Jakarta Concurrency TCK Reference Guide

Table of Contents

1. Preface

This guide describes how to download, install, configure, and run the Technology Compatibility Kit (TCK) used to verify the compatibility of an implementation of the Jakarta Concurrency specification.

The specification describes the job specification language, Java programming model, and runtime environment for Jakarta Concurrency applications.

1.1. Licensing

The Jakarta Concurrency TCK is provided under the **Eclipse Foundation Technology Compatibility Kit License - v 1.0** [https://www.eclipse.org/legal/tck.php].

1.2. Who Should Use This Guide

This guide will assist in running the test suite, which verifies implementation compatibility for:

- implementers of Jakarta Concurrency.

1.3. Terminology - "SE mode" vs. "EE mode"

Building on the previous point, it is convenient to use, as shorthand, the term "EE mode" when talking about the TCK constructs and requirements specifically for users running the TCK to certify against the entire EE platform. It is a convenient shorthand term too, then, to use the term "SE mode" for users that are only trying to certify against the Jakarta Concurrency specification, though this term in some ways might be misleading.

Some specifications have a subset of tests that can run in "SE mode" without the requirement of running against an entire EE Platform. The Concurrency TCK, however, runs all tests in "EE mode" and will require a Jakarta EE platform to test against.

1.4. Terminology - "Standalone TCK"

The community will sometimes refer to this TCK as the "standalone" Concurrency TCK. This usage comes from the fact that Jakarta Concurrency is part of the Jakarta EE Platform, which has a platform-level TCK, which we're distinguishing this "standalone" TCK from.

This terminology is confusing, since readers might interpret "standalone" to mean that this TCK can

be run in SE Mode, when in-fact, it must be run in EE Mode. A better term would be `Specification TCK`, but that terminology is not yet being used.

1.5. Terminology - "Test Client" vs "Test Server"

The Concurrency TCK acts as a `Test Client` that will install test applications onto a Jakarta EE Platform Server. The Platform Server will act as a `Test Server` and run tests based on incoming requests from the `Test Client`. Assertions will occur both on the client and server sides.

1.6. Before You Read This Guide

Before reading this guide, you should familiarize yourself with the Jakarta Concurrency Version 3.0.0 specification, which can be found at https://jakarta.ee/specifications/concurrency/3.0/.

Other useful information and links can be found on the eclipse.org project home page for the Jakarta Concurrency project [https://projects.eclipse.org/projects/ee4j.cu] and also at the GitHub repository home for the specification project [https://github.com/jakartaee/concurrency].

2. Major TCK Changes

This version of the Jakarta Concurrency TCK introduces two major changes to the TCK:

1. We change the official execution of the standalone TCK from Ant to Maven. Though the TCK has long been built with Maven and we even have included execution or "runner" Maven modules, our official documentation described an Ant-based execution. This updated version of the TCK Reference Guide details the requirements and procedures for performing an official Maven-based execution of this TCK.
2. We changed from using a proprietary "Test Harness" framework to deploy and test applications on Jakarta EE Platforms, to using the open source Arquillian test framework.

3. What Tests Must I Pass To Certify Compatibility?

3.1. Runtime Tests and Signature Tests Required

To certify compatibility with the entire Jakarta EE Platform (including Jakarta Concurrency), you will need to run the TCK against your implementation and pass 100% of both the:

- TestNG runtime tests
- Signature tests

The two types of tests are encapsulated in a single execution or configuration. This means that the Signature Tests will run alongside all other tests and no additional execution or configuration is required.

By "runtime" tests we simply mean tests simulating Jakarta Concurrency applications running against the Concurrency implementation attempting to certify compatibility. These tests verify that the Concurrency applications behave according to the details defined in the specification, as validated by the TCK test logic.

3.2. Java SE level - Java 11 or Java 17

The JDK used during test execution must be noted and listed as an important component of the certification request. In particular, the Java SE version is important to note, and this version must be used consistently throughout both the TestNG runtime and Signature tests for a given certification request.

For the current TCK version, this can be done with either Java SE Version 11 or Version 17.

