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
   2.2. Java SE level - Java 8 or Java 11 . . . 1
3. TCK Challenges/Appeals Process . . . 2
   3.1. Filing a Challenge . . . 2
4. Certification of Compatibility . . . 2
   4.1. Filing a Certification Request . . . 3
5. Installation . . . 3
   5.1. Prerequisites . . . 3
   5.2. Obtaining the Software . . . 3
   5.3. The TCK Environment . . . 3
   5.4. A Quick Tour of the TCK Contents . . . 4
6. Example - How to Run Against 'jbatch' implementation . . . 5
7. TestNG Test Suite Setup . . . 5
   7.1. Required - Configuring the TCK to run against your implementation . . . 5
   7.2. Optional - Property for setting additional JVM Arguments . . . 5
   7.3. The batch-tck-testng.properties file . . . 6
8. Executing the TestNG Test Suite . . . 6
9. TestNG test suite - a deeper look . . . 6
   9.1. The flow of a typical TCK test . . . 7
   9.2. Suite XML File - Easy way of dealing with TCK Challenges . . . 7
   9.3. Porting Package SPI . . . 7
   9.4. Adjusting the Default Timeout Value . . . 7
   9.5. Adjust Test-Specific Wait Times . . . 8
   9.6. Working with TCK source (debugging, etc.) . . . 8
10. Executing Signature Tests . . . 8
   10.1. Prerequisite - Obtaining a Signature Test tool . . . 9
   10.2. Important Jakarta API dependencies . . . 9
   10.3. Running the Signature Tests - Java 8 . . . 9
   10.4. Example Signature Test Command Line - Java 8 . . . 10
   10.5. Determining success . . . 11
   10.6. Forcing a Signature Test failure (optional) . . . 11
   10.7. Ant script (optional) . . . 12
   10.8. Generating the Signature Files (optional) . . . 12

11. Running the Signature Tests - Java 11 12
11.1. jimage Setup - Java 11 13
11.2. Command Line - Java 11 13
12. TCK SPI "Porting Package" in-depth (optional) 13
13. Note on previous (JSR 352) TCK guide 14
14. Links 14
15. Change History 14
15.1. Initial Release - Jakarta Batch 1.0 15
15.2. Update - Jakarta Batch 2.0 15

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 2.0 specification, which can be found at https://jakarta.ee/specifications/batch/2.0/.

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.

2.2. Java SE level - Java 8 or Java 11

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 and Signature tests for a given certification request.

For the current TCK version, this can be done with either Java SE Version 8 or Version 11.

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. Filing a Challenge

The challenge process is defined within 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. Filing a Certification Request

The certification of compatibility process is defined within 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 cerification of compatibility 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. Prerequisites

1. Install the JDK you intend to use for this certification request (Java SE Version 8 or Version 11).
    - Note that the IBM JDK Version 8 cannot be used for reasons explained below. For Java SE Version 8 certification, use Open JDK or Oracle JDK instead.
2. Install Apache Ant.

Note that the Ant installation used below should use the JDK installed in 1. when the 'ant' command is executed during the test execution described below (either via the **JAVA_HOME** variable or other typical methods).

5.2. 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 XML definitions, and signature files) in `/artifacts`, the TCK library dependencies in `/lib` and documentation in `/doc`. You can access the current source code from the Git repository: https://github.com/eclipse-ee4j/batch-tck.

5.3. 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-testng.properties
    batch-tck-sigtest.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 signature test files.

`doc` contains the documentation for the TCK: this reference guide, plus a script that helps provide an example of how to run the TCK against the 'jbatch' implementation.

`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-testng.properties`, `batch-tck-sigtest.properties` Specify properties here for each of the TestNG, signature test portions of the TCK, respectively (And the remaining text files are self-explanatory.)

5.4. A Quick Tour of the TCK Contents

5.4.1. 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.

5.4.2. 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 Job Specification Language (JSL) XML files, which are packaged in the `META-INF/batch-jobs` directory within `artifacts/com.ibm.jbatch.tck-x.y.z.jar`

5.4.3. TestNG suite XML file

The `artifacts/batch-tck-impl-SE-suite.xml` artifact provided in the TCK distribution defines the

TestNG suite, and the list of test classes and test methods.

Note: for debugging purposes, 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. However, also note that an implementation **MUST** run against the provided suite XML file unmodified for an implementation to pass the TCK.

5.4.4. Ant buildfile

The `build.xml` file is used for running the test suite via Apache Ant. The default target, **run**, will invoke **TestNG**, running the tests specified in the suite xml file, and a report will be generated in the results directory.

