"...probably the best custom task
XmlTask mailing list, Nove

"...too essential to be left out of the Ant dist
Ant User mailing list, A

Contents

Introduction

xmltask provides the facility for automatically editing XML files as part of an Ant build. Unlike the standard `filter` task provided with Ant, XML-sensitive, but doesn't require you to define XSLTs.

Uses include:

- modifying configuration files for applications during builds
- inserting/removing information for J2EE deployment descriptors
- dynamically building Ant build.xml files during builds
- building and maintaining websites built with XHTML
- driving Ant via a *meta* build.xml to abstract out build processes

"Keep up the good work - xmltask is invaluable to our promotic
User comment

Top

Recent Changes

Read the new tutorial on <xmltask> at java.net

Current version 1.16

- Regular expressions for changing text are now available.

- Copying/cutting to properties can now handle multiple values from an XPath expression. String trimming and concatenation (with a specified separator character) is now supported.
- Support for Java versions prior to 1.5 has been removed. Older versions of **xmltask** are available from the Source project download area.

See the **CHANGES** file in the download for a comprehensive list of changes for each version

Top

Download

Version 1.16 - release 22-Sep-2009

xmltask is released under the Apache license. Right-click and choose *Save As* if your browser doesn't offer you the option.

xmltask.jar - the .jar file to use in Ant.

xmltask.tar.gz - the source code and tests

Checksums:
xmltask.jar - MD5 b2cda8a5800858d503a8b11253a3e642
xmltask.tar.gz - MD5 bdd9815475018b8b4dbcfde08319ebe2

Download previous versions at Sourceforge.net

Top

How To Use

To use this task, make sure:

The xmltask.jar is in your **$CLASSPATH**

Reference the **xmltask** in your **build.xml** eg.

```
<taskdef name="xmltask" classname="com.oopsconsultancy.xmltask.ant.XmlTask"/>
```

Reference the **xmltask** task as part of your build eg.

```
<target name="main">

<xmltask source="input.xml" dest="output.xml">

...
```

Top

Support

xmltask now has a support mailing list.

Email xmltask-users@lists.sourceforge.net

To subscribe, or view the archives, visit the Xmltask-users mail page

Top

Usage

xmltask allows you to specify sections of an XML file to append to, replace, remove or modify. The sections of the XML document to be mc specified by XPath references, and the XML to insert can be specified in-line in the Ant build.xml, or loaded from files.

- The main <xmltask> section takes arguments to define an XML source and a destination file or directory. Note that the XML source if you're creating a new document via <xmltask> instructions. **dest** and **todir** can be omitted if you're reading a document and st subsections in buffers for use by another task (see below).

  <fileset>s are used to define sets of files for **xmltask** to operate on. See the standard Ant documentation for information on using

Parameters

Attribute	Description	Re
source	the source XML file to load. Can take the form of a wildcarded pattern eg. **/*.xml. Note that this capability will be deprecated in favour of <fileset> usage	

sourcebuffer	the source buffer containing XML from a previous <xmltask> invocation. The buffer must contain a *single* root node (i.e be *well-formed*). If the buffer is empty, then this has the effect of starting with a blank document.	
dest	the output XML file to write to	
destbuffer	the output buffer to write to	
todir	the output directory to write to	
report	when set to true, will result in diagnostic output.	
public	sets the PUBLIC identifier in the output XML DOCTYPE declaration.	
expandEntityReferences	when set to true, will enable entity reference expansion. Defaults to true	
system	sets the SYSTEM identifier in the output XML DOCTYPE declaration.	
preservetype	when set to true sets the PUBLIC and SYSTEM identifiers to those of the original document	
failWithoutMatch	when set to true will stop the **xmltask** task (and hence the build process) if any subtask fails to match nodes using the given XPath path	
indent	when set to true enables indented formatting of the resultant document. This defaults to true	
encoding	determines the character encoding value for the output document	
outputter	determines the output mechanism to be used. See formatting for more info.	
omitHeader	when set to true forces omission of the <?xml...?> header. Note that the XML spec specifies the header SHOULD be included, but this is not mandated for XML v1.0	
standalone	when set to true/false sets the **standalone** attribute of the header	
clearBuffers	Clears buffers after population by previous **xmltask** invocations. Buffers are cleared after every input file is processed. Buffers are specified in a comma-delimited string	