4. Prerequisites

4.1. Software To Install

1. **Java/JDK** - Install the JDK you intend to use for this certification request (Java SE Version 11 or Version 17).
2. **Maven** - Install Apache Maven 3.6.0 or higher.
3. **Jakarta EE Platform** - Jakarta EE Application Server or Container [Glassfish, Open Liberty, JBoss, WebLogic, etc.]

4.2. Testing Framework

To better understand how this TCK works, knowing what testing frameworks are being utilized is helpful. Knowledge of how these frameworks operate and interact will help during the project setup.

1. **Arquillian** - Since the EE Platform TCK uses Arquillian to execute tests within an Arquillian "container" for certifying against the EE Platform, you must configure an Arquillian adapter for your target runtime. Version 1.6.0 or later
2. **TestNG** - Since the EE Platform TCK uses TestNG as the entrypoint for tests, and deployments using Arquillian, you must configure a TestNG configuration file to ensure all tests are found and executed.
3. **Signature Test Tool** - No action is needed here, but we note that the signature files were built and should be validated with the Maven plugin with group:artifact:version coordinates: **org.netbeans.tools:sigtest-maven-plugin:1.6**, as used by the sample sigtest runner included in the TCK zip. This is a more specific direction than in earlier releases, in which it was left more open for the user to use a compatible tool. Since there are small differences in the various signature test tools, we standardize on this version.

5. A Guide to the TCK Distribution

This section explains how to obtain the TCK and extract it on your system.

5.1. Obtaining the Software

The Jakarta Concurrency TCK is distributed as a zip file, which contains the TCK artifacts (the test suite binary and source, porting package SPI binary and source, the test suite XML definitions, and signature files) in `/artifacts`, the documentation in `/doc`, and a starter project in `/starter`. You can access the current source code from the Git repository: https://github.com/jakartaee/concurrency.

5.2. The TCK Environment

The software can simply be extracted from the ZIP file. Once the TCK is extracted, you'll see the following structure:

```
concurrency-tck-<version>-dist/
  artifacts/
  doc/
  starter/
  LICENSE
  README.md
```

In more detail:

- `artifacts` contains all the test artifacts pertaining to the TCK: The TCK test classes and source, the TestNG configuration file, a copy of the SignatureTest file for reference, and a script to copy the TCK into local maven repository.
- `doc` contains the documentation for the TCK: this reference guide.
- `starter` a very basic starter maven project to get you started.

5.3. A Quick Tour of the TCK Artifacts

5.3.1. What is included

The Concurrency TCK is a test library that includes four types of packages:

- `ee.jakarta.tck.concurrent.api.*` these are basic API tests that ensure methods throw the correct exceptions and return the valid values.
- `ee.jakarta.tck.concurrent.spec.*` these are more complex SPEC tests that ensure that implementations behave as expected based on the specification.
- `ee.jakarta.tck.concurrent.common.*` these are common packages shared between test packages.
- `ee.jakarta.tck.concurrent.framework` this package is an abstraction layer to make writing tests using TestNG, Arquillian, SigTest, and java.util.logging easier.

TestNG "suite" definition XML files

Here we use the term "suite" informally to describe groups of tests required to pass the TCK (and NOT specifically to refer to any particular "suite" construct defined by the TestNG API).

There are two "suites" included in the TCK. There is a "suite" for the Concurrency API portion of the "Jakarta EE Platform" TCK and there is a second suite for the Concurrency SPI portion of the "Jakarta EE Platform".

These "suites" are both represented within the single `artifacts/suite.xml` file and are identified by their package names:

1. ee.jakarta.tck.concurrent.api.*
2. ee.jakarta.tck.concurrent.spec.*

Note: An implementation **MUST** run against all tests provided in the suite XML file unmodified for an implementation to pass the TCK.

API Signature Files

The one signature file exists for both Java 11 and 17:

1. `artifacts/jakarta.enterprise.concurrent.sig`

Note: This signature file is for reference only. A copy of the signature file is included in the Concurrency TCK test jar.

5.3.2. What is not included

The Concurrency TCK uses but does not provide the necessary application servers, test frameworks, APIs, SPIs, or implementations required to run. It is up to the tester to include those dependencies and set up a test project to run the TCK.

