# Jakarta Batch TCK Reference Guide

# Table of Contents


1. Preface . . . 1
   1.1. Licensing . . . 1
   1.2. Who Should Use This Guide . . . 1
   1.3. Before You Read This Guide . . . 1
2. Introduction . . . 1
   2.1. What Tests Do I Need To Pass (to pass the TCK)? . . . 1
3. TCK Challenges/Appeals Process . . . 1
   3.1. Links . . . 2
4. Certification of Compatibility . . . 2
   4.1. Links . . . 2
5. Installation . . . 3
   5.1. Obtaining the Software . . . 3
   5.2. The TCK Environment . . . 3
   5.3. TCK test classes . . . 4
6. Configuration . . . 4
   6.1. TCK Properties . . . 4
   6.2. Porting Package SPI . . . 5
   6.3. Configuring TestNG to run the TCK . . . 5
7. Executing Signature Tests . . . 5
   7.1. Obtaining a signature test tool . . . 6
   7.2. JDK/JRE prerequisite . . . 6
   7.3. Other included prereqs . . . 6
   7.4. Running the Signature Tests . . . 6
   7.5. Determining success . . . 8
   7.6. Forcing a Signature Test failure (optional) . . . 8
   7.7. Ant script (optional) . . . 8
   7.8. Gotchas . . . 9
8. Executing TestNG Test Suite . . . 9
   8.1. Steps . . . 9
   8.2. Timeouts . . . 9
   8.3. Building the TCK (optional, for reference): . . . 10
9. Working with TCK source (debugging, etc.) . . . 10
10. Note on TCK guide format . . . 10
11. Links . . . 10
12. Change History . . . 11
   12.1. Initial Release . . . 11

# 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 Batch specification.

## 1.1. Licensing

The Jakarta Batch 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 is for implementers of the Jakarta Batch specification, to assist in running the test suite that verifies the compatibility of their implementation.

## 1.3. Before You Read This Guide

Before reading this guide, you should familiarize yourself with the Jakarta Batch Version 1.0 specification, which can be found at https://jakarta.ee/specifications/batch/1.0/.

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

# 2. Introduction

The Jakarta Batch TCK tests implementations of the Jakarta Batch specification, which describes the job specification language, Java programming model, and runtime environment for Jakarta Batch applications.

## 2.1. What Tests Do I Need To Pass (to pass the TCK)?

As an overview, note in order to pass the Jakarta Batch TCK you must run against your implementation, passing 100% of both the:

- Signature Tests
- TestNG Test Suite

The two types of tests are not encapsulated in a single execution task or command; they must be executed separately from each other and each must be executed separately for each version of Java tested (e.g. Java 8, etc.).

# 3. TCK Challenges/Appeals Process

The Jakarta EE TCK Process 1.0 will govern all process details used for challenges to the Jakarta

Batch TCK.

Except from the **Jakarta EE TCK Process 1.0**:

> 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.

## 3.1. Links

Here is a link to the Challenges section within the **Jakarta EE TCK Process 1.0**.

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

As a shortcut through the challenge process mentioned in the **Jakarta EE TCK Process 1.0** you can click here, though it is recommended that you read through the challenge process to understand it in detail.

# 4. Certification of Compatibility

The **Jakarta EE TCK Process 1.0** will define the core process details used to certify compatibility with the Jakarta Batch specification, through execution of the Jakarta Batch TCK.

Except from the **Jakarta EE TCK Process 1.0**:

> 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.

## 4.1. Links

Here is a link to the Certification of Compatibility section within the **Jakarta EE TCK Process 1.0**.

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

As a shortcut through the challenge process mentioned in the **Jakarta EE TCK Process 1.0** you can click here, though it is recommended that you read through the certification process to understand it in detail.

# 5. Installation

This section explains how to obtain the TCK and provides recommendations for how to install/extract it on your system.

## 5.1. Obtaining the Software

The Jakarta Batch 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 descriptor) in `/artifacts`, the TCK library dependencies in `/lib` and documentation in `/doc`. You can download the current source code from the Git repository: https://github.com/eclipse-ee4j/batch-tck.

## 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:

```
jakarta.batch.official.tck-x.y.z/
    artifacts/
    doc/
    lib/
    build.xml
    sigtest.build.xml
    batch-tck.properties
    batch-sigtest-tck.properties
    LICENSE_EFTL.md
    NOTICE.md
    README.md
```

In more detail:

`artifacts` contains all the test artifacts pertaining to the TCK: The TCK test classes and source, the TCK SPI classes and source, the TestNG suite.xml file and the SigTest signature files.

