and the ability to render the logs into HTML reports.

2 Getting started

To start using Anteater, download a binary package for your platform:

http://sourceforge.net/project/showfiles.php?group_id=42970

Install the Anteater package in a directory owned by you with your permissions. In the current release, Anteater requires write access to the installation directory, as its internal Tomcat servlet container needs to write various files.

You need to add Anteater's `bin/` directory in the PATH:

```
 $ PATH=/path/to/Anteater/bin:$PATH
 $ export PATH
```

To write your own scripts you need to declare in them the Anteater tasks and types. Here is a simple skeleton for an Anteater test script:

```
<?xml version="1.0"?>
<project name="Anteater-testscript" default="main">
  <taskdef resource="META-INF/Anteater.tasks"/>
  <typedef resource="META-INF/Anteater.types"/>

  <target name="mytest">
    <echo>
      Start writing Anteater tasks here
    </echo>
  </target>

  <target name="main" depends="mytest"/>
</project>
```

You can name the file however you like it. To run the test script, simply run:

```
 $ anteater -f <your-test-file> [<test target>]
```

If you have installed everything correctly, you should see something like:

```
Buildfile: simple.xml

mytest:
     [echo]
      Start writing Anteater tasks here


main:

BUILD SUCCESSFUL
Total time: 1 second
```

Here's a complete example, which checks that the Anteater website is online, and that the main page contains the word `Anteater`:

```
<?xml version="1.0"?>

<project name="Anteater-test" default="main">
  <taskdef resource="META-INF/Anteater.tasks"/>
  <typedef resource="META-INF/Anteater.types"/>

  <property name="url" value="http://aft.sourceforge.net/index.html"/>

  <target name="check-website">
    <echo>Now downloading and testing ${url}</echo>
    <httpRequest href="${url}">
      <match>
        <responseCode value="200"/>
        <header name="Content-Type" assign="contenttype"/>
        <regexp>Anteater</regexp>
      </match>
    </httpRequest>
    <echo>URL has Content-Type: ${contenttype}</echo>
  </target>

  <target name="main" depends="check-website"/>
```

```
</project>
```

Note the use of Ant Properties, which are very useful for defining reusable bits of text. More documentation on the various Ant commands is available at http://jakarta.apache.org/ant/manual

Also notice how tests like header can be used to assign properties. In general, any test element can also double as a way of assigning a value. So for example, `<contentEquals assign="filecontents"/>` will capture the contents of a file into variable `${filecontents}`.

Here is another example, this time not requiring an external site:

```
<?xml version="1.0"?>
<project name="Anteater-test" default="main" basedir=".">

  <taskdef resource="META-INF/Anteater.tasks"/>
  <typedef resource="META-INF/Anteater.types"/>

  <target name="init">
    <servletContainer port="8100"/>
  </target>

  <target name="content-check" depends="init">
    <echo>Content-check</echo>

    <parallel>
      <listener path="/good.html">
        <match>
          <method value="GET"/>
          <sendResponse href="test/responses/good.html"
            contentType="text/html"
            responseCode="301"/>
        </match>
      </listener>

      <sequential>
        <sleep seconds="1"/>
        <httpRequest path="/good.html">
          <match>
            <responseCode value="301"/>
            <contentEquals href="test/responses/good.html"/>
          </match>
        </httpRequest>
      </sequential>
    </parallel>
  </target>

  <target name="main" depends="content-check"/>

</project>
```

What is happening here? Well, notice that the first target to be run is `init`. This contains a servletContainer task, which starts up a Tomcat server on the specified port (any port above 1024 should do). Then the `content-check` target runs, and via the `parallel` task, starts a listener , as well as an httpRequest , both in parallel. You guessed it: the httpRequest is going to send a query to the listener. Anteater is acting as both a HTTP server and client.

Internally, the listener registers with the Tomcat instance to handle requests for path `/good.html`. The 1 second delay is to give Tomcat a chance to start up. Then the httpRequest task triggers, sending the request. The listener 's sendResponse task triggers, sending back a HTTP response to the httpRequest , which validates it with the contentEquals task. The test assumes there to be a HTML file in `test/responses/good.html`, relative to the `basedir` attribute of the project element.

This pattern of starting a server, registering a listener and then running a test against it is very useful for testing new scripts. For production use, you will probably want to either test against a live server (external to Anteater), or use the deploy task to deploy a webapp to the internal Tomcat server, and then test against that. The deploy task is handy for continuous integration-style, automated (cron-driven) testing.

2.1 Default properties

Anteater's default behaviour is fully configurable from the command-line or from Ant properties. For further information on how this is accomplished, check out the Configuration and Grouping sections.

For now, note that one can change the default host, port and debug level by defining the `default.host`, `default.port` and `default.debug` properties respectively, either by defining properties like this, just before your first task:

```
<group id="default">
  <property name="debug" value="1"/> <!-- 0 lowest, 10 highest -->
  <property name="host" value="mysite.com"/>
  <property name="port" value="8080"/>
</group>
```

You can also use the alternative `<property name="default.debug" value="1"/>` syntax, or from the command-line using the `-Ddefault.debug=1` argument to the `anteater` script. Please refer to the Grouping section for details on what can be customized.

2.2 Logging

Anteater has a pluggable logging system. By default, a text logger logs to the screen. The other main logger is an XML logger. You can configure Anteater to use the XML logger as well by adding the following:

```
<group id="default">
  <logger type="xml"/>
  <logger type="colour"/>
</group>
```

Then if you look in the `logs/` directory, you'll see some XML files, one per Anteater task. These XML files are in roughly the same format as those produced by Ant's <unit> task. This is so that we can reuse Ant's <junitreport> task to style them to HTML. Rather than getting your hands dirty with <unitreport>, you can invoke Anteater's pre-written reporting target with:

```
<target name="report" description="Generate a HTML report">
  <ant antfile="${anteater.report}">
    <property name="log.dir" value="${log.dir}"/>
    <property name="report.dir" value="reports"/>
  </ant>
</target>
```

The 'log.dir' property may be omitted, in which case it defaults to 'logs', the same default as the XML logger uses.

An example of what the output looks like can be found at http://aft.sourceforge.net/example_output/frames/ . A non-frames version is also available.

3 Grouping

Groups are like containers for Anteater objects, allowing reuse of definitions:

```
<group id="mygroup">
  <session/>
  <logger type="xml"/>
  <property name="host" value="localhost"/>
</group>

<!-- Each member task inherits the group's logger and session -->
<httpRequest group="mygroup" path="/a.html" .. />
<httpRequest group="mygroup" path="/b.html" .. />
```

If a task (like httpRequest) is part of a group, then it automatically uses whatever objects belong to that group.

3.1 Group Properties

Anteater tasks' behaviour is configured through properties of the group to which the task belongs. Currently recognised properties are `host`, `port`, `debug`, `timeout`, `protocol`, `haltonerror`, `enable` and `usetidy`. So if we had:

```
<group id="cocoontests">
  <property name="host" value="myhost.com"/>
  <property name="port" value="8080"/>
  <property name="debug" value="0"/>
</group>
<httpRequest group="cocoontests" ... />
<httpRequest group="cocoontests" ... />
```

Then those tasks would run against myhost.com:8080, with debug level 0, unless overridden by attributes on the httpRequest object. See the Configuration section for more details.

Group properties can also be set from outside a group:

```
<property name="cocoontests.debug" value="2"/>
```

This allows group values to be specified in properties files outside the test script, or from the command-line, eg:

```
anteater -Dcocoontests.host=localhost -Dcocoontests.debug=2 -f tests.xml
```

3.2 Group Inheritance

Since a Group object is an Anteater object, a Group can belong to another Group, either by nesting:

```
<group id="a">
  <property name="host" value="myhost.com"/>
  <group id="b"/>
</group>
```

or by the `inherits` attribute (`group` also works):

```
<group id="a">
  <property name="host" value="myhost.com"/>
</group>
<group id="b" inherits="a"/>
```

Group elements are inherited in what I hope seems a natural manner. Properties are passed through unless overridden, so `b` in the above example has host `myhost.com`. Loggers are passed through, unless any loggers are defined in the child group. Same with sessions.

3.3 The Default Group

There is an implicit `default` group, to which all tasks belong unless otherwise indicated. If the default group were written out, it would look like this:

```
<group id="default">
  <session/>
  <logger type="colour"/>
  <property name="host" value="localhost"/>
  <property name="debug" value="0"/>
  <property name="port" value="BUILTIN,8080"/>
  <property name="timeout" value="30s"/>
  <property name="protocol" value="HTTP/1.0"/>
  <property name="haltonerror" value="false"/>
  <property name="usetidy" value="false"/>
  <property name="usetidy-server" value="false"/>
  <property name="filename-format" value="true"/>
  <property name="overwrite" value="true"/>
  <property name="enable" value="true"/>

  <!-- Declare all other groups as children of 'default' here -->
  <group refid=".."/>
  ...
</group>
```

So by default, all tasks get a session, and a logger that prints to stdout, plus a bunch of properties used to configure the default Anteater behaviour.

The default group can be augmented and modified by the user, by declaring a group with id `default`. This way, we can override specific properties for all tasks:

```
<group id='default'>
  <property name="host" value="myhost.com"/>
  <property name="port" value="8080"/>
</group>
```

Or add another logger for all tasks:

```
<group id='default'>
    <logger type="xml"/>
</group>
```

All other items are inherited from the `default` defaults.

And of course the `default` group properties can be overridden at the command-line, e.g. `-Ddefault.host=myotherhost.com` or `-Ddefault.debug=10`.

3.4 Putting it all together

The purpose of grouping has been to make simple things easier, and complicated things possible. Some scenarios, from simple to complex:

3.4.1 Simple grouping

With the advent of the default group, most users need never bother with loggers, sessions, groups or properties. They just rely on the defaults, maybe occasionally overriding them, e.g. `-Ddefault.debug=5`.