Here is an essential checklist of what you will need, and links to the section that describe how to satisfy these requirements:

- An Application Server to test against | Software To Install
- The Concurrency API, Concurrency TCK, Arquillian, and TestNG libraries available to the `Test Client` | Test Client Dependencies
- The Arquillian, TestNG, Derby JDBC, and Signature Test libraries available to the `Test Server` | Test Server Dependencies
- A TestNG configuration file available to the `Test Client` | Configure TestNG
- An Arquillian SPI implementation for your Application Server | Configure Arquillian
- An Application Server configuration with specific security roles | Configure Application Server
- A logging configuration for TCK logging on `Test Client` and `Test Server` | Configure Logging

6. TCK Test Requirements

Because there is flexibility regarding how a user could use Maven to configure a TCK execution, we make a separate, clear note here of the required number of tests needed to be passed in order to claim compliance via this TCK.

6.1. Runtime tests

For the runtime test (JUnit) component of the TCK:

- 164 tests must be passed to successfully execute the EE TCK suite

***Note:** This count includes the signature test

7. Example runner

This section is dedicated to listing example runners for other implementations to use as a reference on how to configure and use the Concurrency TCK.

Below are links to projects where the Concurrency TCK is being used and run successfully:

- **Open Liberty**: https://github.com/OpenLiberty/open-liberty/tree/release/dev/io.openliberty.jakarta.concurrency.3.0_fat_tck

8. Set up a TCK runner project

A simple maven project is required to control the lifecycle of the Concurrency TCK.

8.1. Test Client Dependencies

The entry point to running the TCK will be on the client-side using TestNG. The Test Client will need to be configured with the dependencies necessary to run the TCK. Some of these dependencies will depend on the application server you are using, and comments have been added to this sample describing the customizations necessary.

Example starter/pom.xml:

```
    <!-- The Arquillian test framework -->
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.jboss.arquillian</groupId>
                <artifactId>arquillian-bom</artifactId>
                <version>${arquillian.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
```

```
        </dependencies>
    </dependencyManagement>

    <!-- Client Dependencies -->
    <dependencies>
        <!-- The TCK -->
        <dependency>
            <groupId>jakarta.enterprise.concurrent</groupId>
            <artifactId>jakarta.enterprise.concurrent-tck</artifactId>
            <version>${jakarta.concurrent.version}</version>
        </dependency>
        <!-- The API -->
        <dependency>
            <groupId>jakarta.enterprise.concurrent</groupId>
            <artifactId>jakarta.enterprise.concurrent-api</artifactId>
            <version>${jakarta.concurrent.version}</version>
        </dependency>
        <!-- Arquillian Implementation for TestNG -->
        <dependency>
            <groupId>org.jboss.arquillian.testng</groupId>
            <artifactId>arquillian-testng-container</artifactId>
            <version>${arquillian.version}</version>
        </dependency>
        <!-- TODO add Arquillian SPI impl for your Jakarta EE Platform -->
        <!-- Arquillian transitive dependency on Servlet -->
        <dependency>
            <groupId>jakarta.servlet</groupId>
            <artifactId>jakarta.servlet-api</artifactId>
            <version>${jakarta.servlet.version}</version>
        </dependency>
        <!-- TestNG -->
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>${testng.version}</version>
        </dependency>
        <!-- Signature Test Plugin -->
        <dependency>
            <groupId>org.netbeans.tools</groupId>
            <artifactId>sigtest-maven-plugin</artifactId>
            <version>${sigtest.version}</version>
        </dependency>
    </dependencies>
```

Each of these Arquillian tests run within the runtime's Servlet container, with the help of an Arquillian adapter for that runtime implementation (mentioned as a prerequisite).

8.2. Test Server Dependencies

If the Test Server is running on a separate JVM (recommended), then the Test Server will also need

access to the TestNG, Signature Test, and the Derby JDBC libraries. The Test Server dependencies can be copied over during the build phase.

