Reading XML with Groovy and XPath

Car Example

This example assumes the following class is already on your CLASSPATH:

XmlExamples.groovy

```
class XmlExamples {
  static def CAR_RECORDS = '''
    <records>
      <car name='HSV Maloo' make='Holden' year='2006'>
        <country>Australia</country>
        <record type='speed'>Production Pickup Truck with speed of 271kph</record>
      </car>
      <car name='P50' make='Peel' year='1962'>
        <country>Isle of Man</country>
        <record type='size'>Smallest Street-Legal Car at 99cm wide and 59 kg in
weight</record>
      </car>
      <car name='Royale' make='Bugatti' year='1931'>
        <country>France</country>
        <record type='price'>Most Valuable Car at $15 million</record>
      </car>
    </records>
  '''
}
```

Using Xalan

Here is an example of using Apache Xalan with Groovy to read an existing XML file:

```
// require(groupId:'xalan', artifactId:'xalan', version:'2.6.0')
import org.apache.xpath.XPathAPI
import javax.xml.parsers.DocumentBuilderFactory

messages = []

def processCar(car) {
    def make = XPathAPI.eval(car, '@make').str()
    def country = XPathAPI.eval(car, 'country/text()').str()
    def type = XPathAPI.eval(car, 'record/@type').str()
    messages << make + ' of ' + country + ' has a ' + type + ' record'
}

def builder     = DocumentBuilderFactory.newInstance().newDocumentBuilder()
def inputStream = new ByteArrayInputStream(XmlExamples.CAR_RECORDS.bytes)
def records     = builder.parse(inputStream).documentElement

XPathAPI.selectNodeList(records, '//car').each{ processCar(it) }

assert messages == [
    'Holden of Australia has a speed record',
    'Peel of Isle of Man has a size record',
    'Bugatti of France has a price record'
]
```

Using Jaxen

Here is an example of using Jaxen with Groovy to read an existing XML file:

```
// require(groupId:'jaxen', artifactId:'jaxen', version:'1.1-beta-10')
import org.jaxen.dom.DOMXPath
import javax.xml.parsers.DocumentBuilderFactory

messages = []

def processCar(car) {
    def make = new DOMXPath('@make').stringValueOf(car)
    def country = new DOMXPath('country/text()').stringValueOf(car)
    def type = new DOMXPath('record/@type').stringValueOf(car)
    messages << make + ' of ' + country + ' has a ' + type + ' record'
}

def builder     = DocumentBuilderFactory.newInstance().newDocumentBuilder()
def inputStream = new ByteArrayInputStream(XmlExamples.CAR_RECORDS.bytes)
def records     = builder.parse(inputStream).documentElement

new DOMXPath('//car').selectNodes(records).each{ processCar(it) }

assert messages == [
    'Holden of Australia has a speed record',
    'Peel of Isle of Man has a size record',
    'Bugatti of France has a price record'
]
```

Note: many libraries (e.g. DOM4J, JDOM, XOM) bundle or provide optional support for Jaxen. You may not need to download any additional JARs to use it.

Using Java's native XPath support

If you are using Java 5 and above, you don't need to use the standalone Xalan jar but instead can use the now built-in XPath facilities:

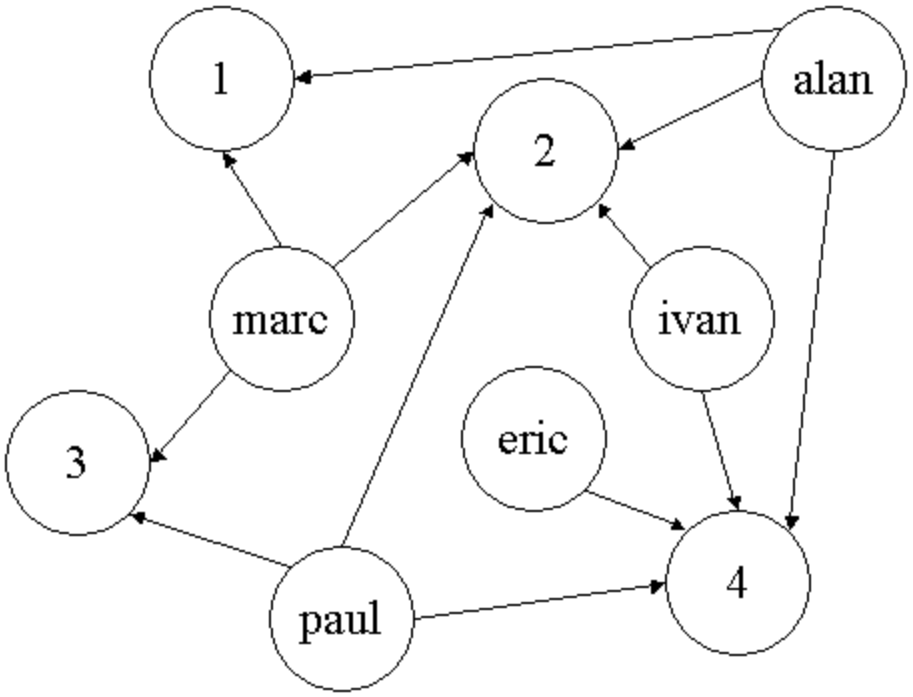
```
import javax.xml.xpath.*

def xpath = XPathFactory.newInstance().newXPath()

def nodes = xpath.evaluate( '//car', records, XPathConstants.NODESET )
nodes.each{
  def make = xpath.evaluate( '@make', it )
  def country = xpath.evaluate( 'country/text()', it )
  def type = xpath.evaluate( 'record/@type', it )
  messages << "$make of $country has a $type record"
}
```

People in Groups Example

Inspired by this example, here is how to use XPath to determine which groups include all three of *alan*, *paul* and *ivan* for this graph representing group membership.



Picture source: http://stage.vambenepe.com/pages/graph.png

First, here is the XML data capturing the group membership information:

```
def xml = '''
<doc>
  <person name="alan"><g>1</g><g>2</g><g>4</g></person>
  <person name="marc"><g>1</g><g>2</g><g>3</g></person>
  <person name="paul"><g>2</g><g>3</g><g>4</g></person>
  <person name="ivan"><g>2</g><g>4</g></person>
  <person name="eric"><g>4</g></person>
</doc>
'''
```

Using Java's native XPath support

Here is how to check that groups 2 and 4 are the groups in question using the built-in XPath capabilities of Java 5 and above:

```
import javax.xml.parsers.DocumentBuilderFactory
import javax.xml.xpath.*

xpath = '''
  /doc/person[@name="alan"]/g
  [.=/doc/person[@name="paul"]/g]
  [.=/doc/person[@name="ivan"]/g]
'''

builder = DocumentBuilderFactory.newInstance().newDocumentBuilder()
doc     = builder.parse(new ByteArrayInputStream(xml.bytes))
expr    = XPathFactory.newInstance().newXPath().compile(xpath)
nodes   = expr.evaluate(doc, XPathConstants.NODESET)
assert  nodes.collect { node -> node.textContent } == ['2', '4']
```

And you can refactor your XPath Expression with something like:

```
import javax.xml.parsers.DocumentBuilderFactory
import javax.xml.xpath.*

groupsOf = { name -> "/doc/person[@name='$name']/g" }
xpath    = "${groupsOf 'alan'}[.=${groupsOf 'paul'}][.=${groupsOf 'ivan'}]"
builder  = DocumentBuilderFactory.newInstance().newDocumentBuilder()
doc      = builder.parse(new ByteArrayInputStream(xml.bytes))
expr     = XPathFactory.newInstance().newXPath().compile(xpath)
nodes    = expr.evaluate(doc, XPathConstants.NODESET)
assert   nodes.collect { node -> node.textContent } == ['2', '4']
```

Using GPath

Of course, you can also do this without XPath by using Groovy's GPath facilities as follows:

```
def root = new XmlParser().parseText(xml)
Set groups = root.person.g.collect{ it.text() }
assert groups.findAll{ group ->
  root.person.g.findAll{
    it.text() == group
  }.'..'.'@name'.containsAll(['alan', 'paul', 'ivan'])
} == ['2', '4']
```