3.4.2 Moderate grouping

For users for whom the defaults need modifying, that can easily be done by overriding the `default` group, and otherwise not touching the script. Want to log to XML as well as the console? Redefine the default group:

```
<group id="default">
  <logger type="minimal"/>
  <logger type="xml" todir="${log.dir}"/>
</group>
```

3.4.3 Advanced grouping

Users with somewhat large scripts, who want to break it up into sections can do so, by defining a hierarchy of groups:

```
<project name="groupdemo" default="main">
  <taskdef resource="META-INF/Anteater.tasks"/>
  <typedef resource="META-INF/Anteater.types"/>

  <group id="mytests">
    <property name="debug" value="0"/>
  </group>
  <group id="livesite" inherits="mytests">
    <property name="host" value="www.mysite.com"/>
    <logger type="xml" todir="{docs.dir}"/> <!--
    HTML report -->
  </group>
  <group id="devsite" inherits="mytests">
    <property name="host" value="www.mysite-dev.com"/>
    <property name="debug" value="1"/> <!-- devsite a bit unstable -->
    <property name="failonerror" value="true"/> <!-- Don't waste time testing whole
site -->

    <group id="devsite-brokenbit"> <!-- Very broken bit of devsite -->
      <property name="debug" value="10"/> </group>
  </group>

  <target name="main">
    <!-- Will have debug=10, host=www.mysite-dev.com, failonerror=true, and log
    to the console -->
    <httpRequest group="devsite-brokenbit" path="/broken.html"/>
  </target>
</project>
```

So we define a hierarchy of groups at the top of the script, and then use it in the subsequent tests.

4 Configuration

Anteater is configured through a set of properties. Properties belong to groups, and are named as such. Thus, `default.debug` is the *debug* property in the `default` group. If you have yet to read the Grouping section, now would be a good time to do so. The main thing to remember is that all tasks and groups belong to the `default` group, unless explicitly overridden, and thus properties in the default group will be inherited in subsequent groups unless overridden.

Currently defined default properties are:

Property name	Type	Default value	Description
default.host	String	`localhost`	Set the default host for httpRequest or soapRequest requests.
default.port	Integer	`BUILTIN,8080`	Set the default port to use. The preceding `BUILTIN` means use the servletcontainer default, and if not available, fall back to the indicated port.
default.debug	Integer	`0`	Default debug level. 0 is lowest, 10 highest. The higher the debug level, the greater the number of information being logged.
default.session	boolean	`true`	Whether the default group defines a session element.
default.timeout	Integer	`30s`	How long the client-side action tasks should wait before assuming a server is dead. If the request times out it is consider to have failed.
protocol	String	`HTTP/1.0`	HTTP protocol to connect as. If virtual hosts are used, must be `HTTP/1.1`.
default.haltonerror	boolean	`true`	Whether a failed action task halts the Anteater test script.
default.usetidy	boolean	`false`	Whether to apply JTidy on the result obtained by action tasks (like httpRequest or soapRequest) to clean up mark-up text and transform it in valid XML. To be able to apply matchers such as xpath in your tests, the content to be matched must be valid XML. If you know the response is an HTML document which is not valid XML, you should set this flag to `true` to be able to use xpath on it. One caveat with transforming HTML into XML is that the structure of your document might change. In order to correctly apply

Property name	Type	Default value	Description
			XPath on the resulting document, you need to inspect the result manually. Anteater tries to anticipate JTidy's changes by automatically setting the 'ignoreSpaces' and 'singleLine' attributes of xpath and regexp to true.
default.usetidy-server	boolean	`false`	Whether to apply JTidy on the content body received in a request by the listener task. This is very similar with the effect of `default.useTidy`, except that its action happens on the incoming requests, rather than on the result obtained by action tasks.
default.filename-format	String	`(see description)`	Flag specifying the default filename format for log files. Default value is `TEST-${groupid}_${taskname}_${url}_line-${lineno}_test-${vm-count}${_run-:run}.xml` Pretty much any property can be used, both Anteater-specific properties (e.g. the task's `description`), Group properties, and Ant <property> properties. There is one quirk in the format: variables of the form `${prefix:variable}`. These are interpreted as follows: if `${variable}` is defined, and has value `value`, then `${prefix:variable}` is replaced with 'prefix`value`'. For example, `${run_:run}` becomes 'run_1', or `${run at :date}` becomes 'run at 10/3/03'. If `variable` is undefined, the variable is replaced with ''. This hackery is primarily for the 'run' variable, which won't exist if `overwrite` is true (see below).
default.overwrite	boolean	`true`	Flag indicating whether, by default, we should overwrite log files from previous Anteater runs. If false, log files are made unique with a `runX` filename suffix, where `X` is incremented to ensure a unique filename.
default.enable	boolean	`true`	This flag indicates whether member tasks will be run or not. For example, one might classify action tasks into 'normal' and 'strict' groups, and then at runtime, choose to disable one or the other group, eg `-Dstrict.enable=false` to turn

Property name	Type	Default value	Description
			off the 'strict' group's tasks.

5 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.

5.1 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.

5.1.1 Action tasks

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.

5.1.2 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.

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

5.1.3 Structural, configuration and metadata objects

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

Configuration objects include: logger s (log testing actions), session s (client-side statefulness), and namespace s (for namespace-aware matchers). Loggers are usually used in conjunction with group s. 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

5.2 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

6 Action tasks

These tasks are used to make HTTP requests to a Web or SOAP server, or to wait for incoming HTTP requests on a local URL. A test uses these tasks to interact with the server or to receive incoming requests from Web or SOAP clients.

Care should be taken if the client and the server machines are separated by a firewall. Anteater allows firewall traversal using HTTP proxies. To setup firewall traversal, use the JVM `http.proxyHost` and `http.proxyPort` properties on the client side, for the HTTP request to work across the firewall. This can be done by setting up the `ANTEATER_OPTS` environment variable like this:

```
$ ANTEATER_OPTS='-Dhttp.proxyHost=<host> -Dhttp.proxyPort=<port>
$ export ANTEATER_OPTS
```

The current Anteater action tasks are:

- httpRequest
- soapRequest
- fileRequest
- listener

6.1 httpRequest

This task makes an HTTP request to a server, and waits for the result. Upon receiving the message, it applies the match tasks specified within it on the HTTP response obtained. If at least one match task succeeds, the request is consider successful.

Attributes

Attribute name	Type	Default value	Description
description	String		A text description of what the httpRequest achieves. This will be used in reporting.
host	String	`localhost`	The name of the host to which the request is to be sent.
port	integer	`8080`	The port number of the Web server to which the request is to be sent.
timeout	String	`30s`	The socket timeout. This determines how long Anteater waits for a non-responding HTTP service before aborting. The suffixes *`ms`*, *`s`* and *`m`* indicate milliseconds, seconds and minutes. Fractions of seconds and minutes can be used, so `1.5 s` means `1500 ms`. A zero or negative number will set an infinite timeout.
path	String		The path of the HTTP request, including any additional GET parameters.
user	String		The user name to be used with basic authentication. If no user is specified, no authentication is used.
password	String		The password to be used with basic authentication.
href	String		The URL of the Web server to which to send the request. Use either a combination of the host and port attributes, or the `href` attribute, to specify where the server is running. The `host`, `port` and `path` attributes are most commonly used when sending requests to the local host. When you send a request to the local host, you can also ignore the `host` attribute, as the default value is `localhost`.
method	String	`GET`	The HTTP request type ('GET' or 'POST'). This can also be specified as a nested method element.
content	String		An URL of a resource whose content is to be sent to the Web server. If you don't specify any protocol, the assumed protocol is `file:`. If the file doesn't start with a `/`, it is assumed to be relative to the directory where Anteater was started from. The URL of the resource can be remote, e.g. you can specify the content to be `http://www.acme.com/`. Anteater will fetch the remote resource and pass it to the Web server specified by the httpRequest element. The content can also be specified by a nested contentEquals element.
protocol	String	`HTTP/1.0`	The HTTP protocol to be used.
debug	Integer		Debug level for messages. Use a debug level greater than

Attribute name	Type	Default value	Description
			0 to get meaningful information on what's going on over the wire.
useTidy	boolean	`false`	Whether JTidy should be applied on the response obtained from the server to XML-ize the generated HTML.
followRedirects	boolean	`false`	Whether, if the HTTP request returns a 302 client-side redirect message, Anteater should automatically issue another HTTP request to the redirected-to URL and return that response. This is useful in many scenarios, eg where a login page automatically redirects a logged-in client to another page.

Elements allowed inside `httpRequest`

Element name	Description
header	HTTP header to be passed in the HTTP request sent to the server.
parameter	Specifies an additional GET or POST parameter to be passed in the HTTP request to the server.
method	Specifies whether to do a HTTP GET or POST operation. Overrides the 'method' attribute.
contentEquals	Specifies the HTTP body contents to send, probably as a HTTP POST operation. The Content-Length header will be automatically computed and added. This element overrides the 'content' attribute.
match	Specifies a set of rules to be used to match the HTTP response against. You can specify multiple match elements as children of an httpRequest or soapRequest element. If all the tests inside the `match` element succeed, the matching is considered successful and the action succeeds, otherwise the action fails.
namespace	Assigns a namespace mapping, from prefix to URI. This mapping will be used in any XML-aware testers defined in this action task.
logger	Used to specify a logger for this HTTP operation. Often this will be a reference, eg `<logger refid="mylogger"/>`. It is usually preferable to set a logger in a group , than directly on action tasks.
session	Used to specify a session for this HTTP operation. This is useful when one specifically doesn't want to use the default session, or wants to run a few HTTP operations with a completely separate session. Often this will be a reference, eg `<session refid="mylogger"/>`. It is usually preferable to set a session in a group , than directly on action tasks.
uses	If this action task requires an Anteater optional feature (eg a new XSD or RNG schema), then this tag can be used to declare the dependency. The task will check if the requirement is satisfied, and print a helpful message if it isn't.

Element name	Description
	Note that 'uses' tags from the action task's groups are also evaluated when the action task is run.