e.g.

```
<xmltask source="input.xml" dest="output.xml">
```

reads from **input.xml** and writes to **output.xml**

```
<xmltask todir="output">
 <fileset dir=".">
  <includes name="*.xml"/>
```

reads from the XML files in the current dir and writes to the same filenames in the **output** dir.

```
<xmltask sourcebuffer="servlet" output="servlet.xml">
```

reads from the previously populated buffer **servlet** and writes to **output.xml**

```
<xmltask source="input.xml" destbuffer="output">
```

reads from a file **input.xml** and writes to the buffer called **output**.

- Nested elements allow replacements to take place, and are applied in the order that they're specified in. Each subsection may matcl more nodes. Standard XPath paths are used here. If you're not familiar with these, the examples below will provide some hints. See more info.
    - The <cut> section allows an XML section to be cut and stored in a buffer or a property. Multiple XML nodes or elements can buffer or property by using the **append** attribute

      Parameters

Attribute	Description	Rec
path	the XPath reference of the element(s) to cut	
buffer	the buffer to store the cut XML	
property	the property to store the cut XML	
append	when set to **true**, *appends* to the given buffer or property. You can only append when creating a new property since Ant properties are immutable (i.e. when an XPath resolves to multiple text nodes)	
attrValue	Cutting an attribute will result in the *whole* attribute plus value being cut. When **attrValue** is set to true then only the attribute's *value* is cut. This is *implicit* for cutting to properties	
trim	trims leading/trailing spaces when writing to properties	

propertySeparator	defines the separating string when appending properties	
if	only performed if the given property is set	
unless	performed *unless* the given property is set	

e.g.

```
<cut path="web/servlet/context/root[@id='2']/text()" buffer="namedBuffer"/>
<cut path="web/servlet/context/root[@id='2']/text()" property="property1"/>
```

- The <copy> section allows an XML section to be copied and stored in a buffer or a property. Multiple XML nodes or element copied to a buffer or property by using the **append** attribute

Parameters

Attribute	Description	Req
path	the XPath reference of the element(s) to copy	
buffer	the buffer to store the copied XML	
property	the property to store the copied XML	
append	when set to **true**, *appends* to the given buffer or property. You can only append when creating a new property since Ant properties are immutable (i.e. when an XPath resolves to multiple text nodes)	
attrValue	Copying an attribute will result in the *whole* attribute plus value being cut. When **attrValue** is set to true then only the attribute's *value* is copied. This is now *implicit* for copying to properties	
propertySeparator	defines the separating string when appending properties	
trim	trims leading/trailing spaces when writing to properties	
if	only performed if the given property is set	
unless	performed *unless* the given property is set	

e.g.

```
<copy path="web/servlet/context/root[@id='2']/text()" buffer="namedBuffer"/>
<copy path="web/servlet/context/root[@id='2']/text()" property="property1"/>
```

- The <paste> section allows the contents of a buffer or a property to be pasted into an XML document. This is a **synonym** f insert section (see below)

- The <insert> section allows you to specify an XML node and the XML to insert below or alongside it

Parameters

Attribute	Description	Req
path	the XPath reference of the element(s) to insert into	
buffer	the buffer to paste	
file	the file to paste	
xml	the literal XML to paste	
expandProperties	indicates whether properties in body text XML are expanded or not. Defaults to **true**	
position	where the XML is to be inserted in relation to the XML highlighted by **path**. The allowed positions are **before**, **after**, or **under**. The default position is **under**.	
if	only performed if the given property is set	
unless	performed *unless* the given property is set	

e.g.

```
<insert path="/web/servlet/context/root[@attr='val']" xml="&lt;B/&gt;"/>
<insert path="/web/servlet/context/root[@attr='val']" file="insert.xml"/>
<insert path="/web/servlet/context/root[@attr='val']" buffer="namedBuffer" position="before"
<insert path="/web/servlet/context/root[@attr='val']" xml="${property1}" position="before"/>
```