Example starter/pom.xml:

```
            <!-- Test Server Dependencies -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-dependency-plugin</artifactId>
                <version>${maven.dep.plugin.version}</version>
                <configuration>
                    <artifactItems>
                        <artifactItem>
                            <groupId>org.testng</groupId>
                            <artifactId>testng</artifactId>
                            <version>${testng.version}</version>
                        </artifactItem>
                        <artifactItem>
                            <groupId>org.apache.derby</groupId>
                            <artifactId>derby</artifactId>
                            <version>${derby.version}</version>
                        </artifactItem>
                        <artifactItem>
                            <groupId>org.netbeans.tools</groupId>
                            <artifactId>sigtest-maven-plugin</artifactId>
                            <version>${sigtest.version}</version>
                        </artifactItem>
                    </artifactItems>
                    <outputDirectory>${application.server.lib}</outputDirectory>
                </configuration>
            </plugin>
```

Using the maven command:

```
$ mvn dependency:copy
```

8.3. Configure TestNG

TestNG needs to be configured to know which packages contain tests to run. This configuration is done via a configuration file. The TestNG configuration file has been provided in the `artifacts/` and `starter/` directories.

In order for your maven project to execute these tests the surefire plugin needs to be configured.

Example starter/pom.xml:

```
            <!-- Surefire plugin - Entrypoint for TestNG -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>${maven.surefire.plugin.version}</version>
                <configuration>
                    <suiteXmlFiles>
                        <suiteXmlFile>${suiteXmlFile}</suiteXmlFile>
                    </suiteXmlFiles>
                    <!-- Ensure surfire plugin looks under src/main/java instead of
src/test/java -->

<testSourceDirectory>${basedir}${file.separarator}src${file.separarator}main${file.sep
ararator}java${file.separarator}</testSourceDirectory>
                </configuration>
            </plugin>
```

8.4. Configure Arquillian

Application Servers that implement the Arquillian SPI use a configuration file to define properties, such as hostname, port, username, password, etc. These properties will allow Arquillian to connect to the application server, install applications, and get test responses. An Arquillian configuration file has been provided in the `starter/` directory.

8.5. Configure Application Server

The Concurrency TCK uses default objects (`java:comp/DefaultManagedExecutorService`) and annotation-based configurations (`@ManagedExecutorDefinition`) to ensure that application servers require minimal customization to run the TCK.

However, the Concurrency TCK does require that Application Servers define a security context for security-based tests.

- Username: javajoe
- Password: javajoe
- Group: Manager

The TCK uses external libraries that also need to be available on the Application Server's class path.

- org.apache.derby:derby - For database testing
- org.testng:testng - For test assertions
- org.netbeans.tools:sigtest-maven-plugin - For signature testing

See Test Server Dependencies

8.6. Configure Logging

The Concurrency TCK uses `java.util.logging` for logging debug messages, and to output test results in some cases. Registered loggers exist both on the Test Client and Test Server meaning you will need to configure both sides to enable logging. This is done by pointing the JVM to the logging configuration file using the property. An example logging configuration file has been provided under the `/starter` directory.

To enable logging for the Client side of tests, add a system property to the surefire plugin:

Example starter/pom.xml:

```
                    <systemProperties>
                        <property>
                            <name>java.util.logging.config.file</name>
                            <value>${logging.config}</value>
                        </property>
                    </systemProperties>
```

To enable logging for the Server side of tests, set the same system property on the JVM running your application server.

8.7. Advanced Configuration

Some application servers may have custom deployment descriptors that they would like to include as part of the applications that are being deployed to their server. The custom deployment descriptors can be included in a programmatic way using ShrinkWrap and the Arquillian SPI.

Example ApplicationArchiveProcessor:

```
public class MyApplicationArchiveProcessor implements ApplicationArchiveProcessor {
    List<String> appNames;

    @Override
    public void process(Archive<?> archive, TestClass testClass) {
        if(appNames.contains(archive.getName())){
            ((WebArchive) archive).addAsWebInfResource("my-custom-sun-web.xml", "sun-
web.xml");
        }
    }
```

Example LoadableExtension:

```
public class MyLoadableExtension implements LoadableExtension {
    @Override
    public void register(ExtensionBuilder extensionBuilder) {
        extensionBuilder.service(ApplicationArchiveProcessor.class,
MyApplicationArchiveProcessor.class);
    }
}
```

Example META-INF/services/org.jboss.arquillian.core.spi.LoadableExtension:

```
my.custom.test.package.MyLoadableExtension
```

9. Running the TCK

Once the TCK Runner project is created and configured the Concurrency TCK is run as part of the maven test lifecycle.

```
$ cd starter
$ mvn clean test
```