Examples

Send an HTTP `GET` request to `http://localhost:8080/`:

```
<httpRequest/>
```

Equivalent to an HTTP `GET` request to `http://localhost/`:

```
<httpRequest port="80"/>
```

Equivalent to an HTTP `POST` request to `http://localhost:8080/servlets/example` passing the content of `/etc/passwd` to the Web server:

```
<httpRequest path="/servlets/example" method="POST" content="/etc/passwd"/>
```

6.2 soapRequest

Sends a SOAP request to a SOAP server. This is equivalent to the httpRequest task, but with the following additions:

- an additional `SOAPAction` header set to `""` is passed in the request.
- the method is set to `POST`
- the `Content-type` header is set to `text/xml`.

Attributes

Attribute name	Type	Default value	Description
description	String		A text description of what the httpRequest achieves. This will be used in reporting.
host	String	`localhost`	The name of the host to which the request is to be sent.
port	integer	`8080`	The port number of the Web server to which the request is to be sent.
timeout	String	`30s`	The socket timeout. This determines how long Anteater waits for a non-responding HTTP service before aborting. The suffixes *`ms`*, *`s`* and *`m`* indicate milliseconds, seconds and minutes. Fractions of seconds and minutes can be used, so `1.5 s` means `1500 ms`. A zero or negative number will set an infinite timeout.
path	String		The path of the HTTP request, including any additional GET parameters.
user	String		The user name to be used with basic authentication. If no user is specified, no authentication is used.
password	String		The password to be used with basic authentication.

Attribute name	Type	Default value	Description
href	String		The URL of the Web server to which to send the request. Use either a combination of the host and port attributes, or the `href` attribute, to specify where the server is running. The `host`, `port` and `path` attributes are most commonly used when sending requests to the local host. When you send a request to the local host, you can also ignore the `host` attribute, as the default value is `localhost`.
method	String	`GET`	The HTTP request type ('GET' or 'POST'). This can also be specified as a nested method element.
content	String		An URL of a resource whose content is to be sent to the Web server. If you don't specify any protocol, the assumed protocol is `file:`. If the file doesn't start with a `/`, it is assumed to be relative to the directory where Anteater was started from. The URL of the resource can be remote, e.g. you can specify the content to be `http://www.acme.com/`. Anteater will fetch the remote resource and pass it to the Web server specified by the httpRequest element. The content can also be specified by a nested contentEquals element.
protocol	String	`HTTP/1.0`	The HTTP protocol to be used.
debug	Integer		Debug level for messages. Use a debug level greater than 0 to get meaningful information on what's going on over the wire.
followRedirects	boolean	`false`	Whether, if the HTTP request returns a 302 client-side redirect message, Anteater should automatically issue another HTTP request to the redirected-to URL and return that response. This is useful in many scenarios, eg where a login page automatically redirects a logged-in client to another page.
useTidy	boolean	`false`	Whether JTidy should be applied on the response obtained from the server. Since we send a SOAP request, we expect to receive back a SOAP message, so there's little reason to XML-ize this content. Set this attribute to true if the server returns broken HTML instead of SOAP.

Elements allowed inside `soapRequest`

Element name	Description
header	HTTP header to be passed in the HTTP request sent to the server.

Element name	Description
parameter	Specifies an additional GET or POST parameter to be passed in the HTTP request to the server.
method	Specifies whether to do a HTTP GET or POST operation. Overrides the 'method' attribute.
contentEquals	Specifies the HTTP body contents to send, probably as a HTTP POST operation. The Content-Length header will be automatically computed and added. This element overrides the 'content' attribute.
match	Specifies a set of rules to be used to match the HTTP response against. You can specify multiple match elements as children of an httpRequest or soapRequest element. If all the tests inside the `match` element succeed, the matching is considered successful and the action succeeds, otherwise the action fails.
namespace	Assigns a namespace mapping, from prefix to URI. This mapping will be used in any XML-aware testers defined in this action task.
logger	Used to specify a logger for this HTTP operation. Often this will be a reference, eg `<logger refid="mylogger"/>`. It is usually preferable to set a logger in a group , than directly on action tasks.
session	Used to specify a session for this HTTP operation. This is useful when one specifically doesn't want to use the default session, or wants to run a few HTTP operations with a completely separate session. Often this will be a reference, eg `<session refid="mylogger"/>`. It is usually preferable to set a session in a group , than directly on action tasks.
uses	If this action task requires an Anteater optional feature (eg a new XSD or RNG schema), then this tag can be used to declare the dependency. The task will check if the requirement is satisfied, and print a helpful message if it isn't. Note that 'uses' tags from the action task's groups are also evaluated when the action task is run.

Examples

This example demonstrates how a listener's match tasks and testers can be arranged to implement if/then/else logic.

The incoming request can be a SOAP message containing either an receipt acknowledge message or a SOAP fault message, indicating an error.

```
<listener path="/receipt">
    <match method="POST">
      <xpath select="/soap:Envelope/soap:Body/receipt-ack"/>
      <xpath select="/soap:Envelope/soap:Body/response-to" assign="replyHref"/>
      <sendResponse href="${replyHref}/>
    </match>
    <match assign="failed">
      <matchBody select="/soap:Envelope/soap:Body/rfq"/>
      <matchMethod code="POST"/>
    </match>
</listener>
```

6.3 fileRequest

This task is the same as httpRequest , but tests against a local file instead of doing a HTTP request.

Attributes

Attribute name	Type	Default value	Description
description	String		A text description of what the httpRequest achieves. This will be used in reporting.
path	String		The path of the local file to test, relative to the Anteater script's `basedir`
debug	Integer		Debug level for messages. Use a debug level greater than 0 to get meaningful information on what's going on over the wire.

Elements allowed inside `fileRequest`

Element name	Description
match	Specifies a set of rules to be used to match the HTTP response against. You can specify multiple match elements as children of an httpRequest , soapRequest or fileRequest element. If all the tests inside the `match` element succeed, the matching is considered successful and the action succeeds, otherwise the action fails.
namespace	Assigns a namespace mapping, from prefix to URI. This mapping will be used in any XML-aware testers defined in this action task.
logger	Used to specify a logger for this HTTP operation. Often this will be a reference, eg `<logger refid="mylogger"/>`. It is usually preferable to set a logger in a group , than directly on action tasks.
uses	If this action task requires an Anteater optional feature (eg a new XSD or RNG schema), then this tag can be used to declare the dependency. The task will check if the requirement is satisfied, and print a helpful message if it isn't. Note that 'uses' tags from the action task's groups are also evaluated when the action task is run.

Examples

Reads a local file, `resources/responses/text.txt`, and checks that its contents (ignoring whitespace) is a certain value.

```
              <fileRequest path="resources/responses/text.txt"
                           description="tests a text file">
                <match>
                  <contentEquals ignoreSpaces="true">
```

```
                    Here is some freeform text saved with DOS linefeeds.
                  </contentEquals>
                </match>
            </fileRequest>
```

6.4 listener

In addition to sending out HTTP requests to Web and SOAP servers, Anteater has the ability to receive incoming HTTP requests. This ability is very useful when you want to implement high level SOAP and XML protocols, like ebXML or BizTalk, which make use of asynchronous SOAP messages to exchange information between parties.

Applications implementing such protocols will accept an HTTP request as a high level asynchronous request. The response to such a request is not usually meaningful. Instead, the server will later generate a reply as another HTTP request to an URL specified by the client in the original message.

In such applications, the client and server role changes depending on the phase of the conversation. To be able to test such applications, a party in such a conversation should be able to act both as a client and as a server.

The listener element tells Anteater to stop the processing of the test script, until a request at a specified URL is received. Anteater will act exactly like an HTTP or SOAP server, by listening on the local host on a specified port, waiting for a request on a given URI path you can specify.

To use the `listener` task, the servlet container within Anteater should first be started. This is done by using the servletContainer task, which allows specifying which are the ports Anteater will listen on when acting as an HTTP server.

In the next example, a request is sent to a SOAP server, and then a response is awaited on the local host at the `/receiptAck` path URI. If the name of the machine which runs this Anteater snippet is `soap.acme.com`, the remove SOAP server would then need to send an HTTP request back to `http://soap.acme.com:8080/receiptAck` for the listener task to be unblocked and the Anteater script's execution to continue:

```
<target name="test">
  <soapRequest href="http://some.remote.server/"
               content="some/file"/>

  <listener port="8080" path="/receiptAck" timeout="7200">
    <namespace prefix="soap" uri="http://schemas.xmlsoap.org/soap/envelope/"/>
    <match>
      <method value="POST"/>
      <xpath select="/soap:Envelope/soap:Body/receipt-ack"/>
      <sendResponse href="responses/response.xml"
                    contentType="text/xml"/>
    </match>
  </listener>
</target>
```

In the above example, if no request is received within 7200 seconds from the start of the listening, the listener task will fail. If a response is received within this time, the incoming request should be an HTTP `POST` request, and should contain in the body a SOAP message with a `receipt-ack` element, for the listener task to succeed. If such a request is received, a response is sent back with the content a local file, using the sendResponse task.

If there's no match task that matches the incoming request, a response may not be generated using sendResponse . In such a case, Anteater will automatically generate a `200 OK` response, with no content in the body, and the enclosing listener element fails. Sending the response ensures the client application obtains a response back, and doesn't block it indefinitely. Future versions of Anteater will allow for the customization of such responses.

For any incoming requests, for which there's no listener task waiting, the response sent back by Anteater is a `404 Not Found`.

Attributes

Attribute name	Type	Default value	Description
path	String		Defines the URI path where an HTTP request should be received on. Requests received on other URI paths will have a `404 Not Found` response returned, unless other listeners are setup for them.
port	integer		The first port defined by the `port` attribute of servlet-Container . See a description of this element for more information.
timeout	integer	`0`	How long in seconds the listener task should wait for an incoming request. If no request is received before the time expires, the listener task fails. A value of `0` specifies an indefinite timeout, so the listener task will wait forever for a request to come.
useTidy	boolean	`false`	Whether JTidy should be applied on the incoming request body, before running the matcher tests on it.