`doc` contains the documentation for the TCK (this reference guide)

`lib` contains the necessary prereqs for the TCK

`build.xml`, `sigtest.build.xml` Ant build files used to run TestNG, signature test portions of the TCK

`batch-tck.properties`, `batch-sigtest-tck.properties` Specify properties here for each of the TestNG, signature test portions of the TCK, respectively

(And the remaining text files are self-explanatory.)

## 5.3. TCK test classes

The TCK test methods are contained in a number of test classes in the `com.ibm.jbatch.tck.tests` package. Each test method is flagged as a TestNG test using the `@org.testng.annotations.Test` annotation.

===TCK test artifacts Besides the test classes themselves, the Jakarta Batch TCK is comprised of a number of test artifact classes located in the `com.ibm.jbatch.tck.artifacts` package. These are the batch artifacts that have been implemented based on the Jakarta Batch API, and which are used by the individual test methods. The final set of test artifacts is the set of test JSL (XML) files, which are packaged in the `META-INF/batch-jobs` directory within `artifacts/com.ibm.jbatch.tck-x.y.z.jar`

The basic test flow simply involves a TestNG test method using the JobOperator API to start (and possibly restart) one or more job instances of jobs defined via one of the test JSLs, making use of some number of `com.ibm.jbatch.tck.artifacts` Java artifacts. The JobOperator is wrapped by a thin layer which blocks waiting for the job to finish executing (more on this in the discussion of the **porting package SPI** later in the document).

# 6. Configuration

## 6.1. TCK Properties

In order to run the TCK, you must define a property pointing to the Jakarta Batch runtime implementation that you are running the TCK against.

### 6.1.1. Required Properties

You will need to set one required property, **batch.impl.classes** prior to running the Jakarta Batch TCK. This property is defined in the `batch-tck.properties` as follows:

Example:

```
# Edit this property to contain a classpath listing of the directories and jars for
the SE Jakarta Batch runtime implementation (that you're running the TCK against)
# For example:
```

```
batch.impl.classes=$HOME/foo/lib/classes:$HOME/foo/lib/foo.jar:$HOME/foo/lib/batch-
api.jar
```

### 6.1.2. Optional JVM Argument Property

An optional property with name **jvm.options** is provided to specify JVM arguments using the `TestNG <jvmarg line=""/>` function: This property value should list the JVM arguments, separated by spaces.

### 6.1.3. Optional Properties for Tuning Wait Times

Finally, some of the TCK tests sleep for a short period of time to allow an operation to complete or to force a timeout. These wait times are defaulted via properties that are also specified in batch-tck.properties.

As with many typical decisions regarding timeout values, we attempt to strike a good balance between failing quickly when appropriate but allowing legitimate work to complete.

These values can be adjusted if timing issues are seen in the implementation being tested. Refer to the documentation for a specific test (i.e. the comments in the test source) as to how the time value is used for that test.

## 6.2. Porting Package SPI

The Jakarta Batch TCK relies on an implementation of a "porting package" SPI to function, in order to verify test execution results. The reason is that the Jakarta Batch specification API alone does not provide a convenient-enough mechanism to check results.

A default, "polling" implementation of this SPI is shipped within the TCK itself. The expectation is that the typical Jakarta Batch implementation will be content to use the TCK-provided, default implementation of the porting package SPI.

Further detail on the porting package is provided later in this document, in case you wish to provide your own, different implementation.

## 6.3. Configuring TestNG to run the TCK

TestNG is responsible for selecting the tests to execute, the order of execution, and reporting the results. Detailed TestNG documentation can be found at testng.org [http://testng.org/doc/documentation-main.html]. One reason TestNG was chosen was the ability to use a single XML file to hold excludes from a set of compiled tests, and to easily add to this exclude list in the event of TCK challenges.

The `artifacts/batch-tck-impl-SE-suite.xml` artifact provided in the TCK distribution must be run by TestNG 6.8.8 (described by the TestNG documentation as "with a testng.xml file") unmodified for an implementation to pass the TCK.

(**Note:** for debugging purposes, however, it may be convenient to use this file to allow tests to be excluded from a run, e.g. to run a single test method.).

# 7. Executing Signature Tests

One of the requirements of an implementation passing the TCK is for it to pass the signature test. This section describes how to run the signature test against your implementation.