9.1. Expected Output

Here is example output when we, as in the starter runner, run successfully:

```
    $ mvn clean test
    ...

    -------------------------------------------------------
     T E S T S
    -------------------------------------------------------
    Running TestSuite
    ...
    ... TestNG 6.14.3 by Cédric Beust (cedric@beust.com)
    ...

    Tests run: 162, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 410.985 sec - in
TestSuite

    Results :

    Tests run: 162, Failures: 0, Errors: 0, Skipped: 0

    16:05:47,343 [INFO]
    16:05:47,344 [INFO] ------<
io.openliberty.jakarta.enterprise.concurrent:tck.runner >-------
    16:05:47,345 [INFO] Building Jakarta Concurrency TCK Runner 1.0-SNAPSHOT
[2/2]
    16:05:47,346 [INFO] --------------------------------[ pom
]---------------------------------
    16:05:47,351 [INFO]
    16:05:47,351 [INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ tck.runner
---
    16:05:47,359 [INFO]
------------------------------------------------------------------------
    16:05:47,380 [INFO] Reactor Summary for Jakarta Concurrency TCK Runner 1.0-
SNAPSHOT:
    16:05:47,381 [INFO]
    16:05:47,382 [INFO] Jakarta Concurrency TCK Runner TCK Module .......... SUCCESS
[06:55 min]
    16:05:47,383 [INFO] Jakarta Concurrency TCK Runner ..................... SUCCESS [
0.017 s]
    16:05:47,384 [INFO]
------------------------------------------------------------------------
    16:05:47,385 [INFO] BUILD SUCCESS
    16:05:47,385 [INFO]
------------------------------------------------------------------------
    16:05:47,386 [INFO] Total time:  06:55 min
    16:05:47,387 [INFO] Finished at: 2022-03-30T16:05:47-05:00
    16:05:47,387 [INFO]
------------------------------------------------------------------------
```

10. Signature Tests

The signature tests validate the integrity of the `jakarta.enterprise.concurrent` Java "namespace" (or "package prefix") of the Concurrency implementation. This would be especially important for an implementation packaging its own API JAR in which the API must be validated in its entirety. For implementations expecting their users to rely on the API released by the Jakarta Concurrency specification project (e.g. to Maven Central) the signature tests are also important to validate that improper (non-spec-defined) extensions have not been added to `jakarta.enterprise.concurrent.*` packages/classes/etc.

10.1. Running signature tests

The Concurrency TCK will run signature tests on the application server itself, and not as part of a separate plugin / execution. This means that the signature tests will run during the maven `test` phase.

You need to configure your application server with a JVM property `-Djimage.dir=<path-your-server-has-access-to>`. This is because, when running the signature tests on JDK 9+ we need to convert the JDK modules back into class files for signature testing.

The signature test plugin we use will also attempt to perform reflective access of classes, methods, and fields. Due to the new module system in JDK 9+ special permissions need to be added in order for these tests to run:

If you are using a Security Manager add the following permissions to the `sigtest-maven-plugin` on your application server:

```
permission java.lang.RuntimePermission "accessClassInPackage.jdk.internal";
permission java.lang.RuntimePermission "accessClassInPackage.jdk.internal.reflect";
permission java.lang.RuntimePermission
"accessClassInPackage.jdk.internal.vm.annotation";
```

By default the java.base module only exposes certain classes for reflective access. Therefore, the Concurrency TCK test will need access to the `jdk.internal.vm.annotation` class. To give the `sigtest-maven-plugin` access to this class set the following JVM properties:

```
--add-exports java.base/jdk.internal.vm.annotation=ALL-UNNAMED
--add-opens java.base/jdk.internal.vm.annotation=ALL-UNNAMED
```

Some JDKs will mistake the space in the prior JVM properties as delimiters between properties In this case use:

```
--add-exports=java.base/jdk.internal.vm.annotation=ALL-UNNAMED
--add-opens=java.base/jdk.internal.vm.annotation=ALL-UNNAMED
```

For more information about generating the signature test file, and how the test run read: [ee.jakarta.tck.concurrent.framework.signaturetest/README.md](https://github.com/jakartaee/concurrency/blob/master/tck/src/main/java/ee/jakarta/tck/concurrent/framework/signaturetest/README.md)

As mentioned in the prerequisite section the signature file formats across the various signature test tools have diverged and this test suite uses the Maven plugin with group:artifact:version coordinates: **org.netbeans.tools:sigtest-maven-plugin:1.6**.