Elements allowed inside `listener`

Element name	Description
match	Match on the incoming request.
namespace	Assigns a namespace mapping, from prefix to URI. This mapping will be used in namespace-aware tasks like xpath .
logger	Used to specify a logger for this HTTP operation. Often this will be a reference, eg `<logger refid="mylogger"/>`. It is usually preferable to set a logger in a group , than directly on action tasks.
session	Used to specify a session for this HTTP operation. This is useful when one specifically doesn't want to use the default session, or wants to run a few HTTP operations with a completely separate session. Often this will be a reference, eg `<session refid="mylogger"/>`. It is usually preferable to set a session in a group , than directly on action tasks.

Examples

This example demonstrates how a listener's match tasks and testers can be arranged to implement if/then/else logic.

The incoming request can be a SOAP message containing either an receipt acknowledge message or a

SOAP fault message, indicating an error.

```
<listener path="/receipt">
    <match method="POST">
      <xpath select="/soap:Envelope/soap:Body/receipt-ack"/>
      <xpath select="/soap:Envelope/soap:Body/response-to" assign="replyHref"/>
      <sendResponse href="${replyHref}/>
    </match>
    <match assign="failed">
      <matchBody select="/soap:Envelope/soap:Body/rfq"/>
      <matchMethod code="POST"/>
    </match>
</listener>
```

7 Match task

The result of an action task is an HTTP response or request object. Anteater allows you to test various characteristics of this object using match and test tasks.

A match task groups multiple tests that need to be performed on the result of an action task. All the tests inside the match task need to be successful in order for the match task to be successful.

An action task can have multiple match tasks that can be checked against the result object. These tasks will be executed in order until one of them succeeds, after which the result of the action task is considered to be successful. If none of the tasks succeed, the action task simply fails.

Since the `match` tasks are executed in the order in which they appear inside the action element, it is important to consider the tests that have side-effects, like assigning results to Ant properties.

```
<soapRequest description="Send a SOAP request"
   href="${url}" content="filename">
  <match>
    ...
  </match>

  <match>
    ...
  </match>

  <!-- Other match tasks here -->
  ...
</soapRequest>

<!-- Other Anteater action tasks here -->
```

In the above example, after the SOAP request returns the SOAP result object, the match tasks will be executed in order. If the first match task succeeds, the matching stops here, and the soapRequest task is successful. Otherwise, the next `match` tasks are executed, until one succeeds. If none of them succeeds, the soapRequest task fails. Depending on the value of the `haltonerror` property inherited from task's group , the subsequent action tasks inside it may or may not be executed.

Attributes

Attribute name	Type	Default value	Description
assign	String		Specifies the name of a property to set if the match task's testers all succeed. In a listener task, this would be the

Attribute name	Type	Default value	Description
			match task responsible for sending the content. This allows script writers to apply conditional logic based on which match succeeded.
value	String		Specifies the value to set the `assign` property if the match task's testers all succeed.

Elements allowed inside `match` : all Test tasks .

7.1 Conditional logic

As noted in the Anteater_tags section, match tasks can be used as a form of boolean logic. Each `match` block contains a set of AND'ed tests (they must all pass), and as any `match` block's success renders the action task successful, they are effectively OR'ed together. Thus the task:

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

implements the boolean logic (A and B) or (C and D and E).

This can be exploited in various interesting ways. Here is how a listener can deliver different responses based on the value of a parameter:

```
<listener path="/foo">
  <match assign="chose123">
    <method value="GET"/>
    <parameter name="a" value="123"/>
    <sendResponse href="test/responses/good.html" contentType="text/html"/>
  </match>
  <match assign="chose456">
    <method value="GET"/>
    <parameter name="a" value="456"/>
    <sendResponse href="test/responses/bad.html" contentType="text/html"/>
  </match>
</listener>
```

Here is a larger example of how one can determine the type of server being tested, by using the 'assign' attribute on matchers:

```
<httpRequest path="/">
  <match assign="servertype" value="gnu-apache">
    <header name="Server" value="Apache/1.3.24 (Unix) Debian GNU/Linux"/>
  </match>
  <match assign="servertype" value="generic-apache">
    <header name="Server" assign="apache-version" pattern="Apache/([\d.]+)"/>
  </match>
  <match assign="servertype" value="IIS6">
    <header name="Server" value="Microsoft-IIS/6.0"/>
  </match>
  <match assign="servertype" value="generic-IIS">
```

```
    <header name="Server" pattern="IIS"/>
  </match>
  <match assign="servertype" value="unknown">
  </match>
</httpRequest>
<echo>servertype is ${servertype}</echo>
```

8 Test tasks

The object an action task generates is either an HTTP response in the case of httpRequest or soapRequest , or an HTTP request, in the case of listener . To test characteristics of such objects, test tasks are used inside an match task.

A match task contains a set of tests to be performed on an action's object, be it an HTTP response or request object. All the test tasks specified associated with a match task must succeed in order for the match task to succeed.

The test tasks are tested in the order they appear inside the `match` task. If some of the test tasks produce side-effects, like setting a global Ant property, then you should consider carefully the order in which they are executed.

8.1 Extracting values from the action result object

The value a test task checks for can be assigned to an Ant property using the `assign` attribute. Such properties can be later referred to in Anteater and normal Ant tasks.

```
<soapRequest href="${url}"
             content="test/requests/get-quote">
  <match>
    <header name="Content-Length" assign="cl"/>
    <responseCode value="200" assign="rc"/>
    <xpath select="soap:Envelope/soap:Body/n:getQuoteResponse/Result"
           assign="result"/>
    <echo>XPath-selected the value '${result}'</echo>
  </match>
</soapRequest>
```

A major difference between Anteater and Ant is that properties can be assigned values multiple times, so you can reuse the same property across the test script. Properties assigned through the `assign` attribute can also be used immediately after their definition.

8.2 header

This task is used for multiple purposes:

- to set an HTTP header when sending an HTTP or SOAP request.
- to test the value of an HTTP header in the response obtained by httpRequest or soapRequest .
- to test the value of an HTTP request header in a request received by listener .

When used to set an additional HTTP header in an HTTP request, the `header` task should be a child of the action task, either an httpRequest or a soapRequest task. In such a usage, the `header` task is not really used as a test task; we nevertheless chose to use the same name for the task to keep things simple.

If the task is used to test the value of a header in either an HTTP response or an HTTP request, it should appear as any other test task directly as a child of the match element. See the samples below for an example of how the `header` task is used.

The header element can take nested body text, which will be stripped of preceding and trailing whitespace, and used as the header value, overriding the 'value' attribute. A nested jelly element can also be specified to dynamically generate the header value.

Attributes

Attribute name	Type	Default value	Description
name	String		The name of the header to be added in the HTTP request, or the name of the header to be tested.
value	String		If the task is used for setting headers in an outgoing request, this attribute contains the value of the header. If the `header` task is used to test the value of an HTTP response or request, the presence of this attribute indicates that an exact match is expected. If the HTTP header specified by `name` doesn't have this exact value, the `header` test task will fail.
pattern	String		Sets a regular expression with which to match the specified HTTP header in a response document. This should only be used when the element is being used as a matcher. If the pattern contains a group, `(...)`, then the matched value, if any, is placed in the `assign` property.
assign	String		Meaningful only when the header task is used for testing an HTTP header. When this attribute is specified, the property named by it will contain the value of the header, upon the successful test of the header. A successful test can happen only if the HTTP header exists and, if the `value` attribute was specified, its value equals the one specified for the HTTP header. Otherwise the property will not be assigned to.

Elements allowed inside `header`: none

Examples

The following HTTP request sets the value of the `Content-type` header to `text/html`:

```
<httpRequest href="${url}">
  <header name="Content-type" value="text/html"/>
  <match>
    ...
  </match>
</httpRequest>
```

Test whether the `Content-type` header value returned by an HTTP request is set to `text/html`:

```
<httpRequest href="${url}">
  <match>
    <header name="Content-type" value="text/html"/>
  </match>
</httpRequest>
```

Assign the value of the `Content-type` header to an Anteater property:

```
<httpRequest href="${url}">
  <match>
    <header name="Content-type" assign="type"/>
    <echo>Content-type of the response is ${type}</echo>
  </match>
</httpRequest>
```

Test whether the `Content-type` header value received in an incoming HTTP request is any sort of text response, and store that type in a variable:

```
<httpRequest path="/good.html">
  <match assign="type" value="text">
    <header name="Content-Type" assign="texttype" pattern="text/(.*)"/>
  </match>
  <match assign="type" value="image">
    <header name="Content-Type" pattern="image/.*"/>
  </match>
</httpRequest>
```

8.3 method

This task is used to:

- Set the HTTP method (typically GET or POST) of a HTTP request to send to a server.
- test the HTTP method of an incoming HTTP request accepted by the listener .

Value can be specified as inline text, possibly dynamically generated.

Attributes

Attribute name	Type	Default value	Description
value	String		The value of the method to check for. If this attribute is present, the specified method will be compared against the actual method in the request: if they don't match, this test fails, otherwise it succeeds. If this attribute is not present, any HTTP method is accepted for the incoming request. This can also be specified as nested text, or generated dynamically by a nested jelly element.
assign	String		Assigns the actual value of the request's method to an Anteater property.

Elements allowed inside `method`

Element name	Description
jelly	Specify a Jelly script, which will dynamically generate a string selecting the HTTP method (GET, POST, HEAD etc).