The XML to insert can be a *document fragment* - that is to say it doesn't require a root node. Examples of insertable XML in

```
<welcome-file-list/>
```

(a well formed document)

```
<servlet-mapping id="1"/><servlet-mapping id="2"/>
```

(a well-formed document without a root node)

The XML to insert can be specified as *body text* within the <insert> task eg.

```
<insert path="web/servlet/context/root[@id='2']/text()">
<![CDATA[
 <node/>
]]>
</insert>
```

Note that the XML has to be specified within a CDATA section. Ant properties are expanded within these sections, unless **expandProperties** is set to **false**

You can create a new document by not specifying a source file, and making the first instruction for <xmltask> an <insert> with the appropriate root node (and any subnodes).

- The <replace> section allows you to specify an XML node and what to replace it with

Parameters

Attribute	Description	Req
path	the XPath reference of the element(s) to replace. If this represents an attribute, then the value of the attribute will be changed. In this scenario you can only specify text as replacement	
withText	the text to insert in place of the nominated nodes	
withXml	the literal XML to insert in place of the nominated nodes	
withFile	the file containing XML to insert in place of the nominated nodes	
withBuffer	the buffer containing XML to insert in place of the nominated nodes	
expandProperties	indicates whether properties in body text XML are expanded or not. Defaults to **true**	
if	only performed if the given property is set	
unless	performed *unless* the given property is set	

e.g.

```
<replace path="web/servlet/context/root[@id='2']/text()" withText="2"/>
```

```
<replace path="web/servlet/context/root[@id='2']/@id" withText="3"/>
<replace path="web/servlet/context/root[@id='2']/text()" withXml="&lt;id&gt;"/>
<replace path="web/servlet/context/root[@id='2']/" withFile="substitution.xml"/>
<replace path="web/servlet/context/root[@id='2']/" withBuffer="namedBuffer"/>
```

(note that to include literal XML using **withXml**, angle brackets have to be replaced with entities). The XML can be a well-fo document *without* any root node. The XML to insert can be specified as *body text* within the <replace> task eg.

```
<replace path="web/servlet/context/root[@id='2']/text()">
<![CDATA[
 <node/>
]]>
</replace>
```

Note that the XML has to be specified within a **CDATA** section. Ant properties are expanded within these sections, unless **expandProperties** is set to **false**

- The <attr> section allows you to specify an XML node and how to add, change or remove its attributes

Parameters

Attribute	Description	Req
path	the XPath reference of the element(s) to be changed	y
attr	the name of the attribute to be added/changed or removed	y
value	the value to set the attribute to	

remove	if set to **true**, indicates that the nominated attribute should be removed	
if	only performed if the given property is set	
unless	performed *unless* the given property is set	

e.g.

```
<attr path="web/servlet/context[@id='4']/" attr="id" value="test"/>
<attr path="web/servlet/context[@id='4']/" attr="id" remove="true"/>
```

Note that in the first example, if the attribute **id** doesn't exist, it will be added.

- The <remove> section allows you to specify an XML section to remove

Parameters

Attribute	Description	Rec
path	the XPath reference of the element(s) to be removed	
if	only performed if the given property is set	
unless	performed *unless* the given property is set	

e.g.

```
<remove path="web/servlet/context[@id='redundant']"/>
```

- The <regexp> section allows you to specify XML text to change via regular expressions.

Parameters

Attribute	Description	Rec
path	the XPath reference of the element(s) to be changed or copied	
pattern	The regular expression to apply to the text node or attribute value	
replace	The text to replace the matched expression with	
property	The property to copy the matched expression into. A capturing group must be used to specify the text to capture	
buffer	The buffer to copy the matched expression into. A capturing group must be used to specify the text to capture	
casesensitive	Sets case sensitivity of the regular expression. Defaults to *true*	
if	only performed if the given property is set	
unless	performed *unless* the given property is set	

The **<regexp>** task uses the standard Java regular expression mechanism. Replacements can make use of capturing group copying to a buffer or a property, a capturing group *must* be specified to determine the text to be copied.

e.g.

```
<regexp path="/web-app/servlet/servlet-name/text()" pattern="Test" replace="Prod"/>
<regexp path="/web-app/servlet/servlet-name/text()" pattern="Servlet-([a-z])-([0-9]*)"
replace="Servlet-$2-$1"/>
<regexp path="/web-app/servlet/servlet-name/text()" pattern="(.*)Test" property="servlet.nam
<regexp path="/web-app/servlet/servlet-name/text()" pattern="(.*)Test" buffer="servlet.name"
```

Note the use of the capturing groups to reverse components of the servlet's name, or to determine the servlet name substring to buffer or property.

