COPPER

Compatibility Rules

for the Migration of Persistent Workflow Instances

Version 1.2, 09.12.2019

Motivation

In case you need to fix a bug or implement a new feature for an existing COPPER workflow, you might face problems when deploying the modified workflow into an environment with still active persistent workflow instances in the COPPER database, if these workflow instances have been instantiated with the old version of that workflow.
There are different options to handle this situation:

1. delete all active instances of the corresponding workflow instances from the database.
2. keep the old workflow unchanged and create a new version of it using COPPERs versioning feature (See COPPER annotation `@WorkflowDescription`). Doing so, the old workflow instances will keep using the old workflow and the new instances will use the new one.
3. migrate the existing workflow instances in the database before deploying the new workflow
4. only apply code modifications to the workflow, that keep it compatible to former versions.

Whereas option 1 and 2 are not allways possible, especially in production environments, the effort for option 3 might be very high.
So, in the following we list some compatibility rules that might help you to change workflows in a way which keeps them compatible to former versions.

Compatibility rules

In common, downward compatible changes of a workflow are code changes, that do not mix up the order of COPPER wait calls in a method and that do not potentially change the stack frames of workflow methods that directly or indirectly call COPPER wait.

Furthermore, when using Javas Object Serialization mechanism for the workflow instance persistence (which is the default), then all compatibility rules to Java Serialization apply. See Java documentation
http://docs.oracle.com/javase/6/docs/platform/serialization/spec/version.html

https://docs.oracle.com/en/java/javase/11/docs/specs/serialization/version.html.

Some downward compatible changes are:

1. **renaming of variable/parameter names**. See source code example `de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_0001`
2. **adding and using a new field**. See source code example `de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_0002`
3. **adding a new method** and using it somewhere/everywhere within the workflow. The new method may not use COPPERs wait directly or indirectly and it may not declare to throw InterruptException. See source code example

de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_0003

4. **Renaming a method**. See source code example
   de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_0004
5. **Changing the implementation of a method**, as long as no COPPER wait calls are mixed up. See source code example
   de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_0005
6. **Adding a new waiting method** and calling it AFTER existing wait calls (directly or indirectly). See source code example
   de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_0006
7. **Adding a new wait call** AFTER existing wait calls (directly or indirectly). See source code example
   de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_0006
8. **Removing an obsolete field**. See source code example
   de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_0007
9. **Removing an obsolete method**. See source code example
   de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_0008

Incompatibility rules

Some downward incompatible changes are:

1. adding a new parameter to a directly or indirectly waiting method. See source code example
   de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_E001
2. adding a new local variable to a directly or indirectly waiting method. See source code example
   de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_E002
3. adding and using a new directly waiting method somewhere before an existing COPPER wait or a method that uses directly or indirectly COPPER wait. See source code example
   de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_E003
4. changing the type of a field. See source code example
   de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_E004
5. changing the serialVersionUID of the workflow class. See source code example
   de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_E005
6. Replacing method calls of methods that directly or indirectly use COPPER wait. See source code example
   de.scoopgmbh.copper.test.versioning.compatibility.check2.CompatibilityCheckWorkflow_E101
7. Adding a field that is not serializable and not transient (when using standard Java serialisation). See source code example
   de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_E006
8. Reordering local variables in a directly or indirectly waiting method. See source code example
   de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_E007
   de.scoopgmbh.copper.test.versioning.compatibility.CompatibilityCheckWorkflow_E008