Examples

Generates a POST request.

```
        <httpRequest path="/text.txt">
          <method>
            POST
          </method>
          <contentEquals>Posted contents</contentEquals>
          <match>
            ...
          </match>
      </httpRequest>
```

This example tests if the incoming request is a `GET` request:

```
<listener path="/abc">
  <match>
    <method value="GET"/>
    ...
  </match>
</listener>
```

Accept an incoming request, no matter what is the method, and assign the method type to the `mth` Anteater property:

```
<listener path="/abc">
  <match>
    <method assign="mth"/>
    <echo>Received a ${mth} request</echo>
  </match>
</listener>
```

8.4 parameter

As with the header task, the `parameter` task is used for multiple purposes:

- to define an additional parameter that should be passed in an outgoing HTTP request when using the httpRequest or soapRequest tasks.
- to test the value of a parameter in an incoming HTTP request accepted by the listener task.

The parameter value may be specified as (dynamically generated) nested text.

Attributes

Attribute name	Type	Default value	Description
name	String		The name of the parameter to be passed in the HTTP request when using httpRequest or soapRequest , or to be tested for in the request received by listener .

Attribute name	Type	Default value	Description
value	String		The value of the parameter to be sent in the HTTP request using httpRequest or soapRequest , or the expected value of the parameter when using the listener task.
type	String		Meaningful only when sending out parameters. Specifies how the parameter should be passed in the HTTP request, possible values being `GET` or `POST`. `GET` parameters are encoded in the HTTP URL, while `POST` parameters are passed in the body of the request as headers. Usually this attribute is not used, as Anteater will do the proper thing and pass the parameter according to the request type. However you may want to test a particular behavior of the HTTP or SOAP server, and pass parameters in a different way than they are expected. E.g. you may pass a `GET` parameter as a `POST` parameter and vice-versa.
assign	String		Set the named Anteater property to the value of the actual HTTP request parameter.

Elements allowed inside `parameter`

Element name	Description
jelly	Specify a Jelly script, which will dynamically generate the parameter value.

Examples

Set a parameter in an outgoing HTTP request:

```
<httpRequest href="${url}">
  <parameter name="a" value="123"/>
  <match>
    ...
  </match>
</httpRequest>
```

Test if the a parameter is present in the request, and assign its value to the `paramA` Anteater property:

```
<listener path="/abc">
  <match>
    <parameter name="a" assign="paramA"/>
    ...
  </match>
</listener>
```

Return a different response based on a parameter value.

```
<listener path="/abc" description="wait for an incoming request">
```

```
      <match>
        <parameter name="a" value="123"/>
        <sendResponse href="resources/responses/good.html"
          contenttype="text/html"/>
      </match>

      <match>
        <parameter name="a" value="456"/>
        <sendResponse href="resources/responses/bad.html"
          contenttype="text/html"/>
      </match>
    </listener>
```

8.5 image

This task tests if the HTTP body contains binary image data.

Only a few representative bytes are tested, so some corrupt images may slip through. The type of image my be specified either as a MIME type or as a file extension.

Attributes

Attribute name	Type	Default value	Description
mimetype	String		MIME type indicating what form we expect the data to follow. The following types are supported: • application/x-shockwave-flash • image/bmp • image/gif • image/iff • image/jpeg • image/pcx • image/png • image/psd • image/ras • image/x-portable-bitmap • image/x-portable-graymap • image/x-portable-pixmap
extension	String		Indicates the type of content we're expecting by the com-mon file extension. Useful for types like 'application/x-shockwave-flash' which is much harder to remember than 'swf'. Supported extensions are: • jpg or jpeg • gif • png

Attribute name	Type	Default value	Description
			• bmp • pcx • iff • ras • psd • swf

Elements allowed inside `image`: none

Examples

Here's a simple example of validating a gif image

```
        <httpRequest href="http://jakarta.apache.org/images/jakarta-logo.gif">
          <match>
            <image mimetype="image/gif"/>
          </match>
        </httpRequest>
```

8.6 contentEquals

This task tests if one of the following exactly matchers a specified character sequence:

- the HTTP or SOAP *`response`* received from an httpRequest or soapRequest
- the HTTP or SOAP *`request`* received using listener

The value to be matched against could be specified either inline, as part of the contentEquals element, or in an external resource, like a file. Such an external resource is indicated by using the `href` attribute. You can refer to file relative to the current directory by specifying a relative URL which doesn't start with a `/` character.

You can also choose to ignore any white spaces differences between the two values to be checked. You can do this by setting the `ignoreSpaces` attribute.

Attributes

Attribute name	Type	Default value	Description
href	String		URL to be used to obtain the resource to check the HTTP response against. This URL can be relative, in which case the resource is a file relative to the directory in which Anteater was started from. You should use either this attribute or you should specify the value of the text inline, in the contentEquals element.
ignoreSpaces	boolean	`false`	Specifies whether spaces should be ignored when check-

Attribute name	Type	Default value	Description
			ing for equality. Whitespaces in the two values compared are ignored if this attribute is set to `true`.

Elements allowed inside `contentEquals`

Element name	Description
jelly	Specify a Jelly script, which will generate the contents required from the server.

Examples

The following example shows how to specify the value to be test against inline. It also ignores any whitespace differences between the two values.

```
              <httpRequest path="/text.txt"
                useTidy="false">
                <match>
                  <contentEquals ignoreSpaces="true">
                    Here is some freeform text saved with DOS linefeeds.
                  </contentEquals>
                </match>
            </httpRequest>
```

Here is how to generate a HTTP POST body dynamically with a Jelly script, and then require that the HTTP response body contain the text 'hello world':

```
               <httpRequest path="/text.txt">
                 <method>POST</method>
                 <contentEquals>
                   <jelly script="genBody.jelly"/>
                 </contentEquals>
                 <match>
                   <contentEquals ignoreSpaces="true">
                     hello world
                   </contentEquals>
                </match>
            </httpRequest>
```

8.7 regexp

This task checks whether:

- the HTTP or SOAP response received from an httpRequest or soapRequest
- or the HTTP or SOAP request received using listener

match a regular expression pattern.

The language used for specifying the regular expression is that of Perl5. Here is a brief reminder of this language:

- \ - quote the next metacharacter

- `^` - match at the beginning of the line
- `.` - match any character except newline (but see below on how to alter this behavior)
- `|` - specifies an alternative
- `()` - specifies a group
- `[]` - a character class. Use `-` to specify ranges, or simply enumerate the characters in the set.
- `*` - match 0 or more times
- `+` - match 1 or more times
- `?` - match 1 or 0 times
- `{n}` - match exactly `n` times
- `{n,}` - match at least `n` times
- `{n,m}` - match at least `n` times, but no more than `m` times
- any other character outside the above constructs matches that character

See the Jakarta ORO package documentation for a fuller description.

Attributes

Attribute name	Type	Default value	Description
pattern	String		Specifies the regular expression pattern to be used. You must specify the pattern using either this attribute or by placing it inline, inside the regexp element.
ignoreCase	boolean	`false`	Whether the match should be performed case insensitive or not.
singleLine	boolean	`true`	This attribute indicates whether `.` in a regular expression should match newlines (`\n`). The default is true, so for example `<html>.*</html>` will match even if there are multiple newlines between the `html` tags.
ignoreSpaces	boolean	`false`	If true, whitespace in the regexp is normalized and converted to '\s+', thus making irrelevant any whitespace differences in the matched text. This is useful for matching automatically generated HTML responses where whitespace (including linefeeds) doesn't matter. If the 'usetidy' property is true, ignoreSpaces is automatically set to true unless explicitly overridden. The JTidy algorithm generally reindents the text, making this a sensible default behaviour.
assign	String		Name of the attribute which will contain, in the case of a successful match, the value matched by the paranthesised group specified using the `group` attribute. If the match is not successful, the property is not modified.
group	int	`0`	In case of a successful match, the value matched by this group number (specified using paranthesis `()`) is assigned to the `assign`property. If the match is not suc-

Attribute name	Type	Default value	Description
			cessful the property is not modified.

Elements allowed inside `regexp`

Element name	Description
jelly	Specify a Jelly script, which will dynamically generate the required regexp pattern.

Examples

This example checks if the page returned by the server is an HTML page:

```
<httpRequest useTidy="false" href="some URL">
  <match>
    <regexp><![CDATA[<html>.*</html>]]></regexp>
  </match>
</httpRequest>
```

Given a text file of the form:

```

...
struts-dev 795
struts-user 1740
taglibs-dev 342
taglibs-user 644
tomcat-dev 934
tomcat-user 2456
turbine-dev 263
turbine-jcs-dev 17
turbine-jcs-user 21
...
```

The following example extracts a single line from the file, and assigns part of it to a variable, `${tomcat-user}`

```
<httpRequest
 href="http://jakarta.apache.org/~rubys/stats/subscribers/jakarta.apache.org">
  <match>
    <regexp assign="tomcat-user" singleLine="false" group="1">tomcat-user
(.*)</regexp>
  </match>
</httpRequest>
```

8.8 responseCode

Tests the response code of an HTTP response received by httpRequest or soapRequest .

See http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html for a link of valid response codes. Common ones include `200` (OK), `301` (Moved Permanently), `404` (Not Found).

Attributes

Attribute name	Type	Default value	Description
value	int		The expected response code, eg `200`

Attribute name	Type	Default value	Description
pattern	int		A regexp pattern which the response code (or part thereof) should match. Eg `3..` for all 300-class response codes.
assign	string		Name of a property to set to the returned response code. If a regexp group was specified in `pattern`, then the group value will be set instead.

Elements allowed inside `responseCode`

Element name	Description
jelly	Specify a Jelly script, which will dynamically generate the required response code (equivalent to the 'value' attribute).

Examples

The following example shows how to test the response code of an httpRequest task. The expected response code here is 200.

```
<httpRequest href="some URL">
  <match>
    <responseCode value="200"/>
  </match>
</httpRequest>
```

Here we use a regexp pattern to test if the response code was 300-class "Redirection" response.

```
<httpRequest href="some URL">
  <match>
    <responseCode pattern="3.."/>
  </match>
</httpRequest>
```

This is useful in conjunction with a property setter:

```
<httpRequest href="some URL">
  <match assign="ok">
    <responseCode pattern="2.."/>
  </match>
  <match assign="redirected">
    <responseCode pattern="3.."/>
  </match>
  <if>
    <isset property="redirected"/>
    <then>
      <echo>We were redirected... </echo>
    </then>
  </if>
</httpRequest>
```

8.9 xpath

This task is used to test the existence of a particular element or its value. This task assumes that the body of the response or request is an XML document. Action tasks like soapRequest , and httpRequest and listener with the `useTidy` attribute set to `true`, assume the response or request body is an XML document.