- The <rename> section allows you to specify an XML element or attribute to rename

Parameters

Attribute	Description	Rec
path	the XPath reference of the element(s) to be renamed	
to	the new node name	

if	only performed if the given property is set	
unless	performed *unless* the given property is set	

e.g.

```
<rename path="a/b/c[@id='1']" to="d"/>
<rename path="a/b/@c" to="d"/>
```

- The <call> section allows you to perform actions or call Ant targets in the same **build.xml** file for nodes identified by an X

Parameters

Attribute	Description	Rec
path	the XPath reference of the element(s) to be identified	
target	the Ant target to call for each identified node	
buffer	the buffer to use to store each identified node (for the duration of the target call)	
inheritAll	boolean indicating if the target being called inherits all properties. Defaults to *true*	
inheritRefs	boolean indicating if the target being called inherits all references. Defaults to *false*	
if	only performed if the given property is set	
unless	performed *unless* the given property is set	

e.g. in the below example, the Ant target **CNode** is called for *each* occurrence of the **C** node in the given XPath expression. F to **CNode** the buffer **abc** is populated with the node identified (plus any subnodes).

```
<call path="a/b/c" target="CNode" buffer="abc"/>
```

In the below example, Ant actions are *embedded* within the **<call>** action (Ant 1.6 and above only):

```
<call path="a/b/c">
  <actions>
    <echo>Found a node under a/b/c</echo>
  </actions>
</call>
```

This mechanism can be used to drive Ant builds from existing XML resources such as **web.xml** or **struts.xml**, or to pro **meta-build** facility for Ant, by driving the **build.xml** from a higher level proprietary XML config.

Properties can be set for the target being called using XPath syntax or simply as existing properties or static strings. eg.

```
<call path="a/b/c" target="CNode" buffer="abc">
  <param name="val" path="text()"/>
  <param name="id" path="@id" default="n/a"/>
  <param name="os" value="${os.name}"/>
</call>
```

will call the Ant target **CNode** as above, but for each invocation, the property *val* is set to the value of the text node under **C** property id is set to the corresponding id attribute. If the id attribute is missing then "n/a" will be substituted. *os* is set to the

The same can be done for *embedded* actions:

```
<call path="a/b/c">
  <param name="val" path="text()"/>
  <param name="id" path="@id" default="n/a"/>
  <param name="os" value="${os.name}"/>
  <actions>
    <echo>val = @{val}</echo>
    <echo>id = @{id}</echo>
  </actions>
</call>
```

Note how the parameters are dereferenced in this example (using **@{...}**). Note also that for embedded actions each prope have a value assigned to it. If in doubt use the **default** attribute in the **<param>** instruction.

- The <print> section allows you to dump out to standard output the XML matching a given XPath expression, or the content buffer. This is a considerable help in performing debugging of scripts

Parameters

Attribute	Description	Req
path	the XPath reference of the element(s) to be identified	
buffer	the buffer to print out	
comment	a corresponding comment to print out	

e.g..

```
<print path="a/b/c" comment="Nodes matching a/b/c"/>
<print buffer="buffer1" comment="Contents of buffer 1"/>
```

This instruction has no effect on the documents being scanned or generated.

- **xmltask** now supports the Ant 1.5 <xmlcatalog> element, which allows you to specify local copies of DTDs allows you to specify a DOCTYPE referred to in the original document, and the local DTD to use instead (us you're behind firewalls and the like).

  e.g.

```
<xmlcatalog id="dtds">
 <dtd publicId="-//OOPS Consultancy//DTD Test 1.0//EN" location="./local.dtd"/>
</xmlcatalog>

<xmltask source="18.xml" dest="18-out.xml" report="true">
  <xmlcatalog refid="dtds"/>
 <!-- set a text element to a value -->
 ...
</xmltask>
```

  references a local copy of a DTD.

