Storage Upkeep

Introduction

Stores can often get extra elements in them, such as unwanted or abandoned entries. There is a facility for each store that will allow the administrator to specify actions that may be taken. Generally these are in the form of rules that are applied in order. These are inside the store tag in the configuration and apply exactly to that store.

Usage

This facility can be invoked on any store that tracks access patterns. It would not be used on stores managed by the system (such permissions or transactions) since that would interfere with processing. To use this, simply put an upkeep element in the tag for the store. There is a full example of this at the end.

The upkeep DTD

```
<!DOCTYPE upkeep [
  <!ELEMENT upkeep (rule)*>
  <!ATTLIST upkeep
          alarms CDATA #REQUIRED
          debug CDATA #REQUIRED
          enabled CDATA #REQUIRED
          interval CDATA #REQUIRED
          output CDATA #REQUIRED
          runCount CDATA #REQUIRED
          skipVersions CDATA #IMPLIED
          testOnly CDATA #REQUIRED
          verbose CDATA #REQUIRED>
  <!ELEMENT rule (id|date)*>
  <!ATTLIST rule
          action CDATA #REQUIRED
          extends CDATA #IMPLIED
   skipCollateral  CDATA #IMPLIED
          name CDATA #IMPLIED
          skipVersions CDATA #IMPLIED
          verbose CDATA #IMPLIED >
  <!ELEMENT id (#PCDATA)>
  <!ATTLIST id
          regex CDATA #IMPLIED>
  <!ELEMENT date (#PCDATA)>
  <!ATTLIST date
          type CDATA #REQUIRED
          value CDATA #IMPLIED
          when CDATA #REQUIRED>
 ]>
```

Basic structure

The basic structure of this element is

```
<upkeep attributes >
  <rule attributes >
     conditions
  </rule>
  …  more rules
</upkeep>
```

Logical connectives

Each rule is separated logically by and OR. In side each rule, all dates are logically ANDed and all ID tests are ORed.

```
<upkeep>
<rule name="whitelist" action="retain">                        <!-- Rule list -->
    <id>client:/my_ersatz</id>
    <id regex="true"><![CDATA[^localhost.*]]></id>
    <id regex="true">^test:.*</id>
</rule>

<rule name="unused" action="delete">                         <!-- Rule List -->
    <date type="created" when="before" value="6 hr"></date>    <!-- rule entries --
>
    <date type="accessed" when="never"></date>
</rule>
<rule name="abandoned" action="archive">                       <!-- Rule list -->
    <date type="accessed" when="before" value="1 year"></date>
</rule>

<rule name="blacklist" action="delete">
    <id regex="true">^foo.*</id>
    <id regex="true">^delete.*</id>
    <id regex="true">^testScheme.*</id>
</rule>
</upkeep>
```

The logical effect for selecting a store entry to delete is

```
(!whitelist) &&( unused || abandoned || blacklist)
```

and within say, unused,

```
(created <= now - 6 months) && (accessed never)
```

Short-circuit logic

When creating long expressions, keep in mind that *short-circuit* evaluation is used. In this case, the first term that would render the rest of the expression immaterial evaluation halts evaluation. In logical terms.

```
A ∧ B
```

B is not evaluated if A is false

```
A ∨ B
```

B is not evaluated if A is true.

The chief ramification is that if you have a complex list of expressions not all of them will necessarily be evaluated. This is why rules to retain objects should be done before rules to remove objects, *e.g.*, and overly complex lists of rules should be scrutinized.

Element tables

upkeep

Name	Type	Req?	Default	Description
alarms	time list	N	--	Comma delimited list of times of the form hh:mm:ss. E.g. alarms="06:00:00, 18:30:00" runs this daily at those times.
debug		N	false	If debugging *in addition to* server settings
enabled	flag	N	true	If false, this will turn off this entire configuration
interval	string	N	6 hr	A time interval. See time units
output	string	N	--	Full path to a file where the results of each application are sent. The file is in JSON format.
runCount	int	N	--	The number of times for this to run. Each time the facility runs, this is decrements and when all have been used, the facility is disabled. Note that not setting this (the default) means that the facility runs at intervals indefinitely.
skipVersions	flag	N	true	Rules skip any versions. If you set this **false** then versions are subjected to the rules and may end up being deleted as well.
verbose	flag	N	false	Makes output to the logs much chattier. In particular, a summary of each iteration is printed in the logs.
testOnly	flag	N	false	If true then ***NO*** actual operations on the store are done. This overrides every rule. The intent is that you can switch off all actual processing if you need to debug.