10.2. Expected output

The the Signature Test plugin will log output to `System.out`. Whereas, the Signature Test framework we use to set up the test will log using `java.util.logging` so you may see these logs output to two separate locations depending on your application server.

```
[3/30/22, 16:01:03:250 CDT] 00000045 TestServlet        I --> testSignatures
[3/30/22, 16:01:12:618 CDT] 00000045 SystemOut          O ********** BEGIN PACKAGE
LEVEL SIGNATURE VALIDATION **********
[3/30/22, 16:01:12:618 CDT] 00000045 SystemOut          O ********** BEGIN VALIDATE
PACKAGE 'jakarta.enterprise.concurrent' **********
[3/30/22, 16:01:12:618 CDT] 00000045 SystemOut          O ********** VALIDATE IN
STATIC MODE - TO CHECK CONSANT VALUES ****
[3/30/22, 16:01:12:619 CDT] 00000045 SystemOut          O Static mode supports checks
of static constants values
[3/30/22, 16:01:12:622 CDT] 00000045 SystemOut          O Setting static mode flag to
allow constant checking.
...
[3/30/22, 16:01:13:392 CDT] 00000045 SystemOut          O ********** Status Report
'jakarta.enterprise.concurrent' **********
[3/30/22, 16:01:13:392 CDT] 00000045 SystemOut          O SignatureTest report
Base version: 3.0.0-SNAPSHOT
Tested version:
Check mode: src [throws normalized]
Constant checking: on

Warning: The return type java.util.concurrent.Future<{%%0}> can't be resolved
Warning: The return type java.util.concurrent.Future<{%%0}> can't be resolved
Warning: The return type java.util.concurrent.Future<?> can't be resolved

[3/30/22, 16:01:13:392 CDT] 00000045 SystemOut          O ********** Package
'jakarta.enterprise.concurrent' - PASSED (STATIC MODE) **********
[3/30/22, 16:01:13:393 CDT] 00000045 SystemOut          O ********** VALIDATE IN
REFLECTIVE MODE  ****
[3/30/22, 16:01:13:393 CDT] 00000045 SystemOut          O Reflective mode supports
verification within containers (ie ejb, servlet, etc)
[3/30/22, 16:01:13:394 CDT] 00000045 SystemOut          O Not Setting static mode flag
to allow constant checking.
[3/30/22, 16:01:13:404 CDT] 00000045 SystemOut          O Calling:
com.sun.tdk.signaturetest.SignatureTest() with following args:
```

```
...
[3/30/22, 16:01:13:892 CDT] 00000045 SystemOut          O ********** Status Report
'jakarta.enterprise.concurrent' **********
[3/30/22, 16:01:13:892 CDT] 00000045 SystemOut          O SignatureTest report
Base version: 3.0.0-SNAPSHOT
Tested version:
Check mode: src [throws normalized]
Constant checking: on

Warning: The return type java.util.concurrent.Future<{%%0}> can't be resolved
Warning: The return type java.util.concurrent.Future<{%%0}> can't be resolved
Warning: The return type java.util.concurrent.Future<?> can't be resolved

[3/30/22, 16:01:13:893 CDT] 00000045 SystemOut          O ********** Package
'jakarta.enterprise.concurrent' - PASSED (REFLECTION MODE) **********
[3/30/22, 16:01:13:893 CDT] 00000045 SystemOut          O ********** END VALIDATE
PACKAGE 'jakarta.enterprise.concurrent' **********
[3/30/22, 16:01:13:893 CDT] 00000045 SystemOut          O ********** BEGIN VALIDATE