  Alternatively, you can use the legacy <entity> element within <xmltask>, as below:

```
<entity remote="-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN" local="web.dtd"/>
<entity remote="-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN" local=""/>
```

  The first version above specifies a local version of the DTD. The second indicates that the remote entity will be ignored comp Note that the **remote** attribute can take either the PUBLIC specification or the SYSTEM specification.

- The **uncomment** instruction allows you to uncomment sections of XML. This means you can maintain differe fragments within one document and enable a subset. For instance you can maintain different configs and enable one at deployment

  Parameters

Attribute	Description	Req
path	the path of the comment to uncomment. This must resolve to a comment within the input document	
if	only performed if the given property is set	
unless	performed *unless* the given property is set	

  e.g.

```
<xmltask source="server.xml" dest="server.xml" report="true">
  <!-- enables a servlet configuration -->
  <uncomment path="/server/service[@name='Tomcat-Standalone']/comment()"/>
 ...
</xmltask>
```

- The sections above can be chained together to provide successive modifications to an XML file eg.

```
<target name="main">
 <xmltask source="input.xml"
     dest="output.xml
```

```
      public="//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
      system="http://java.sun.com/j2ee/dtds/web-app_2_2.dtd"
      report="true">
    <replace path="web/servlet/context/config[@id='1']/text()" withFile="config1.xml"/>
    <replace path="web/servlet/context/config[@id='2']/text()" withFile="config2.xml"/>
    <insert path="/web/security/" file="uat.security.xml"/>
    <remove path="web/servlet/context/config[@id='4']"/>
   </xmltask>
  </target>
```

Here the **report** attribute is enabled to view the XML transformations as they occur. The input is loaded from **input.xml** and the go to **output.xml**. The files **config1/2.xml** replace the text below the appropriate <config> nodes, the file **security.xml** is in then the config id #4 is removed. **output.xml** will be output with the appropriate **DOCTYPE** setting for a Servlet 2.2 **web.xml** (usi **public/system** settings - note that if **input.xml** has the **public** and **system** ids set already, **preserveType="true"** could b here).

Top

Buffers

Buffers are used to store nodes found by <cut> and <copy> operations, and those nodes can be inserted into a document using <insert> /

Buffers exist for the duration of the Ant process and consequently can be used across multiple invocations of <xmltask>. eg. the following i

```
<target name="cut">
  <xmltask source="input.xml" dest="1.xml >
    <cut path="web/servlet/context/config[@id='4']" buffer="storedXml" />
  </xmltask>
</target>

<target name="paste" depends="cut">
  <xmltask source="input.xml" dest="output.xml >
    <paste path="web/servlet/context/config[@id='5'] buffer="storedXml" />
  </xmltask>
</target>
```

so the buffer *storedXml* is maintained across multiple targets.

Buffers are simply defined by names. eg. valid buffers would be *servlet*, *buffer100* etc.

A buffer can record *multiple* nodes (either resulting from multiple matches or multiple <cut> / <copy> operations). This operation is enable use of the `append` attribute. e.g.

```
<cut path="web/servlet/context/config" buffer="storedXml" append="true" />
```

A buffer can store all types of XML nodes e.g. text / elements / attributes. Note that when recording an attribute node, both the name of th and the value will be recorded. To store the value alone of an attribute, the `attrValue` attribute can be used e.g.

```
<copy path="web/servlet/@id" buffer="id" attrValue="true" />
```

This will store the value of the **id** attribute. The value can be used as a text node in a subsequent <insert> / <paste>.

Buffers can be persisted to files. This permits buffers to be used across Ant invocations, and uses of <antcall>. To persist a buffer to a file, s it using a file URL. e.g.

```
    <cut path="/a/b" buffer="file://build/buffers/1"/>
```

and the operation will write the cut XML to a file **build/buffers/1**. This file will persist after Ant exits, so care should be taken to remove required. The file will be created automatically, but any directories required must exist prior to the buffer being used.

