How to migrate from SNMP4J-Agent 2.x to 3.x?

When upgrading from SNMP4J-Agent 2.x to 3.x some changes to your existing code might be necessary, which are described here:

Migration Tasks

Custom ManagedObjects should implement new GenericManagedObject Interface

SNMP4J-Agent 3 introduces a new interface `GenericManagedObject` which needs to be implemented by a `ManagedObject` to be found by the agent (`Command` handler respectively).

The following SNMP4J-Agent classes already implement `GenericManagedObject`:

- `MOScalar`
- `DefaultMOTable`
- `MOSubtreeProxy`
- `StaticMOGroup`

If your class extends one of these, you do not have to change anything. Otherwise, please implement `GenericManagedObject` instead `ManagedObject`.

Do not forget to implement GenericManagedObject interface

If you do not implement GenericManagedObject for your ManagedObject, then it will not be found during MIB walk or for a GET request. It is then not accessible for SNMP requests.

You can use generics even more precisely in your ManagedObject code if you overload the methods of the GenericManagedObject interface like AgentXNode of SNMP4J-AgentX version 3 does:

```
public void prepare(SubRequest<?> request) {
    prepare((SnmpRequest.SnmpSubRequest)request);
}

public void prepare(SnmpRequest.SnmpSubRequest request) {
    addAgentXSet2Queue(request);
    markAsProcessed(request);
}
```

SNMP4J ArgumentParser Interface

- The interface of the `org.snmp4j.util.ArgumentParser` has changed. Especially, the `parse` method now returns `Map<String, List<Object>>` instead `Map<String,List<?>>` in version 2.x.
- You need to change your source code accordingly, although this API change is binary compatible.

MOChangeEvent Granularity Changed

1. **`MOChangeEvent`** in SNMP4J-Agent version 3.0 and later **is fired by `DefaultMOTable` for row level changes too**!
2. In version 2.x and earlier, `MOChangeEvent` was fired on `Variable` updates only. If fired in the prepare phase of a SNMP SET request processing it was "deniable" and if fired after then commit it was not deniable.
3. In version 3.1 and later, in addition to (2.) row level changes (added row, updated, and deleted rows) are fired as `MOChangeEvents` with a special `OidType index`.
4. Note: In version 3.0.x, the new fields `OidType` and `Modification` were not set for `MOChangeEvents` fired on behalf of (2.). In version 3.1 or later `OidType` is always set to `OidType.index` and `Modification` is deducted from the `Variable` change that happened.
5. If you trigger operations like writing MIB table objects to disk that itself modify those tables on row level, then you will need to ignore `MOChangeEvents` while these operations execute - at least you will have to ignore all `MOChangeEvents` with `OidType.index` in order to retain the same behaviour of your existing code as with SNMP4J-Agent 2.x or earlier.

UsmUserTable

The `UsmMIB` implementation of the `UsmUserTable` will no longer store authentication and privacy passphrase in persistent storage and memory! That means any USM users in an agent need to be localised.

Loading Users from Properties File

When creating users in the USM using the PropertyMOInput you now can specify the localisation engine ID as shown in the following example that creates an user using passphrases:

```
snmp4j.agent.cfg.index.1.3.6.1.6.3.15.1.2.2.1.3={o}$#{1.3.6.1.6.3.10.2.1.1.0}.12.'SHA256AES128'
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.3.0={s}SHA256AES128
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.3.1={o}
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.3.2={o}1.3.6.1.6.3.10.1.1.5
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.3.3={s}
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.3.4={s}
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.3.5={o}1.3.6.1.6.3.10.1.2.4
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.3.6={s}
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.3.7={s}
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.3.8={s}
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.3.9={i}4
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.3.10={i}1
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.3.11={s}SHA256AES128AuthPassword
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.3.12={s}SHA256AES128PrivPassword
# Add this user localized without storing the passphrases in the agent persistently
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.3.13={$1.3.6.1.6.3.10.2.1.1.0}
```

Alternatively, the localised keys can be specified directly:

```
# Caution: This user will not work if the agent's engine ID is differs from 80:00:13:70:01:7f:00:00:01:14:e1:a2:
f2
snmp4j.agent.cfg.index.1.3.6.1.6.3.15.1.2.2.1.6={o}$#{1.3.6.1.6.3.10.2.1.1.0}.15.'SHA256AES128key'
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.0={s}SHA256AES128key
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.1={o}
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.2={o}1.3.6.1.6.3.10.1.1.5
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.3={s}
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.4={s}
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.5={o}1.3.6.1.6.3.10.1.2.4
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.6={s}
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.7={s}
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.8={s}
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.9={i}4
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.10={i}1
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.11=
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.12=
# The following columns specify the localisation engine ID and the corresponding localised auth/priv key:
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.13={x}80:00:13:70:01:7f:00:00:01:14:e1:a2:f2
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.14={x}74:d0:40:c6:2a:97:b7:b8:82:68:2a:42:ce:9d:a3:13:7b:99:bc:
94:5b:72:b5:78:ef:b9:a4:6f:1a:17:d7:cc
snmp4j.agent.cfg.value.1.3.6.1.6.3.15.1.2.2.1.6.15={x}e2:64:16:bf:e0:a1:a0:c6:ef:52:a3:64:8e:2d:1a:b1
```

Config files from SNMP4J-Agent v2.x will still work, but it is recommended to explicitly set the localisation engine ID to ensure that the localisation takes place when the configuration is being loaded and not later when the user is actually used first (which could leave the plain text passphrases in memory of the agent).

If you load `USM` users from a properties file ***after*** the agent has been initialised one of the following approaches have to be used to avoid USM data corruption:

1. Remove existing user rows from the USM before loading rows from the properties file.
2. Ensure that you specify all `UsmUserTable` columns from 0 to 15 with values or null (empty value part) to make sure, existing data in the agent's row is consistently overwritten.

Related articles

- How and why to migrate from SNMP4J-Agent DefaultMOPersistenceProvider to MOXodusPersitenceProvider?
- How to migrate from SNMP4J-Agent 2.x to 3.x?