6. Example - How to Run Against 'jbatch' implementation

The TCK includes documentation in the form of the script: `doc/how-to-run-tck-against-jbatch-script.sh` to show how to execute the entire TCK against the **com.ibm.jbatch** implementation.

The script is written as a template, so you could follow the instructions within comments in the script to edit values unique to your installation, (e.g. **JAVA_HOME** based on the JDK install location), and then run it yourself.

Note: the script is not fully parameterized, so you could not just run it without making some edits first.

7. TestNG Test Suite Setup

A TestNG suite is used to provide the Jakarta Batch runtime execution portion of the TCK. The suite defines the selection of tests to execute, the order of execution, and handles the reporting of test results. Detailed TestNG documentation can be found at testng.org.

This TestNG suite is driven via an Apache Ant buildfile included in the TCK.

7.1. Required - Configuring the TCK to run against your implementation

In order to run the TCK, you must set one required property, **batch.impl.testng.path** to refer to the Jakarta Batch runtime implementation that you are running the TCK against, plus any needed dependencies.

7.2. Optional - Property for setting additional JVM Arguments

An optional property with name **jvm.options** is provided to specify JVM arguments using a `<jvmarg line=""/>` child element of the **TestNG** Ant task. This property value should list the desired JVM

arguments, separated by spaces.

7.3. The batch-tck-testng.properties file

The Ant buildfile will load properties from the file `batch-tck-testng.properties`, which can be a convenient place to define properties like **batch.impl.testng.path** and **jvm.options**.

This `batch-tck-testng.properties` file also contains a set of predefined values for test-specific sleep and wait times, which may be customized for a given implementation/environment and still result in a valid TCK execution, suitable for certification.

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.testng.path=$HOME/foo/lib/classes:$HOME/foo/lib/foo.jar:$HOME/foo/lib/jakar
ta.batch-api.jar
```

8. Executing the TestNG Test Suite

1. Edit `batch-tck-testng.properties` to point to your Jakarta Batch API and implementation, via the **batch.impl.testng.path** property, and any other desired properties.
2. Run via `ant -f build.xml`. Of course, it is valid to specify properties on the command line, e.g. `ant -f build.xml -Dprop=val` in addition to, or instead of within `batch-tck-testng.properties`.
3. 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.

4. If you experienced a failure, it is possible that you experienced a timing issue. The TCK has several built-in properties allowing for tuning of execution to deal with these. There is more information on this later on in the guide.

9. TestNG test suite - a deeper look

9.1. The flow of a typical TCK test

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).

Several tests intentionally produce failures to test relevant portions of the specification, so a normal execution may cause a number of stack traces, error messages, etc. to stdout.

9.2. Suite XML File - Easy way of dealing with TCK Challenges

One reason TestNG was chosen was the ability to use a single XML file to hold excludes from a set of compiled tests. This is an easy way to update a suite after a TCK challenge, by updating the exclude list in this single XML file without having to update and rebuild Java source.

9.3. 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.

9.4. Adjusting the Default Timeout Value

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).

This prevents tests from "hanging" indefinitely if something catastrophic occurs causing the job to never complete (or if the porting package SPI "waiter" is never notified for some reason).

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 default value less than that applied to all tests would cause failures simply due to timing (and not because of any failure in the underlying Jakarta

Batch implementation).

The value of 900 seconds was chosen, then, to avoid falsely reporting an error because of timing out too soon, allowing plenty of time for a test to finish executing, even on slower hardware, and leaves some time to attach a debugger.

It does not, however, provide "fast failure" in case of a hang or runaway thread.

In any case, this timeout value can be customized (say, to increase when debugging or decrease to force a faster failure in some cases).

9.5. Adjust Test-Specific Wait Times

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-testng.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 comments in the test source for a specific test to better understand how the time value is used for that test.

9.6. 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.

Though the TestNG `build.xml` script has a **compile** target, using a **src** property which could be set appropriately, this is an older usage that we haven't focused on keeping updated. More recently we have focused on Maven automation.

Note too 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).

10. Executing Signature Tests

One of the requirements of an implementation passing the TCK is for it to pass the signature test, which tests that implementations have not added their own extensions (classes, methods, etc.) to specification-defined packages. In the case of Jakarta Batch this tests that an implementation conforms to the contents of the **jakarta.batch.*** packages defined by the specification.

This section describes how to run the signature test against your implementation.

10.1. Prerequisite - 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.

10.2. Important Jakarta API dependencies

The Jakarta Dependency Injection and the Jakarta Contexts Dependency Injection specifications provide a couple key dependency API classes, including:

- the **jakarta.enterprise.util.Nonbinding** class, provided by the `jakarta.enterprise.cdi-api-x.y.z` jar, (e.g. Maven coordinates: **jakarta.enterprise:jakarta.enterprise.cdi-api:3.0.0**)
- the **jakarta.inject.*** classes, provided by the `jakarta.inject-api-x.y.z` jar, (e.g. Maven coordinates: **jakarta.inject:jakarta.inject-api:2.0.0**)

It is not obvious that the above annotations are necessary to completely define the "signature" of some Jakarta Batch classes, which is why we call out this detail here.

10.2.1. Dependencies no longer packaged with TCK

In previous versions of the Jakarta Batch TCK the two jars were packaged as a convenience, but this was removed to clarify that it was not a requirement to run with these particular jars.

10.3. Running the Signature Tests - Java 8

The TCK package contains the signature file: `batch.standalone.tck.sig_2.0_se8` in the `artifacts` directory.

Run the signature test by executing a command like the following (from the `artifacts` directory):