Care should be taken when matching against the result returned by the httpRequest task. In most cases, you cannot use xpath to match for elements, attributes or values using XPath expressions. You can do this only when you know the HTML response you obtained is a valid XML document, like in the case of XHTML. If you do want to test for XPath expressions, then you need to run JTidy on the response; you do this using the `useTidy` attribute of the httpRequest element. JTidy can be enabled for a whole script by setting the 'default.usetidy' property to true (see the Configuration section).

To test the existence of a particular element or element, the xpath task uses XPath as the language to address parts of the XML document.

To test the existence of an element or attribute, you should set the `select` attribute to the XPath value you're interested in. The xpath task will verify the existence of that element in the XML document in the body. If in addition to the `select` attribute, you also specify the `assign` attribute, the text value of the selected element or attribute is assigned to the property named by `assign`.

If you're interested in a particular value of an element or attribute, in addition to the `select` attribute, you should specify the `value` attribute. The actual value of the element or attribute is literally compared against this value. If the `assign` attribute is also present, and the element or attribute described by `select` exists, and its value is the same with the one specified by the `value` attribute, the property named by `assign` will contain the matched value.

Attributes

Attribute name	Type	Default value	Description
select	String		The XPath string to identify an element or attribute in the body of the HTTP response or request. Mandatory.
value	String		The value expected for element or attribute identified by the `select` attribute.
pattern	String		Sets a regular expression which the string value of the 'select' XPath expression must match. If the pattern contains a group, `(...)`, then the matched value, if any, is placed in the `assign` property.
ignoreSpaces	boolean	`false`	If true, whitespace in the 'pattern' regexp is normalized and converted to '\s+', thus making irrelevant any whitespace differences in the matched text. ignoreSpaces is automatically set to true if the 'usetidy' flag is set, unless explicitly set.
group	int	`0`	Used in conjunction with the 'pattern' attribute. In case of a successful pattern match, the value matched by this group number (specified using paranthesis `()`) is assigned to the `assign`property.
assign	String		In case of a successful match, it will contain the value of the attribute or element specified by `select`.

Elements allowed inside `xpath`

Element name	Description
jelly	Specify a Jelly script, which will dynamically generate the string value of the matched xpath node (equivalent to the 'value' attribute).

Examples

Check for the existence of a particular element, and assign its value to a property. In this example the message is printed out only if there is an element with with the indicated path. If no suche element exists, the matcher fails and the `echo` task is not executed.

```
<soapRequest href="http://services.xmethods.net:80/soap"
  content="test/requests/get-quote.xml">
  <namespace prefix="soap" uri="http://schemas.xmlsoap.org/soap/envelope/"/>
  <namespace prefix="n" uri="urn:xmethods-delayed-quotes"/>
  <match>
    <xpath select="soap:Envelope/soap:Body/n:getQuoteResponse/Result"
      assign="result"/>
  </match>
</soapRequest>
<echo>XPath-selected the value '${result}'</echo>
```

This example is the same as above, but it will print the message only if the value matches exactly the one specified in the `value` attribute of `xpath`.

```
<soapRequest href="http://services.xmethods.net:80/soap"
                content="test/requests/get-quote.xml">
  <namespace prefix="soap" uri="http://schemas.xmlsoap.org/soap/envelope/"/>
  <namespace prefix="n" uri="urn:xmethods-delayed-quotes"/>
  <match>
    <xpath select="soap:Envelope/soap:Body/n:getQuoteResponse/Result"
      value="20"
      assign="result"/>
  </match>
</soapRequest>
<echo>XPath-selected the value '${result}'</echo>
```

8.10 relaxng

This task tests XML for conformance to a specified Relax NG schema. It is based on James Clark's Jing task for Ant .

Attributes

Attribute name	Type	Default value	Description
rngFile	String		Basedir-relative path to a file containing a Relax NG schema
compactSyntax	boolean	`false`	Whether or not the specified file is in Relax NG's compact syntax .
checkId	boolean	`true`	Whether to check for ID/IDREF/IDREFS compatibility

Elements allowed inside `relaxng`: none

Examples

Checks that an XSLT file is valid

```
<httpRequest group="std" path="/identity.xsl">
  <match>
    <!--
    <xpath select="xsl:stylesheet/xsl:output"/>
    -->
    <relaxng rngFile="test/xslt.rng"/>
  </match>
</httpRequest>
```

8.11 sendResponse

Sends a response to an HTTP request received by the listener task.

With this task you can specify:

- the body content of the response to be sent back, by a nested 'contentEquals' element, 'href' attribute, nested text or nested jelly task.
- the response code to be used, by a nested 'responseCode' element or 'responseCode' attribute.
- any HTTP headers to send with the response, through nested 'header' elements.
- the MIME type of the response via the 'contentType' attribute.

Attributes

Attribute name	Type	Default value	Description
href	String		Indicates the URL of a resource that contains the response to be sent back. A relative URL is interpreted as relative to the directory Anteater was started from. You can use any URL supported in Java, including HTTP and FTP, to read an external resource of file. Mandatory.
contentType	String	`text/html`	Specifies the MIME type to be associated with the response.
responseCode	int	`200`	The response code to be sent be as part of the HTTP response.

Elements allowed inside `sendResponse`

Element name	Description
responseCode	Specify the HTTP response code.
header	Adds a header to the HTTP response.
contentEquals	Specifies the HTTP body.
jelly	Specify a Jelly script, which will dynamically generate the HTTP response body.

Examples

The following listener responds to POSTed message by returning a HTTP 202 response, with header

'X-date' set to whatever ${date} is, and a text body with normalized spaces.

```
              <listener path="/text.txt">
                <match>
                  <method>POST</method>

                  <sendResponse>
                    <responseCode>202</responseCode>
                    <header name="X-date" value="${date}"/>
                    <contentEquals ignoreSpaces="true">
                      Here   is some   freeform text.
                    </contentEquals>
                  </sendResponse>
                </match>
              </listener>
```

This example receives a request on a URL and sends back an HTTP response with a response code of `201`:

```
<listener path="/good.html"
          description="Process a simple request">
  <match>
     <method value="GET"/>
       <sendResponse href="test/responses/good.html"
                     contentType="text/html"
                     responseCode="301"/>
   </match>
</listener>
```

9 Webapp tasks

These tasks relate to Anteater's ability to deploy webapps on an internal Tomcat server.

9.1 servletContainer

Anteater has the ability to listen for incoming HTTP requests, which is useful in testing asynchronous SOAP. Advanced SOAP applications interact by sending asynchronous SOAP requests to each other. After the request is sent, the other party responds later with a SOAP request to the initiating party.

Listening for incoming HTTP requests is achieved by embedding the Tomcat 3.3 servlet container inside Anteater.

servletContainer is used in conjunction with the listener element, which registers *`handlers`* with the servlet container. The servlet container should be started before any listener task is to be executed. Usually this is achieved by placing the `servletContainer` task inside an *`init`* target, on which all the other tasks depend upon:

```
<target name="init">
  <servletContainer port="8100, 8101"/>
</target>

<target name="my-test" depends="init">
  <listener path="/foo" ... /> <!-- Register a listener -->
  <httpRequest request="/foo" .. > <!-- Call the listener -->
</target>
```

Attributes

Attribute name	Type	Default value	Description
port	Integer	`8080`	Defines the ports on which the servlet container should listen on. Multiple ports can be specified by enumerating them with comma or space in between.
action	String	`start`	What action you wish the servlet container to take. If not specified, defaults to starting the container. Possible values are `start` and `stop`.
maxThreads	Integer		Defines the maximum number of threads that can be spawned off by the servlet container when handling incoming HTTP requests.
maxSpareThreads	Integer		Defines the maximum number of spare threads that can be alive at any time.
minSpareThreads	Integer		Defines the minimum number of spare threads that should be available at all times.

Elements allowed inside `servletContainer`: none

9.2 deploy

Because Anteater embeds a full blown servlet container in it, you can use Anteater to quickly deploy and test your Web application. This testing is not a replacement for testing how your Web application behaves when deployed on your preferred application server however. It is provided just as a quick way for you test the features of your application much faster.

In future, Anteater will add the ability to deploy a Web application on external servlet containers and application servers. And since Anteater is based on Ant, you can write a target which will start your application server right before the tests, run them and then shut it down at the end of the tests.

Attributes

Attribute name	Type	Default value	Description
path	String		Specifies the *`context`* path where the new Web application will be accessible in the URL space of the servlet container.
webapp	String		The path to the location on the file system of the Web application. This path should point to either a `.war` file or to the expanded directory of the Web application.

Elements allowed inside `deploy`: none

Examples

The following example shows how to deploy Apache Cocoon on Anteater's internal servlet container.

```
<target name="deploy" depends="init" description="Run Cocoon">
```

```
  <deploy path="/cocoon" webapp="../xml-cocoon2/build/cocoon/webapp"/>
  ...
</target>
```

10 Auxiliary tasks

These tasks have structural, configuration or metadata roles in an Anteater script.

10.1 group

A Group is a scoping mechanism for Anteater data types and tasks. A group acts as a container for sets of Anteater objects, where member objects can access each other. So, for example, action tasks will automatically use any logger or session objects defined in the group they belong to.

Since a Group is an Anteater object like any other, Groups can belong to groups. This is exploited to implement *`group inheritance`*, where Groups inherit properties, loggers and sessions from their *`parent`* Group, where a Group's *`parent`* is the Group it belongs to.

