Quick (20 minute) introduction to the Jakarta EE TCK projects

Scott Marlow

Purpose of EE TCKs (test compatibility kits)

- Validate that Jakarta EE implementations correctly implement the Jakarta EE Specifications (e.g. Plaform, Web Profile, Persistence, CDI, Servlet, ...)
- Are 100% of the Specification requirements tested?
  - No, only the most difficult to implement aspects are tested.
- How long does it take to run the EE TCKs?
  - It would take around 3 days on a single computer with earlier EE versions.
  - It takes around 6-9 hours to run on multiple (CI) virtual machines.

Relationship between the TCK tests and your applications

- How do applications benefit from WildFly passing TCK tests?
  - Ensures that applications written to the EE specs are (somewhat) portable to different Jakarta EE implementations.
  - Why exactly only somewhat portable?
    - Goal is testing the difficult to implement parts of the EE specs.
      - What does that mean?
- Can anyone contribute to the EE TCKs?
  - Good question! See next slide…

Connection from app portability to EE TCKs

- What happens when you discover that an EE implementation like WildFly doesn’t seem to implement an EE specification they way you expect?
    - This can happen for EE SPEC requirements/APIs that do not have a covering TCK test.
    - You can of course file a WildFly bug report for any such situations.
    - You can also contribute a TCK test
        - create a https://github.com/jakartaee/platform-tck pull request to add a test for the not yet covered SPEC requirement.
        - Also sign the Eclipse contributor agreement (https://accounts.eclipse.org/user/eca) using your email address that you associated with your github account.
    - Is it currently easy to add TCK tests?
        - It is hard for Jakarta EE 10 and earlier releases.
        - For EE 11, we are switching to JUnit + Maven for testing which helps.

Deeper dive into TCKs

- Prove that Persistence (EE) container can access third party persistence provider tck/persistence/ee/pluggability/contracts/resource_local/Client.java#L77
  - @testName: createEMF @assertion_ids: PERSISTENCE:JAVADOC:1479; PERSISTENCE:SPEC:981; PERSISTENCE:SPEC:982
  - What is an assertion_id???
  - See map from ^ ID to JavaDoc via PersistenceJavaDocAssertions_2_0.xml
  - See map from ^ ID to Spec via PersistenceSpecAssertions_2_0.xml
    - PERSISTENCE:SPEC:981 = "The interface jakarta.persistence.spi.PersistenceProvider is implemented by the persistence provider"
    - PERSISTENCE:JAVADOC:1479 = "Called by the container when an EntityManagerFactory is to be created."

More about test assertions

- The test assertion ids give an idea of which Jakarta EE Specification or JavaDoc (really SPEC API) requirement is tested by a TCK test.
- It is not yet that easy to locate the reason why a test was added to the TCK.
  - https://github.com/jakartaee/platform-tck/issues/2006 is for adding an TCK AssertionID annotation that can instead be used in newly added tests
  - Perhaps ^ can be based on "Test Assertions Guidelines" http://docs.oasis-open.org/tag/guidelines/v1.0/cn02/guidelines-v1.0-cn02.pdf
  - We will update test assertions to make it easier to deal with.

Resources

- Sign up for the Platform TCK mailing list https://accounts.eclipse.org/mailing-list/jakartaee-tck-dev
- Sign up for other Jakarta EE mailing lists https://jakarta.ee/connect/mailing-lists/
- Platform TCK mailing list https://github.com/jakartaee/platform-tck
- Platform TCK project page https://projects.eclipse.org/projects/ee4j.jakartaee-tck
- Jakarta EE Specifications https://jakarta.ee/specifications/