PACKAGE 'jakarta.enterprise.concurrent.spi' **********
[3/30/22, 16:01:13:893 CDT] 00000045 SystemOut          O ********** VALIDATE IN
STATIC MODE - TO CHECK CONSANT VALUES ****
[3/30/22, 16:01:13:893 CDT] 00000045 SystemOut          O Static mode supports checks
of static constants values
[3/30/22, 16:01:13:894 CDT] 00000045 SystemOut          O Setting static mode flag to
allow constant checking.
[3/30/22, 16:01:13:909 CDT] 00000045 SystemOut          O Calling:
com.sun.tdk.signaturetest.SignatureTest() with following args:
...
[3/30/22, 16:01:14:292 CDT] 00000045 SystemOut          O ********** Status Report
'jakarta.enterprise.concurrent.spi' **********
[3/30/22, 16:01:14:293 CDT] 00000045 SystemOut          O SignatureTest report
Base version: 3.0.0-SNAPSHOT
Tested version:
Check mode: src [throws normalized]
Constant checking: on

Warning: The return type java.util.concurrent.Future<{%%0}> can't be resolved
Warning: The return type java.util.concurrent.Future<{%%0}> can't be resolved
Warning: The return type java.util.concurrent.Future<?> can't be resolved

[3/30/22, 16:01:14:293 CDT] 00000045 SystemOut          O ********** Package
'jakarta.enterprise.concurrent.spi' - PASSED (STATIC MODE) **********
[3/30/22, 16:01:14:293 CDT] 00000045 SystemOut          O ********** VALIDATE IN
REFLECTIVE MODE  ****
[3/30/22, 16:01:14:293 CDT] 00000045 SystemOut          O Reflective mode supports
verification within containers (ie ejb, servlet, etc)
[3/30/22, 16:01:14:294 CDT] 00000045 SystemOut          O Not Setting static mode flag
to allow constant checking.
[3/30/22, 16:01:14:301 CDT] 00000045 SystemOut          O Calling:
com.sun.tdk.signaturetest.SignatureTest() with following args:
...
```

```
[3/30/22, 16:01:14:616 CDT] 00000045 SystemOut          O ********** Status Report
'jakarta.enterprise.concurrent.spi' **********
[3/30/22, 16:01:14:616 CDT] 00000045 SystemOut          O SignatureTest report
Base version: 3.0.0-SNAPSHOT
Tested version:
Check mode: src [throws normalized]
Constant checking: on

Warning: The return type java.util.concurrent.Future<{%%0}> can't be resolved
Warning: The return type java.util.concurrent.Future<{%%0}> can't be resolved
Warning: The return type java.util.concurrent.Future<?> can't be resolved

[3/30/22, 16:01:14:617 CDT] 00000045 SystemOut          O ********** Package
'jakarta.enterprise.concurrent.spi' - PASSED (REFLECTION MODE) **********
[3/30/22, 16:01:14:617 CDT] 00000045 SystemOut          O ********** END VALIDATE
PACKAGE 'jakarta.enterprise.concurrent.spi' **********
[3/30/22, 16:01:14:618 CDT] 00000045 ConcurrencySigTest I
*****************************************************
All package signatures passed.
    Passed packages listed below:
        jakarta.enterprise.concurrent(static mode)
        jakarta.enterprise.concurrent(reflection mode)
        jakarta.enterprise.concurrent.spi(static mode)
        jakarta.enterprise.concurrent.spi(reflection mode)
*****************************************************
[3/30/22, 16:01:14:620 CDT] 00000045 TestServlet
I <-- testSignatures
```