The inheritance rules are as follows:

- Properties are inherited unless overridden.
- Loggers are inherited as a set, ie they are either all inherited, or all overridden. Thus if our parent defines two loggers, and we define one, only our one will be used.
- Sessions are inherited unless overridden

There is a default, primordial Group to which all Groups and Tasks belong, unless otherwise specified. This is defined in org.apache.anteater.test.DefaultGroup , and is overridden by any group defined with id `default`.

Groups can be declared either within targets, or straight under Ant's `project` element.

See the Grouping and Configuration sections for a user-perspective overview of how grouping works.

Attributes

Attribute name	Type	Default value	Description
id	string		Sets the group's id string. This is a required attribute, since without it there is no way to refer to the group. If the id is set to `default`, the group will be used as the base of the group hierarchy, ie every other group and task will (possibly indirectly) belong to the default group.
inherits	string		Sets this group's parent group. This group inherits properties, loggers and sessions from its parent. If a group is assigned id `default`, it acts as the root group, from which all others inherit. Thus by redefining the default group (eg by adding a logger), one can change the behaviour of all tasks in a script. See Group-

Attribute name	Type	Default value	Description
			ing for details.

Elements allowed inside `group`

Element name	Description
session	Add a session to the group. This session will override that declared in the group's parent. All member tasks will use this session.
logger	Add a logger to the group, causing all member tasks to use it. No loggers will be inherited from the group's parent.
property	Add a property to the group. If a property of the same name was defined in this group's parent, then that property is overridden. Otherwise, properties are inherited. Properties are used to configured Anteater behaviour. See the Configuration section for more on this.

uses	Specifies requirements on the underlying Anteater installation that members of this group have. For example:
	`<uses>`
	`  <feature name="jelly"/>`
	`  <feature name="xhtml-schema"/>`
	`</uses>`
	Specifies that tasks in this group depend on the 'jelly' and 'xhtml-schema' Anteater upgrades.

group	Adds a group as a member of this group. The current group becomes the added group's parent. Alternatively, the `inherits` attribute may be used to indicate group inheritance.

Examples

Taken from the Grouping section:

```
<project name="groupdemo" default="main">
  <taskdef resource="META-INF/Anteater.tasks"/>
  <typedef resource="META-INF/Anteater.types"/>

  <group id="mytests">
    <property name="debug" value="0"/>
  </group>
  <group id="livesite" inherits="mytests">
    <property name="host" value="www.mysite.com"/>
    <logger type="xml" todir="{docs.dir}"/> <!--
    HTML report -->
  </group>
  <group id="devsite" inherits="mytests">
    <property name="host" value="www.mysite-dev.com"/>
    <property name="debug" value="1"/> <!-- devsite a bit unstable -->
    <property name="failonerror" value="true"/> <!-- Don't waste time testing whole
site -->
```

```
    <group id="devsite-brokenbit"> <!-- Very broken bit of devsite -->
      <property name="debug" value="10"/> </group>
  </group>

  <target name="main">
    <!-- Will have debug=10, host=www.mysite-dev.com, failonerror=true, and log
    to the console -->
    <httpRequest group="devsite-brokenbit" path="/broken.html"/>
  </target>
</project>
```

10.2 logger

Anteater logs various events that occur when running a script. These include notifications of errors (unexpected), failures (expected), when an action tasks and tests start or stop.

Typically, action tasks get their loggers through their group, although loggers can be added directly to action tasks. The default group contains a logger of type `colour`, which is responsible for the messages seen on the console.

The XML logger produces XML log files. These can be rendered to HTML by calling the built-in Anteater `report` task like this:

```
  <target name="report" description="Generates a HTML report">
    <ant antfile="${anteater.report}">
      <property name="log.dir" location="${log.dir}"/>
      <property name="report.dir" location="reports"/>
    </ant>
  </target>
```

The `${anteater.report}` variable is automatically set from the `anteater` script, as is `${anteater.home}`.

Attributes

Attribute name	Type	Default value	Description
type	String	`plain`	Specifies the type of logger, which determines what to do with logs. Currently defined loggers are: **minimal** Writes minimal text logs to the console **plain** Logs as plain text, by default to the terminal **colour** Logs as colour text (ANSI escape codes) to the terminal. By default, messages are displayed as normal, and errors are displayed in red, allowing one to detect at a glance if something went wrong. **xml** Logs as xml, by default to a file.
classname	String		The value must be a valid and existing Java class name.

Attribute name	Type	Default value	Description
			This attribute specifies exactly which class to use as a logger. The class can be defined externally to Anteater. Loggers must implement the Logger interface.
useFile	boolean		Specifies whether to send logs to a file or to the console.
filenameFormat	String		Specify the filename format for log files. Default is `TEST-${groupid}_${taskname}_${url}_line-${lineno}_test-${vm-count}${_run-:run}.xml` Pretty much any property can be used, both Anteater-specific properties (e.g. the task's `description`), Group properties, and Ant <property> properties. The following properties are predefined:

Property name	Description
groupid	Variable containing the 'group identifier' in filesystem-friendl form.
taskname	Variable containing name of c rently running Task
url	Variable containing a filesyste friendly rendition of the reque URL, minus the request param
lineno	Variable containing line numb of task that generated this log try
vm-count	Variable containing a counter unique within this Java Virtua Machine. Used to ensure uniqu ness for looping test outputs.
run	Variable containing a counter that ensures the current filenar will be unique within its directory. Prevents multiple runs of Anteater script from overwritin a single output file.
fqcn	Variable containing fully quali fied classname of current task.
raw-url	Variable containing unmodifie request URL.
raw-url-noparams	Variable containing request U with request params stripped.

Property name	Description	Ex
groupid-raw	Variable containing the 'group identifier' in unescaped form.	`de`

Attribute name	Type	Default value	Description
			There is one quirk in the format: variables of the form `${prefix:variable}`. These are interpreted as follows: if `${variable}` is defined, and has value `value`, then `${prefix:variable}` is replaced with 'prefix`value`'. For example, `${run_:run}` becomes 'run_1', or `${run at :date}` becomes 'run at 10/3/03'. If `variable` is undefined, the variable is replaced with ''. This hackery is primarily for the 'run' variable, which won't exist if `overwrite` is true (see below).
overwrite	boolean		Specifies whether to overwrite log files from previous Anteater runs. By default, if an Anteater script is run twice (two JVM instances), the log files of the second will overwrite the first. By setting overwrite to `false`, log files will have `_runX` appended to their name, where `X` is the next in the file sequence.
todir	String	`logs`	Specifies a directory in which to create logs, if any. The value must be a directory relative to Anteater's base directory. Only relevant if `useFile` is `true`.
extension	String		If logging to a file, sets the file extension, e.g. if the value is `.xml`, it becomes the file extension.
group	String		Add this logger to the specified group.

Elements allowed inside `logger`: none

10.3 session

Declares an object which stores cookies, and transparently maintains *`state`* between multiple action tasks.

The session object does what users have come to expect browsers to do; it caches cookies sent from the server, and resends them on subsequent requests to that server. This is the standard way in which *`state`* is maintained in HTTP-based client/server applications.

Usually, one would not need to use this tag, as the default group already defines a session. This tag is useful when you don't want to use the default session for some reason. A session can be shared among multiple action tasks by assigning it an id, and then using `refid` to refer to it.

Attributes

Attribute name	Type	Default value	Description
id	string		Sets the session id, for use later on with `<session refid="..."/>`

Elements allowed inside `session`: none

10.4 namespace

Specifies a mapping from XML namespace prefix to namespace URI. This mapping is used in XML-aware testers like xpath

A namespace mapping is required so that when namespace-prefixed elements are used in tasks like xpath , they correctly match equivalent elements in the HTTP response's XML, regardless of their prefix. So if we got back `<x:foo xmlns:x="some.uri"/>`, and tried to match it with `<xpath select="/y:foo"/>`, we'd need to a namespace mapping with `<namespace prefix="y" uri="some.uri"/>`

If you didn't understand a word of this, and don't know what a namespace is, please see the namespace FAQ .

Attributes

Attribute name	Type	Default value	Description
prefix	string		The namespace prefix. This prefix cannot be blank.
uri	string		The namespace URI to associate with the prefix.

Elements allowed inside `namespace`: none

Examples

This example applies a bunch of XPath tests to a Cocoon-generated XML document

```
<httpRequest path="/nsxml.xml">
  <!--
  We can't use a blank namespace here. According to the jaxen javadocs:
  "In XPath, there is no such thing as a 'default namespace'.  The
  empty prefix always resolves to the empty namespace"
  -->
  <namespace prefix="x" uri="http://xml.apache.org/cocoon/requestgenerator/2.0"/>
  <match>
    <xpath select="/"/>
    <xpath select="/x:request"/>
    <xpath select="/x:request/x:requestHeaders" assign="h"/>
    <xpath select="/x:request/x:requestHeaders/x:header[@name='host']"/>
    <xpath select="/x:request/x:requestHeaders/x:header[@name='host']/text()"/>
  </match>
</httpRequest>
```

10.5 uses

Specifies what Anteater features the script (or a group) needs to run. A Feature is either some aspect of Anteater itself (notably the version), or an optional feature.

Since the advent of the Update System , an Anteater install can have 'updates' applied to it, to give it extra capabilities. Scripts that rely on extra capabilities (extra schemas, for example) will break on Anteater installations lacking those updates. The <uses> tag lets such a script declare it's dependence on an optional feature.

The <uses> tag is scoped by the group it belongs to. By declaring it in the 'default' group, it applies to the whole script. <uses> tags are cumulatively inherited from parent groups, and only 'evaluated' when a task in the group is executed. Outside a group, a <uses> tag is meaningless, so they should always be found either inside a group tag, or have a 'group' attribute.

Attributes

Attribute name	Type	Default value	Description
version	dotted decimal (x.y.z)	`(Any anteater version)`	This optional attribute specifies the Anteater version the script is known to work with. The format is a series of decimals separated by dots, most significant first, eg '0.9.14'. Setting a version does *not* imply that the script is limited to running on the specified version (the tag is 'uses', not 'requires'). The version attribute merely provides information to Anteater, allowing future versions to maintain better backwards-compatibility (eg, by applying an XSLT at runtime to make a script comply with a later format).
group	string		Specifies the group that this 'uses' applies to. The same thing can be achieved by nesting the 'uses' tag inside a group element. The specified group must exist. If 'default', the requirements apply to the whole script.

