6. Ogrescript Gotchas

The main category of problems which may arise for the Ogrescript coder has to do with the non-idempotency obtaining from a more static element tree (the result of the optimizations effected in version 0.6).

Java Object Semantics

1. Objects are no longer re-instantiated/rebuilt unless they have an internal `Collection` to which they add through an `addX` method. If the object has data structures such as a `Map`, they will not be cleared or reconstructed by the Ogrescript engine on multiple calls to execute the containing task (i.e., inside a loop). In such cases, the object or task must be desigined specifically to discard the entries in the structure (probably as the last thing done by the `execute` method).

2. If you have implemented an `addX` or `setX` method which takes a field of the parameter object and uses it as a key to an internal map, be careful of the situation in which that field may change (under looping conditions where its value references the environment). For instance:

```
<for var="i" from="0" to="${max}">
    <declare name="filter" global="true">
       <local-event-filter topic="TROLL-${i}">
           <event-header-comparator>
                <property-comparator comparator="EQUALS">
                    <property name="component-${i}">
                        <value>master</value>
                    </property>
                </property-comparator>
           </event-header-comparator>
       </local-event-filter>
    </declare>
</for>
```

It so happens that the `<property-comparator>`'s method for adding properties takes the property name and uses it as a key to an internal map (e.g., {p.name=p}). This means that the comparator instance (which is static), will have a property with an updated name in the map on each call, but the key will always be *component-0*. The suggested solution here is to use new:

```
<for var="i" from="0" to="${max}">
    <declare name="filter" global="true">
      <new>
         <local-event-filter topic="TROLL-${i}">
           <event-header-comparator>
                <property-comparator comparator="EQUALS">
                    <property name="component-${i}">
                        <value>master</value>
                    </property>
                </property-comparator>
           </event-header-comparator>
         </local-event-filter>
      </new>
    </declare>
</for>
```

NOTE

For a fuller explanation of when objects corresponding to elements in the XML tree are re-instantiated, see loop semantics.

Accumulator Tasks

Tasks which do not meet these internal conditions for being rebuilt, but which may affect other lists or objects with embedded lists referenced from the environment, present a similar problem. Consider the following script:

```
<ogrescript name="test-for-accumulation">
        <declare name="list0" global="true">
                <list/>
        </declare>
        <declare name="bean">
                <test-type/>
        </declare>
        <for var="i" from="0" to="10">
                <declare name="I">
                        <to-string>${i}</to-string>
                </declare>
                <add-to-collection collection="${list0}">
                        <property name="i">
                                <value>${I}</value>
                        </property>
                </add-to-collection>
                <add bean="${bean}" property="testElement">
                        <property name="i">
                                <value>${I}</value>
                        </property>
                </add>
        </for>
        <get bean="${bean}" property="testElements" declare="list1" global="true"/>
        <echo message=" ${list0}" stdout="true" />
        <echo message=" ${list1}" stdout="true" />
</ogrescript>
```

Because the `<value>` element on each `<property>` does not involve an accumulator (it is not added, but merely set, on the property), the `<property>` object itself is not rebuilt at each iteration. Thus the end result of the two lists will be 10 references to the same `<property>` object, and these will all have the 10th value of I.

Wrapping each of the `<property>` elements in a `<new>` element, however, ensures that they will be fully re-configured (meaning their corresponding Java object will be reinstantiated as well):

```
<ogrescript name="test-for-accumulation">
        <declare name="list0" global="true">
                <list/>
        </declare>
        <declare name="bean">
                <test-type/>
        </declare>
        <for var="i" from="0" to="10">
                <declare name="I">
                        <to-string>${i}</to-string>
                </declare>
                <add-to-collection collection="${list0}">
                        <new>
                                <property name="i">
                                        <value>${I}</value>
                                </property>
                        </new>
                </add-to-collection>
                <add bean="${bean}" property="testElement">
                        <new>
                                <property name="i">
                                        <value>${I}</value>
                                </property>
                        </new>
                </add>
        </for>
        <get bean="${bean}" property="testElements" declare="list1" global="true"/>
        <echo message=" ${list0}" stdout="true" />
        <echo message=" ${list1}" stdout="true" />
</ogrescript>
```

NOTE:

<new> is only available as a tag inside of elements which subclass <assign> – see there for further uses. The two tasks shown here (<add> and <add-to-collection>), along with <add-all>, are the only core tasks of this sort which can side-effect collections or objects with collections in the environment.