```
    $JAVA_HOME/bin/java -jar $SIGTEST_DEV_JAR SignatureTest -static -package
jakarta.batch \
    -filename batch.standalone.tck.sig_2.0_se8 -classpath \
    $JAVA_HOME/jre/lib/rt.jar:$IMPL_PATH
```

Note the variables above, the valuesof which you may need to modify:

- **JAVA_HOME**: the home of your JDK Version 8 installation that you're using for this test

execution.

- **SIGTEST_DEV_JAR**: the location of your signature test tool jar, which you must download separately.
- **IMPL_PATH**: this path should include:
  - the **jakarta.batch.** API classes
  - plus the remainder of your own Jakarta Batch implementation classes
  - the **jakarta.inject.** API classes
  - the **jakarta.enterprise.util.Nonbinding** API class

10.4. Example Signature Test Command Line - Java 8

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.jar`
   - The Jakarta Dependency Injection JAR as `jakarta.inject-api-2.0.0.jar`
   - The Jakarta Contexts Dependency Injection JAR as `jakarta.enterprise.cdi-api-3.0.0.jar`
   - Your Jakarta Batch implementation, represented here by `com.ibm.jbatch.container-2.0.0.jar` and `com.ibm.jbatch.spi-2.0.0.jar`
3. Your JAVA_HOME variable points to an Oracle or OpenJDK Version 8 JDK

So with the above assumptions, the directory structure would look like:

```
jakarta.batch.official.tck-x.y.z/
    artifacts/
        batch.standalone.tck.sig_2.0_se8
        ...
    doc/
    ...
    ... as detailed above ...
    ...
sigtestdev.jar
jakarta.batch-api-2.0.0.jar
jakarta.inject-api-2.0.0.jar
jakarta.enterprise.cdi-api-3.0.0.jar
com.ibm.jbatch.container-2.0.0.jar
com.ibm.jbatch.spi-2.0.0.jar
```

10.4.1. Command Line

The command line would look like this:

```
   IMPL_PATH=../jakarta.batch-api-2.0.0.jar\
    :../jakarta.inject-api-2.0.0.jar\
    :../jakarta.enterprise.cdi-api-3.0.0.jar\
    :../com.ibm.jbatch.container-2.0.0.jar\
    :../com.ibm.jbatch.spi-2.0.0.jar

    $JAVA_HOME/bin/java -jar ../sigtestdev.jar SignatureTest -static -package
jakarta.batch \
    -filename artifacts/batch.standalone.tck.sig_2.0_se8
    -classpath $JAVA_HOME/jre/lib/rt.jar:$IMPL_PATH
```

10.5. 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.

10.6. 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. E.g., continuing the last example, if we remove the jakarta.enterprise.cdi-api JAR and instead do:

```
   IMPL_PATH=../jakarta.batch-api-2.0.0.jar\
    :../jakarta.inject-api-2.0.0.jar\
    :../com.ibm.jbatch.container-2.0.0.jar\
    :../com.ibm.jbatch.spi-2.0.0.jar

    $JAVA_HOME/bin/java -jar ../sigtestdev.jar SignatureTest -static -package
jakarta.batch \
    -filename artifacts/batch.standalone.tck.sig_2.0_se8
    -classpath $JAVA_HOME/jre/lib/rt.jar:$IMPL_PATH
```

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
```

10.7. 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.