Elements allowed inside `uses`

Element name	Description
feature	Specifies an optional Anteater 'upgrade' that tasks in the current group require to run.

Examples

Here is an example which applies to the whole script (default group), specifying the Anteater version known to work (0.9.14), and a requirement on the 'xhtml-schema' upgrade.

```
...
<group id="default">
  <uses version="0.9.14">
    <feature name="xhtml-schema"/>
  </uses>
</group>
```

Then later, the script could safely rely on the optional schema:

```
              <httpRequest>
                <match>
                  <relaxng
rngFile="${anteater.resources}/schemas/rng/xhtml/xhtml.rng"/>
                </match>
              </httpRequest>
```

Here is a hierarchy of groups to demonstrate how requirements are accumulated.

```
              <group id="default">
                <uses version="0.9.14"/>  <!-- Known to run with 0.9.14 -->
                <group id="xhtml-tests">
                  <uses>
                    <feature name="xhtml-schema"/>
                  </uses>
                  <group id="xhtml+mathml-tests">
                    <uses>
                      <feature
                        name="mathml-schema"/>
                    </uses>
                  </group>
                </group>
              </group>
```

Tasks in group 'xhtml-tests' will fail unless 'xhtml-schema' is installed, and tasks in group 'xhtml+mathml-tests' will fail unless both 'xhtml-schema' and 'mathml-schema' are installed.

10.6 feature

This tag is nested inside the uses tag. It specifies an Anteater feature that must be present, typically installed via the Update System

Attributes

Attribute name	Type	Default value	Description
name	string		Specifies the name of the required Anteater feature, eg 'jelly', or 'xhtml-schema'.

10.7 checkuses

The checkuses task will accumulate the features specified by all uses elements in the task's group, and check if the current Anteater installation can provide them.

This check is performed on every action task that contains (or whose group contains) a uses tag, but occasionally one may want to perform this check explicitly, which is what 'checkuses' is for. It takes no nested elements or attributes other than 'group'.

Attributes

Attribute name	Type	Default value	Description
group	string	`default`	Specifies the group whose requirements we are to check. Like all tasks, by default this belongs to the 'default'

Attribute name	Type	Default value	Description
			group.

Examples

Here is how we could rely on the 'jelly' upgrade to check if we can use the jelly task "natively".

```
              <uses group="default" version="0.9.13">
                <feature name="jelly"/>
              </uses>

              <target name="jelly">
                <checkuses/>
                <taskdef name="jelly"
                  classname="org.apache.commons.jelly.task.JellyTask"/>
                <jelly script="resources/jelly/hello_world.jelly"/>
                <echo>title is '${title}'</echo>
              </target>
```

As no 'group' is specified, 'default' is assumed. Without the <checkuses> element, the target would die with an error, as the specified class is not in Anteater by default.

11 External tasks

Anteater provides few more tasks which do not fit into any of the categories above. They are used to provide additional functionality, useful when writing tests.

Some of these tasks are provided by Ant Contrib project, distributed with Anteater.

The following tasks are available to Anteater:

- foreach - iterate over a collection of values

11.1 foreach

See the ant-contrib documentation for info about this task.

11.2 jelly

Many Anteater tasks can have nested content. For example, the contentEquals element can take nested text, which will constitute the HTTP body:

```
            <contentEquals>This is the HTTP body</contentEquals>
```

Anteater provides a way to dynamically generate this body text, using Jelly scripts. One simply replaces the static text with a <jelly> element:

```
            <contentEquals>
```

```
              <jelly script="generateResponse.jelly"/>
            </contentEquals>
```

Anteater will run the Jelly script, and use it's output as the nested content.

By default, Anteater does not come with all the jars required to use Jelly. To jelly-enable an Anteater installation, run the following command:

```
            [UNIX]    anteater -f $ANTEATER_HOME/resources/scripts/install-jelly.xml
            [Windows] anteater -f
%ANTEATER_HOME%\resources\scripts\install-jelly.xml
```

Or alternatively, run the following Anteater script:

```
<project name="jelly-install" default="main" basedir=".">
  <target name="main">
    <ant dir="${anteater.resources}/scripts" antfile="install-jelly.xml"/>
  </target>
</project>
```

Where ANTEATER_HOME is where you installed Anteater.

You will be prompted to enter the URL of a jar repository. If you just press enter, the default will be used, and the install is automatic from there on. Alternatively, if you have a Maven repository locally, you can try pointing the script at this.

Attributes

Attribute name	Type	Default value	Description
script	string		File path a Jelly script to run.
url	string		URL of a Jelly script to run.
output	string		Specifies the path to a file in which to store the script results.

Elements allowed inside `jelly`: none

12 Invoking from Ant

Often, people want to integrate Anteater with an existing Ant-based build system. Due to classpath issues, Anteater tasks cannot currently be used directly within an existing Ant script. The current solution is to invoke Anteater with a <java> task, as follows:

```
<property name="anteater.home" location="/usr/local/anteater"/>
<java classname="org.apache.tools.ant.Main" fork="true">
  <classpath>
    <pathelement location="${anteater.home}/resources"/>
    <fileset dir="${anteater.home}">
      <include name="lib/**/*.jar"/>
      <include name="tomcat/**/*.jar"/>
    </fileset>
  </classpath>
  <jvmarg value="-Dant.home=${anteater.home}"/>
  <jvmarg value="-Danteater.home=${anteater.home}" />
  <jvmarg value="-Danteater.report=${anteater.home}/resources/scripts/report.xml" />
  <jvmarg value="-Danteater.resources=${anteater.home}/resources" />
  <arg line="-f examples.xml"/>
  <arg value="-propertyfile" />
  <arg value="${anteater.home}/resources/META-INF/Anteater.properties" />


  <!--
  <arg value="-Ddefault.debug=10"/>
  -->
</java>
```

The `anteater.home` variable must be set to where you have installed Anteater. Replace `examples.xml` with your script. Alternatively, you can parametrize this:

```
<antcall target="anteater">
  <param name="script" value="examples.xml"/>
</antcall>
```

with this target:

```
<target name="anteater" description="Run Anteater">
  <property name="anteater.home" location="build/anteater-${version}"/>
  <java classname="org.apache.tools.ant.Main"
        fork="true">
    <classpath>
      <fileset dir="${anteater.home}">
        <include name="lib/**/*.jar"/>
        <include name="tomcat/**/*.jar"/>
      </fileset>
    </classpath>
    <jvmarg value="-Dant.home=${anteater.home}"/>
    <arg line="-f ${script}"/>
    <!--
    <arg value="-Ddefault.debug=10"/>
    -->
  </java>
</target>
```

13 Related projects

There are a number of related open source Java projects in the realm of functional testing:

13.1 Latka

Latka is an Apache Jakarta project with the same aims as Anteater. It has an equivalent set of *`validators`*, good documentation, and much better HTTP/HTTPS support than Anteater due to its use of the HttpClient API .

In terms of implementation, Latka plays it very straight, implementing its own scripting engine with the SAX API. No distiction is made between parse time and execution time. The general Latka API is clean and well-designed.

Anteater's primary advantage over Latka is the flexibility engendered by building on top of the Ant engine. Latka scripts do not let one set properties (although properties can be passed in), or the ability to group tests (Ant targets). However, it looks likely that future version of Latka will be based on Jelly , an

XML scripting language that is a functional superset of Ant, and would thus form an excellent base for a functional testing tool.

13.2 WebTest

Canoo WebTest is another Ant-based functional testing system. It is primarily aimed at testing HTML sites, with a number of HTML-specific validators, and the ability to *`script`* interactions over multiple HTML pages. In contrast, Anteater is more more low-level, but contains better support for XML and web services testing. WebTest and Anteater are thus quite complementary, especially since both run in Ant.

13.3 PushToTest TestMaker

PushToTest TestMaker is a relatively mature product, having been in development for 5 years (see this general@jakarta email). TestMaker scripts are written in Python; more specifically, jython , which compiles Python scripts into Java bytecode. They have a custom Java API (TOOL) which is the core testing code, and is used in jython scripts.

The approach of using Python as a testing language sounds really good. Why mess around inventing XML scripting languages when you could use a real one? Python is one of the best, and by using the Jython compiler, one gets all the portability of Java too.

In practice, I'm not too sure how well it works. The TestMaker scripts look very low-level, and overly complicated for what they do. But then I haven't really explored much, so don't take this criticism too seriously.

TestMaker is a curious project in the sense that it seems primarily an integration effort, combining the NetBeans API, Jython and a testing API together to create an integrated testing system. The result is good, especially if you like IDEs.

14 Acknowledgements

Anteater was started by Ovidiu Predescu 's need to have a testing framework for testing asynchronous Web services (those that send asynchronous SOAP messages between them, like ebXML and BizTalk), but also for testing Apache Cocoon .

Very early and extremely valuable feedback was provided by Jeff Turner . He later spent more time implementing new features and improving the general design and implementation of the code, and became an active developer.

Anteater would not have been possible without the Apache Ant project. Ant is a great little tool, very useful for robust software development. If you're not already using it, you should consider using it in your projects.

Anteater was inspired by Tomcat's 3.x HttpClient testing framework, whose primary author is Costin Manolache . HttpClient is still in use in the 3.x releases of Tomcat.

Anteater's `listener` facility would not have been possible without Costin's expert help, who fixed the

major bugs in the Anteater code embedding Tomcat 3.3.

Anteater makes use of some Ant extension tasks, provided by the Ant-contrib project.

The following people provided valuable feedback, which helped improve the usability and stability of the code:

- William Vambenepe
- Shridhar Diwan
- Bill Jones
- Ivelin Ivanov

Anteater's primary developers are:

- Ovidiu Predescu
- Jeff Turner