2. Looping

There are currently three looping constructs available: **`for`**, **`for-each`** and **`while`**; these are pretty much what one would expect from a scripting language. **Note that these are not intended to run efficient, tight loops**, because there is some processing overhead involved (each child element of the loop is re-instantiated on each iteration). As a benchmark, one can figure that an empty triple loop of 1,000,000 iterations (i 0 to 100, j 0 to 100, k 0 to 100) takes about 50 seconds on a 2 GHz Dual-processor machine.

As of ELF version 0.6 (11/3/2007), the build mechanism underlying Ogrescript has been significantly optimized.

The following changes have occurred:

(1) Introspection is done only once per execution.

`Method` objects are now cached, so that if reflection invocation is repeatedly required (say, for changing values from the environment), the class hierarchies will not have to be repeatedly inspected.

(2) All value-carrying nodes of the graph are now marked for mutability.

Only those which are susceptible to change based on the presence of references to the environment will be reconfigured inside loops.

(3) The element tree is not rebuilt in its entirety inside loops.

Only when the text of an element actually represents that element, that text is mutable, and the element is added to its parent rather than being set as a singleton on it, will that element (and its children) be rebuilt.

(4) As a consequence, invocation of setter or adder methods recurs repeatedly *only* when their values are mutable.

The example below demonstrates the difference between the "static" case (which typifies the majority of code) and the "rebuild" case.

```
<!--
    SAMPLE OUTPUT (should be close to this result)

    start first:      2007/11/03 16:20:20
    start second:     2007/11/03 16:20:28      8 seconds
    finished:         2007/11/03 16:20:52     24 seconds
-->

<ogrescript name="test-collection-to-array">
    <declare name="rep" int="10000" />
    <declare name="t0">
        <date-time type="formatted" />
    </declare>

    <!-- Text is added (like an attribute) to the expression element; consequently neither it nor its parent
needs to be rebuilt -->
    <for var="i" from="0" to="${rep}">
        <declare name="list">
            <list>
                <boolean-expression>${i} == 0</boolean-expression>
                <boolean-expression>${i} == 2</boolean-expression>
                <boolean-expression>${i} == 4</boolean-expression>
                <boolean-expression>${i} == 6</boolean-expression>
                <boolean-expression>${i} == 8</boolean-expression>
                <boolean-expression>${i} == 10</boolean-expression>
            </list>
        </declare>
    </for>

    <!-- Text is actually assignable to the include element (substitutes for it); the entire uri-pattern needs
to be rebuilt -->
    <declare name="t1">
        <date-time type="formatted" />
    </declare>
    <for var="i" from="0" to="${rep}">
        <declare name="list">
            <uri-pattern base="file:/tmp">
                <include>${i}*/0</include>
                <include>${i}*/2</include>
                <include>${i}*/4</include>
                <include>${i}*/6</include>
                <include>${i}*/8</include>
                <include>${i}*/10</include>
            </uri-pattern>
        </declare>
    </for>
    <declare name="t2">
        <date-time type="formatted" />
    </declare>

    <echo message="start first: ${t0}" stdout="true" />
    <echo message="start second: ${t1}" stdout="true" />
    <echo message="finished: ${t2}" stdout="true" />
</ogrescript>
```

Previous to this version, the subtree represented by a for-loop was entirely rebuilt on each call, meaning for every attribute and child, introspection and reflection was carried out for each setter or adder method, as well as fresh instantiation of each child element object.

The above speed-up over 10000 iterations suggests a 3 to 1 gain. Further testing is necessary to ascertain how much efficiency has indeed been acquired by these changes. -alr

---

With reference to the empty triple for-loop mentioned above as taking 50 seconds, the overhead has now been reduced to 1 second.

That is,

```
<for var="i" from="0" to="99">
  <for var="j" from="0" to="99">
     <for var="k" from="0" to="99">
     </for>
  </for>
</for>
```

now takes 1 second instead of 50 to complete.

Moreover, there does not seem to be a significant difference between the two cases below in terms of assignment, because we have cached the method object used to do the invocation of the setter when the values come from the environment. The time it takes to do 1M declarations is mostly in the call to the environment (as well as the addition and removal of frames).

```
<!-- start first:  2007/11/03 19:50:55    -->
<!-- start second: 2007/11/03 19:52:11    -->
<!-- finished:     2007/11/03 19:53:41    -->

<ogrescript name="test-triple-loop">
        <declare name="t0">
                <date-time type="formatted" />
        </declare>
        <for var="i" from="0" to="99">
                <for var="j" from="0" to="99">
                        <for var="k" from="0" to="99">
                                <declare name="a" string="constant" />
                        </for>
                </for>
        </for>
        <declare name="t1">
                <date-time type="formatted" />
        </declare>
        <for var="i" from="0" to="99">
                <for var="j" from="0" to="99">
                        <for var="k" from="0" to="99">
                                <declare name="a" string="(${i},${j},${k})" />
                        </for>
                </for>
        </for>
        <declare name="t2">
                <date-time type="formatted" />
        </declare>
        <echo message="start first: ${t0}" stdout="true" />
        <echo message="start second: ${t1}" stdout="true" />
        <echo message="finished: ${t2}" stdout="true" />
</ogrescript>
```