Top

Formatting

The formatting of the output document is controlled by the attribute 'outputter'. There are three options:

```
<xmltask outputter="default"...
```

outputs the document *as is*. That is to say, all whitespace etc. is preserved. This is the default option. Note that attribute ordering *may* chan elements containing attributes may be split over several lines etc. ie. the document remains the same *semantically*.

<xmltask outputter="simple"...
outputs the document with a degree of formatting. Elements are indented and given new lines wherever possible to make a more readable This is not suitable for all applications since some XML consumers will be whitespace sensitive.

Spacing can be adjusted by using **<xmltask outputter="simple:{indent}...>".** e.g. **<xmltask outputter="simple:1"...** re

```
<root>
 <branch/>
</root>
```

The indent level can be increased: **<xmltask outputter="simple:4"...** results in:

```
<root>
    <branch/>
</root>
```

<xmltask outputter="{classname}"...
outputs the document using the nominated class as the outputting mechanism. This allows you to control the output of the document to yo tastes. The specified class must:

1. have a default constructor (i.e. no arguments)
2. implement the com.oopsconsultancy.xmltask.output.Outputter interface.

The custom class will be loaded and instantiated, then passed to a javax.xml.transform.sax.SAXResult object. Hence the outputter object w SAX events for each node in the resultant XML document. Note:

1. **com.oopsconsultancy.xmltask.output.Outputter** extends **org.xml.sax.ContentHandler**, so the appropriate SAX methods need to be implemented.
2. The standard SAX callbacks will not include callbacks for comments, **CDATA** sections etc. If you want to receive these events, then you also need to implement the **org.xml.sax.ext.LexicalHandler** interface as well.
3. For each callback, you should generate your results and write them to the writer object passed in via **setWriter()**

A simple introduction is to look at the **com.oopsconsultancy.xmltask.output.FormattedDataWriter** source code (in the source t

Top

Examples

Some examples of common usage:

- Extracting the title from an XHTML file and storing it in a buffer:

```
<copy path="/xhtml/head/title/text()" buffer="title"/>
```

- Extracting the title from an XHTML file and storing it in a property:

```
<copy path="/xhtml/head/title/text()" property="title"/>
```

- Inserting a servlet definition into a **web.xml**. Note how this occurs *only* if the property **insert.reqd** is set:

```
<insert if="insert.reqd" path="/web-xml/servlet[last()]" position="after" file="newservlet.xml
```

- Inserting a servlet definition into a **web.xml** (another way - note properties usage):

```
<insert path="/web-xml/servlet[last()]" position="after">
  <![CDATA[
    <servlet>`
      <servlet-name>
```

```
        ${project.name}
      </servlet-name>
    </servlet>
  ]]>
</insert>
```

- Replacing text occurences within particular div tags:

```
<replace path="//div[@id='changeMe']/text()" withText="new text"/>
```

- Changing an attribute (method number 1):

```
<attr path="//div[@id='1']" attr="id" value="2"/>
```

- Changing an attribute (method number 2):

```
<replace path="//div[@id='1']/@id" withText="2"/>
```

- Removing an attribute:

```
<remove path="//div[@id='1']/@id"/>
```

- Removing an attribute (another way):

```
<attr path="//div[@id='1']" attr="id" remove="true"/>
```

- Copying an attribute's value into a buffer:

```
<copy path="//div[@id='1']/@id" attrValue="true" buffer="bufferName"/>
```

- Copying an attribute's value into a property:

```
<copy path="//div[@id='1']/@id" property="propertyName"/>
```

- Copying multiple values into one buffer. Note the clearing of buffers **a, b** and **c** prior to appending. Buffer **b** con
  the **div** elements for each input file :

```
<xmltask clearBuffers="a,b,c">
  <fileset dir=".">
    <includes name="*.xml"/>
  </fileset>
  <copy path="//div" buffer="b" append="true"/>
  ...
```

- Removing all comments:

```
<remove path="//child::comment()"/>
```

- Inserting the appropriate system identifiers in a transformed **web.xml**:

```
<xmltask source="web.xml" dest="release/web.xml"
    public="-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN "
    system="http://java.sun.com/j2ee/dtds/web-app_2_2.dtd" >
    ...
```

OR

```
<xmltask source="web.xml" dest="release/web.xml"
    preserveType="true"
    ...
```

if you're transforming an existing **web.xml**.