It also detects whether it's running against a Java 8 vs. 11 JVM, and can switch signature files accordingly.

We list the "raw" `java -jar ... ` approach as the "official" one but this may be helpful as a convenience, and with such a thin wrapper it should be easy enough to agree whether results should be valid.

10.8. Generating the Signature Files (optional)

For reference, note that the signature tests for the TCK were generated by building a similar classpath as the one used for executing the tests, but using the 'Setup' argument in place of the 'SignatureTest' argument, e.g.:

```
    $JAVA_HOME/bin/java -jar ../sigtestdev.jar Setup -static -package jakarta.batch \
    -filename artifacts/batch.standalone.tck.sig_2.0_se8
    -classpath $JAVA_HOME/jre/lib/rt.jar:$IMPL_PATH
```

11. Running the Signature Tests - Java 11

In addition to the Java 8 signature file, the TCK package contains the signature file: `batch.standalone.tck.sig_2.0_se11` in the `artifacts` directory.

Read through the instructions regarding Java 8, with a couple changes needed for certifying with Java 11.

The obvious changes are to run with a Java 11 JDK and to use signature file: `batch.standalone.tck.sig_2.0_se11` in the execution command line.

In addition, there is a change needed to reflect changes to the JDK due to the addition of Java

modules.

11.1. jimage Setup - Java 11

Assuming **JAVA_HOME** points to the Java 11 JDK you're using to certify, you need to:

1. Extract modules using the 'jimage' command-line tool

```
cd $JAVA11_MODULES
jimage extract $JAVA_HOME/lib/modules
```

1. Replace `$JAVA_HOME/jre/lib/rt.jar` in the Java 8 example command lines with the "java.base" module you just extracted, which would, continuing this example, be: `$JAVA11_MODULES/java.base`.

11.2. Command Line - Java 11

Having done the jimage extract, you can now run the signature test by executing a command like the following (from the `artifacts` directory):

```
    $JAVA_HOME/bin/java -jar $SIGTEST_DEV_JAR SignatureTest -static -package
jakarta.batch \
    -filename batch.standalone.tck.sig_2.0_se11 -classpath \
    $JAVA11_MODULES/java.base:$IMPL_PATH
```

12. TCK SPI "Porting Package" in-depth (optional)

We save this until the end since most commonly it won't be needed.

The two porting package SPI classes in the Jakarta Batch TCK are:

- **com.ibm.jbatch.tck.spi.JobExecutionWaiter**
- **com.ibm.jbatch.tck.spi.JobExecutionWaiterFactory**

The default implementations of these provided by the Jakarta Batch TCK are, respectively:

- **com.ibm.jbatch.tck.polling.TCKPollingExecutionWaiterFactory$TCKPollingExecutionWaiter**
- **com.ibm.jbatch.tck.polling.TCKPollingExecutionWaiterFactory**

The interface definitions are simply:

```
public interface JobExecutionWaiterFactory {public JobExecutionWaiter createWaiter
(long executionId, JobOperator jobOp, long sleepTime);}

public interface JobExecutionWaiter {JobExecution awaitTermination() throws
JobExecutionTimeoutException;}
```

This SPI can be understood with a simple example showing how it used by the TCK (this sample code is extracted from class **com.ibm.jbatch.tck.utils.JobOperatorBridge**)

```
long executionId = jobOp.start(jobName, jobParameters);
JobExecutionWaiter waiter = waiterFactory.createWaiter(executionId, jobOp, sleepTime);
try {
  terminatedJobExecution = waiter.awaitTermination();  }
catch (JobExecutionTimeoutException e) { // ... }
```

So all that's happening here is that we're "waiting" for the asynchronous job execution to complete, using a blocking method that will either return when execution is complete, or throw an exception if we reach the specified 'sleepTime'.And the provided, **com.ibm.jbatch.tck.polling.TCKPollingExecutionWaiterFactory** implementation simply polls repeatedly until the timeout.

Finally, note that the **java.util.ServiceLoader** mechanism is used to reference and load the particular SPI implementation. This implies that you need to update file `META-INF/services/com.ibm.jbatch.tck.spi.JobExecutionWaiterFactory` and update the contents with your factory classname, in order to replace the default implementation.

13. Note on previous (JSR 352) TCK guide

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 former JSR 352 reference guide in case it is of use.

14. 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

15. Change History

15.1. Initial Release - Jakarta Batch 1.0

- July 17, 2019

15.2. Update - Jakarta Batch 2.0

- July 30, 2020