Auxiliary tasks

by Ovidiu Predescu, Jeff Turner

1. Auxiliary tasks

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

1.1. 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.

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 Grouping for details.

Table 1: Attributes

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.

Table 2: Elements allowed inside group

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 sit

    <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>
```

1.2. 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}`.

Note:
Logging support is quite weak at the moment; many of the attributes don't work. Development has mainly gone into getting XML logging working. Send a feature request if you need this fixed.

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. 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. **Note:** Loggers currently ignore this; XML always logs to a file, the text loggers always log to the screen
filenameFormat	String		Specify the filename

			format for log files. Default is `TEST-${groupid}_${taskname}_${` 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		
groupid		Variab identif form.
taskname		Variab curren
url		Variab filesys the r reque
lineno		Variab of tas entry
vm-count		Variab uniqu Machi uniqu outpu
run		Variab that filenar its dir runs c overw
fqcn		Variab qualifi task.
raw-url		Variab

		reque
raw-url-noparams		Variab URL stripp
groupid-raw		Variab identif

			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. **Note:** The rest of the filename is determined by the logger, and will generally be chosen to be unique within the directory.
group	String		Add this logger to the specified group.

Table 1: Attributes

Elements allowed inside `logger`: none

1.3. 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.

Note:
Currently, there is no way to undefine a session, other than to edit default.properties in the jar and turn the default session off. Send a feature request if you need this fixed.

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

Table 1: Attributes

Elements allowed inside `session`: none

1.4. 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"/>`

Note:
This is really a hack; if Ant made namespaces available to tasks, one could instead use normal XML `xmlns` attributes to declare namespace mappings.

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

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.

Table 1: Attributes

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>
```

1.5. 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.

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.

Table 1: Attributes

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

Table 2: Elements allowed inside uses

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.

1.6. feature

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

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

Table 1: Attributes

1.7. 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'.

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' group.

Table 1: Attributes

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.