Getting started

by Ovidiu Predescu, Jeff Turner

1. 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

`ilecontents}`.

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.

1.1. 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.

1.2. 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

h:

```
<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.