Rule

Name	Type	Req?	Default	Description
action	string	Y	--	An action for this rule.
enabled	flag	N	true	enable or disable this. It is overridden by the upkeep flag `testOnly`.
extends	string	N	--	A list of names in order of inheritance. This allows you to set rules and re-use them. Note that this is not a terrible clever facility in that if there is an unknown rule or

				any issue at all an error will be raised. This is on purpose since you do not want some set of rules misapplied an potentially corrupting your store.
name	String	N		A name unique within this upkeep. Note that if no name is given, a random one is generated.
skipCollateral				
skipVersions	flag	N	true	See note above for the upkeep element. This overrides that.
verbose	flag	N	false	This will make output to the log chattier just for this rule if true.

Allowed actions are

- archive = create an archive of the given item and remove the main entry
- delete = delete the entry.
- retain = retain the entry. Note that this will override all other rules to archive or delete subsequently.
- test = only see what actual store entries this rule would apply to.

Extending rules.

You may specify other rules by name in order. This means that if your list is

```
<rule name="my_rule"
      extends="A, B, C"
```

Then all the rules for A are added, then those for B, then those for C and finally any specific rules in my_rule are added.

E.g.

```
<upkeep>
  <rule name="no_ligo"
        enabled="false">
      <!--
           rule applies to everything except clients whose
           ids start with test-ligo:
      -->
      <id regex="true" negate="true">^test-ligo:.*</id>
  <rule>
  <rule name="never_accessed"
        enabled="false">
       <date type="accessed" when="never"/>
  </rule>
  <rule name="no_ligo_last_week"
        extends="no_test,never_accessed">
         <date type="created" when="after" value="1 week"/>
  </rule>
</upkeep>
```

Note that the parent rules are disabled or they will run. This applies to clients that were created one week ago, never accessed and whose client ids do not start with **`test-ligo`**:

Rule entries

There are two main types of rule entries. Date and ID. ID refers to the primary key of the element in the store. Dates are those managed by the system and are for creation time, last accessed time and the last modified (i.e. updated) time.

ID entries

These are conditionals for the identifier. The content of the entry is either the identifier or a regular expression. Matching is either exact or can be done with a regular expression. Attributes are

Name	Type	Req?	Default	Description
negate	flag	N	false	An action for this rule.
regex	flag	N	false	If the value of the element is a regular expression

If you set the negate flag to ***true*** then anything that does *not* match the regex is taken. This can be useful, but do use sparingly at best since it is easy to mis-state a regex rule with negation and delete most of your store.

Examples

```
<id>lh:/dwd-7</id>
<id>client:/my_ersatz</id>
<id regex="true"><![CDATA[[a-zA-Z&&[^whatEver\\s\\d+$]]]]></id>
<id regex="true" negate="true">.*production.*</id>
```

Respectively these check the first two for exact matches. The next shows that CDATA elements can be used if needed when the rule is complex. The last example shows that any identifier that does contain the work **production** should be taken.

Date entries

Supported attributes are

Name	Type	Req?	Default	Description
type	string	Y	--	The type of date, created, accessed or modifiede
value	string	Y	--	Either an ISO 8601 (absolute date) or a time interval. See time units.
when	string	Y	--	When does this apply, before, after or never.

Examples

Here is a pair of conditions for a rule.

```
<rule name="unused" action="delete">
  <date type="created" when="before" value="6 mo"></date>
  <date type="accessed" when="never"></date>
</rule>
```

The meaning is and entry created before 6 months ago and never accessed matches. The rule then specifies these are deleted. The name of the rule is arbitrary, but “unused” seems pretty apt.

Full example

Here is an example from one of our test servers

```
    <mariadb ...>
        <clients>
              <upkeep output="/home/ncsa/temp/report.json"
                      interval="6 hrs.">
                   <rule name="allowlist" action="retain">
                      <id>lh:/dwd-7</id>
                      <id>client:/my_ersatz</id>
                      <id regex="true">^localhost.*</id>
                      <id regex="true">^auto-test:.*</id>
                      <id regex="true">^test:.*</id>
                    </rule>
                    <rule name="unused" action="delete">
                       <date type="created" when="before" value="6 mo"/>
                       <date type="accessed" when="never"/>
                    </rule>
             </upkeep>
          </clients>
  <!-- other stores →
 </mariadb>
```

This writes a report of the results at every run (which is harvested by another program). It runs every 6 hours and starts with a whitelist of testing clients (in this case). These are retained even if the next rule would otherwise flag them for removal. The last rule deletes every other client that is older than 6 months and has never been accessed.