## 7.1. Obtaining a signature test tool

We do not prescribe a certain version/distribution of signature test library. In testing the TCK (**in the com.ibm.jbatch.tck.dist.exec module**), we use the version of `sigtestdev.jar` released to Maven Central under coordinates **net.java.sigtest:sigtestdev:3.0-b12-v20140219** (the JAR is here), in spite of the fact that the POM comments mention that this is an "unofficial" release.

Some alternate suggestions:

1. The sigtestdev.jar version used by the Jakarta EE TCK project.
2. A distribution from the sigtest project, an OpenJDK project.

It is assumed all these options will give similar results.

## 7.2. JDK/JRE prerequisite

The official run of the signature tests must be performed with an Open JDK with HotSpot VM, using a distribution matching the Java version being tested (e.g. Java 8).

Note also that informal runs against certain JDK/JRE distributions may fail, simply because the layout of the JVM internals differs from what the sigtest tooling expects, (and not because of a signature mismatch or other Java language issue).

## 7.3. Other included prereqs

The other prereqs needed for the signature tests are included by the TCK distribution:

- an implementation of class jakarta.enterprise.util.Nonbinding - provided by the CDI API JAR.
- the `jakarta.inject.*`` classes

## 7.4. Running the Signature Tests

The TCK package contains signature files (e.g. `batch-api-sigtest-java8.sig`) in the `artifacts` directory.

Run the signature test by executing a command like the following:

```
java -jar $SIGTEST_DEV_JAR SignatureTest -static -package jakarta.batch \
-filename batch-api-sigtest-java8.sig -classpath \
$JAVA_RUNTIME_JAR:$jakarta_INJECT_JAR:$jakarta_ENTERPRISE_UTIL_JAR: \
$MY_BATCH_API_JAR
```

Note the four dependencies plus JDK/JRE here, the locations of which you may need to modify:

- JAVA_RUNTIME_JAR: the location of the `rt.jar` from your JDK/JRE running the 'java' executable here. (It may be `$JAVA_HOME/lib/rt.jar` or `$JAVA_HOME/jre/lib/rt.jar`)

- SIGTEST_DEV_JAR: the location of 'sigtestdev.jar' from your sigtest download.
- jakarta_INJECT_JAR: (for class jakarta.inject.Qualifier, shipped with TCK)
- jakarta_ENTERPRISE_UTIL_JAR: (for class jakarta.enterprise.util.Nonbinding, shipped with TCK)
- MY_BATCH_API_JAR: Your own API JAR from your own implementation, which you are running the signature test against.

### 7.4.1. Example Execution

Here is an example showing a sample set of values for the shell variables used in the shorthand above.

It assumes:

1. You have unzipped the TCK into the present working directory.
2. You have copied into the working directory's parent directory each of:
   - the sigtest tool `sigtestdev.jar`
   - The Jakarta Batch API JAR under test `jakarta.batch-api-2.0.0-M2.jar`
3. Your JRE distribution has the runtime JAR `rt.jar` at location `$JAVA_HOME/jre/lib/rt.jar`.
4. Your 'java' executable and your 'rt.jar' come from a Java 8 JDK/JRE, since in the example you are running against the Java 8 signature file (based on the -filename argument)

```
java -jar ../sigtestdev.jar SignatureTest -static -package jakarta.batch \
-filename artifacts/batch-api-sigtest-java8.sig \
-classpath ../jakarta.batch-api-2.0.0-
M2.jar:$JAVA_HOME/jre/lib/rt.jar:lib/jakarta.inject-api-
1.0.jar:lib/jakarta.enterprise.cdi-api-2.0.1.jar
```

So to be clear, the directory structure looks like

```
jakarta.batch.official.tck-x.y.z/
    artifacts/
    doc/
    ...
    ... as detailed above ...
    ...
jakarta.batch-api-2.0.0-M2.jar
sigtestdev.jar
```

Again, be sure to choose the correct version of the signature file depending on your the Java version (e.g V8) of your JDK/JRE.

## 7.5. Determining success

The output of your execution should include, at the very end:

```
STATUS:Passed
```

Again, in order to pass the Jakarta Batch TCK you have to make sure that your API passes the signature tests.