11. TCK Challenges/Appeals Process

The Jakarta EE TCK Process 1.1 will govern all process details used for challenges to the Jakarta Concurrency TCK.

Except from the **Jakarta EE TCK Process 1.1**:

> Specifications are the sole source of truth and considered overruling to the TCK in all senses. In the course of implementing a specification and attempting to pass the TCK, implementations may come to the conclusion that one or more tests or assertions do not conform to the specification, and therefore MUST be excluded from the certification requirements.

> Requests for tests to be excluded are referred to as Challenges. This section identifies who can make challenges to the TCK, what challenges to the TCK may be submitted, how these challenges are submitted, how and to whom challenges are addressed.

11.1. Filing a Challenge

The challenge process is defined within the Challenges section within the **Jakarta EE TCK Process 1.1**.

Challenges will be tracked via the issues of the Jakarta Concurrency Specification repository.

As a shortcut through the challenge process mentioned in the **Jakarta EE TCK Process 1.1** you can click here, though it is recommended that you read through the challenge process to understand it in detail.

12. Certification of Compatibility

The Jakarta EE TCK Process 1.1 will define the core process details used to certify compatibility with the Jakarta Concurrency specification, through execution of the Jakarta Concurrency TCK.

Except from the **Jakarta EE TCK Process 1.1**:

> Jakarta EE is a self-certification ecosystem. If you wish to have your implementation listed on the official https://jakarta.ee implementations page for the given specification, a certification request as defined in this section is required.

12.1. Filing a Certification Request

The certification of compatibility process is defined within the Certification of Compatibility section within the **Jakarta EE TCK Process 1.1**.

Certifications will be tracked via the issues of the Jakarta Concurrency Specification repository.

As a shortcut through the certification of compatibility process mentioned in the **Jakarta EE TCK Process 1.1** you can click here, though it is recommended that you read through the certification process to understand it in detail.

13. Rules for Jakarta Concurrency Products

The following rules apply for each version of an operating system, software component, and hardware platform Documented as supporting the Product:

- **Concurrency1** The Product must be able to satisfy all applicable compatibility requirements, including passing all Conformance Tests, in every Product Configuration and in every combination of Product Configurations, except only as specifically exempted by these Rules. For example, if a Product provides distinct Operating Modes to optimize performance, then that Product must satisfy all applicable compatibility requirements for a Product in each Product Configuration, and combination of Product Configurations, of those Operating Modes.
- **Concurrency1.1** If an Operating Mode controls a Resource necessary for the basic execution of

the Test Suite, testing may always use a Product Configuration of that Operating Mode providing that Resource, even if other Product Configurations do not provide that Resource. Notwithstanding such exceptions, each Product must have at least one set of Product Configurations of such Operating Modes that is able to pass all the Conformance Tests. For example, a Product with an Operating Mode that controls a security policy (i.e., Security Resource) which has one or more Product Configurations that cause Conformance Tests to fail may be tested using a Product Configuration that allows all Conformance Tests to pass.

- **Concurrency1.2** A Product Configuration of an Operating Mode that causes the Product to report only version, usage, or diagnostic information is exempted from these compatibility rules.
- **Concurrency1.3** An API Definition Product is exempt from all functional testing requirements defined here, except the signature tests.
- **Concurrency2** Some Conformance Tests may have properties that may be changed. Properties that can be changed are identified in the configuration interview. Properties that can be changed are identified in the JavaTest Environment (.jte) files in the Test Suite installation. Apart from changing such properties and other allowed modifications described in this User's Guide (if any), no source or binary code for a Conformance Test may be altered in any way without prior written permission. Any such allowed alterations to the Conformance Tests will be provided via the Jakarta EE Specification Project website and apply to all vendor compatible implementations.
- **Concurrency3** The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.
- **Concurrency4** The Exclude List associated with the Test Suite cannot be modified.
- **Concurrency5** The Maintenance Lead can define exceptions to these Rules. Such exceptions would be made available as above, and will apply to all vendor implementations.
- **Concurrency6** All hardware and software component additions, deletions, and modifications to a Documented supporting hardware/software platform, that are not part of the Product but required for the Product to satisfy the compatibility requirements, must be Documented and available to users of the Product. For example, if a patch to a particular version of a supporting operating system is required for the Product to pass the Conformance Tests, that patch must be Documented and available to users of the Product.
- **Concurrency7** The Product must contain the full set of public and protected classes and interfaces for all the Libraries. Those classes and interfaces must contain exactly the set of public and protected methods, constructors, and fields defined by the Specifications for those Libraries. No subsetting, supersetting, or modifications of the public and protected API of the Libraries are allowed except only as specifically exempted by these Rules.
- **Concurrency7.1** If a Product includes Technologies in addition to the Technology Under Test, then it must contain the full set of combined public and protected classes and interfaces. The API of the Product must contain the union of the included Technologies. No further modifications to the APIs of the included Technologies are allowed.
- **Concurrency8** Except for tests specifically required by this TCK to be rebuilt (if any), the binary Conformance Tests supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

- **Concurrency9** The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.

14. Links

- Jakarta Concurrency TCK repository - https://github.com/jakartaee/concurrency
- Jakarta Concurrency specification/API repository - https://github.com/jakartaee/concurrency
- Jakarta Concurrency project home page - https://projects.eclipse.org/projects/ee4j.cu
- Arquillian and ShrinkWrap doc: https://arquillian.org/guides/shrinkwrap_introduction/