- Setting the output character set to Japanese encoding:

```
<xmltask source="web.xml" dest="release/web.xml"
    encoding="Shift-JIS" >
    ...
```

- Converting all unordered lists in an XHTML document to ordered lists

```
<rename path="//ul" to="ol"/>
```

- Creating a new document with a root node <root>

```
<xmltask dest="release/web.xml">
  <insert path="/">
    <![CDATA[
      <root/>
    ]]>
  </insert>
    ...
```

- Counting nodes and recording the result in a property

```
<xmltask source="multiple.xml">
  <copy path="count(/servlet)" property="count"/>
    ...
```

- Identifying elements with namespaces. This example copies the **node** element which is tied to a namespace via a directive. See this XML.com article for namespace-related issues.

```
<xmltask source="input.xml">
  <copy path="//*[local-name()='node']" property="count"/>
    ...
```

- Call the *deploy* task for each servlet entry in a **web.xml**. For each invocation the **servletDef** buffer contains the c servlet specification from the deployment file, and the property *id* contains the servlet id (if there is no id attribu *NO ID* will be substituted). The **servletDef** buffer can be used in suceeding **xmltask** invocations.

```
<xmltask source="web.xml">
  <call path="web/servlet" target="deploy" buffer="servletDef"/>
    <param name="id" path="@id" default="NO ID"/>
  </call>
</xmltask>
```

- Performs actions for each servlet entry in a **web.xml**. For each invocation the embedded actions are performed (A above only).

```
<xmltask source="web.xml">
  <call path="web/servlet"/>
    <param name="id" path="@id" default="NO ID"/>
    <actions>
      <echo>Found a servlet @{id}</echo>
      <!-- perform deployment actions -->
      ....
    </actions>
  </call>
</xmltask>
```

- Uncomment and thus enable a set of users in a tomcat-users.xml file. The users are set up in the first 2 commen

```
<xmltask source="tomcat-users.xml">
  <uncomment path="tomcat-users/comment()[1]"/>
  <uncomment path="tomcat-users/comment()[2]"/>
</xmltask>
```

- Cutting a section of XML to a buffer, and displaying the buffer to confirm to the developer that a suitable XML fra been identified/stored

```
<xmltask source="input.xml">
  <cut path="web/servlet[@id='1']" buffer="servlet"/>
  <print buffer="servlet" comment="Copied to 'servlet' buffer"/>
    ...
```

- Cutting a section of XML to a persisted buffer (the file **buffers/servlet**) for later use

```
<xmltask source="input.xml">
  <cut path="web/servlet[@id='1']" buffer="file://build/buffers/servlet"/>
    ...
```

Top

Known Issues

The Java 1.4.2 release as of June 2003 has tightened up XPath parsing and what is regarded as acceptable XPath synta particular, the following usage of trailing path separators is now regarded as incorrect:

/root/branch/

and should be replaced with

/root/branch

Some of the **xmltask** examples and documentation have used the incorrect syntax. This is now rectified.

Top

XPath Links

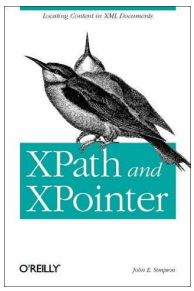
The XPath spec can be found here

An excellent XPath tutorial can be found here

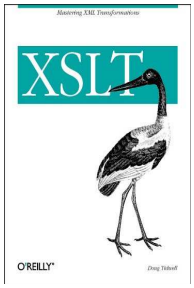
Hints and tips on XPath can be found at O'Reilly's XML web site

Many Xpath issues relate to namespace matching. This XML.com article offers an excellent discussion of the issues.

The following books are invaluable for XPath issues:

XPath and XPointer by John E. Simpson 

XSLT by Doug Tidwell. Contains useful XPath information



Top

Contact

xmltask is written and maintained by OOPS Consultancy Ltd.. For enquiries see the support info.

Now hosted on

SOURCEFORGE.NET®