## 7.6. Forcing a Signature Test failure (optional)

For additional confirmation that the signature test is working correctly, a failure can be forced by removing the last classpath entry and instead doing:

```
java -jar sigtestdev.jar SignatureTest -static -package jakarta.batch \
-filename artifacts/batch-api-sigtest-java8.sig \
-classpath jakarta.batch-api.jar:$JAVA_HOME/jre/lib/rt.jar:lib/jakarta.inject-api-
1.0.jar
```

You will see a failure like:

```
Warning: Not found annotation type jakarta.enterprise.util.Nonbinding
```

```
Added Annotations
-----------------
```

```
jakarta.batch.api.BatchProperty:          name():anno 0
jakarta.enterprise.util.Nonbinding()
```

```
STATUS:Failed.1 errors
```

## 7.7. Ant script (optional)

We also provide a `sigtest.build.xml` which should typically do a good job encapsulating the `java` execution described above. It uses the `batch-sigtest-tck.properties` file to supply the four classpath entries detailed above.

We list the above approach as the "official" one but this may be helpful as a convenience, and with such a thin wrapper it should be obvious enough whether results should apply.

## 7.8. Gotchas

The differing location of `rt.jar` in different JDK/JRE distributions has been a common cause of non-obvious failures not explained by real divergence in the signatures being tested.

# 8. Executing TestNG Test Suite

The build.xml file is used for running the test suite in standalone mode with ant. The default target, run, will invoke TestNG, running the tests specified in the suite xml file at artifacts/batch-tck-impl-SE-suite.xml (described by the TestNG documentation as "with a testng.xml file"). A report will be generated by TestNG in the results directory.

The list of test cases to run can be customized (for debugging) by modifying the the TestNG suite xml file at artifacts/batch-tck-impl-suite.xml. (Note that an implementation must run against that provided suite.xml file as-is, to pass the TCK.

## 8.1. Steps

1. Edit `batch-tck.properties` to point to your Jakarta Batch API and implementation. Read the comments within this file to understand what values to set.
2. Run via `ant -f build.xml`. Look for results like:

```
[testng] ===============================================
[testng] Jakarta Batch TCK SE
[testng] Total tests run: 152, Failures: 0, Skips: 0
[testng] ===============================================
```

**Note**: there are many forced failure scenarios tested by the TCK, so typically the log will show a lot of exception stack traces during a normal, successful execution even.

## 8.2. Timeouts

The JobOperatorBridge is a utility/helper class in the Jakarta Batch TCK which makes use of the following system property:

```
tck.execution.waiter.timeout
```

using a default value of `900000` (900 seconds).

The intention here is that the test should not wait forever if something catastrophic occurs causing the job to never complete (or if the porting package SPI â€œwaiterâ€ is never notified for some reason). The test also can't end too soon, causing a test failure because the wait was not long enough.

This timeout value can be customized (say, to increase when debugging or decrease to force a faster

failure in some cases).

Note that some of the tests (e.g. the chunk tests involving time-based checkpointing) will take at least 15-25 seconds to run on any hardware, so any value less than that for the whole TCK will cause some test failures simply due to timing (and not because of any failure in the underlying Jakarta Batch implementation).

The 900 seconds value, then, was chosen to avoid falsely reporting an error because of timing out too soon, allowing plenty of leeway. It also facilitates debugging. It does not, however, provide "fast failure" in case of a hang or runaway thread.

### 8.3. Building the TCK (optional, for reference):

The TCK tests can be optionally built from source. However, note that for an implementation to pass the TCK, it must run against the shipped TCK test suite binary as-is (and not against a modified TCK). Still it may be convenient to be able to build the TCK from source for debugging purposes.

## 9. Working with TCK source (debugging, etc.)

For most development/debug use cases it is recommended to refer to the source in the Jakarta Batch TCK] GitHub repository [https://github.com/eclipse-ee4j/batch-tck], and to leverage the Maven automation and artifacts there using the associated documentation.

It should be documented how to use tags/releases, etc. to match the official level tested in the TCK distribution.

It is also possible to use the TestNG `build.xml` script's **compile** target, after setting the **src** property appropriately. We have paid less attention to this more recently and instead focused on the Maven approach.

## 10. Note on TCK guide format

The Jakarta Batch TCK evolved out of the earlier JSR 352 TCK (for more detail see JSR 352: Batch Applications for the Java Platform) and most likely will continue to evolve.

Since there are still some details in the previous JSR 352 TCK reference guide that could possibly be helpful to someone workin with the Jakarta Batch TCK project not yet "ported" to this new guide, we include a link to the old, former JSR 352 reference guide in case it is of use.

## 11. Links

- Jakarta Batch TCK repository - https://github.com/eclipse-ee4j/batch-tck
- Jakarta Batch specification/API repository - https://github.com/eclipse-ee4j/batch-api
- Jakarta Batch project home page - https://projects.eclipse.org/projects/ee4j.jakartabatch

# 12. Change History

## 12.1. Initial Release

- July 17, 2019