Technology Compatibility Kit Reference Guide for JSR-352: Batch Applications for the Java Platform

Specification Lead: IBM

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 JSR-352: Batch Applications for the Java Platform specification.

The Batch Applications for the Java Platform TCK is built atop TestNG. The Batch Applications for the Java Platform TCK uses TestNG version 6.8 to execute the test suite.

The Batch Applications for the Java Platform TCK is provided under the Apache Public License 2.0 [http://www.apache.org/licenses/LICENSE-2.0].

1. Who Should Use This Guide

This guide is for implementers of the Batch Applications for the Java Platform specification, to assist in running the test suite that verifies the compatibility of their implementation.

2. Before You Read This Guide

Before reading this guide, you should familiarize yourself with the Batch Applications for the Java Platform specification.

Information about the specification, including links to the specification documents, can be found on the JSR-352 JCP page [http://jcp.org/en/jsr/detail?id=352].

Before running the tests of the JSR-352 TCK you should become familiar with TestNG. Documentation for TestNG is located at http://testng.org/doc/documentation-main.html

Introduction

The JSR-352 TCK tests implementations of the Batch Applications for the Java Platform specification, which describes the job specification language, Java programming model, and runtime environment for batch applications for the Java platform.

1.1 TCK Primer

A TCK, or Technology Compatibility Kit, is one of the three required pieces for any JSR (the other two being the specification document and the reference implementation). The TCK is a set of tools and tests to verify that an implementation of the technology conforms to the specification. The tests are the primary component, but the tools serve an equally critical role of providing a framework and/or set of SPIs for executing the tests.

For a particular implementation to be certified, all of the required tests must pass (i.e., the provided test suite must be run unmodified).

A TCK is entirely implementation agnostic. It should validate assertions by consulting the specification's public API.

1.2 Compatibility Testing

Compatibility testing is the process of testing a technology implementation to make sure that it operates consistently with each platform, operating system, and other environment. The goal is to ensure portability.

Compatibility test development for a given feature relies on a complete specification and reference implementation for that feature. Compatibility testing is not primarily concerned with robustness, performance, or ease of use.

1.2.1 Why Compatibility Testing is Important

Java platform compatibility is important to different groups involved with Java technologies for different reasons:

• Compatibility testing is the means by which JCP ensures that the Java platform does not become fragmented as it is ported to different operating systems and hardware environments.

• Compatibility testing benefits developers working in the Java programming language, allowing them to write applications once and then to deploy them across heterogeneous computing environments without porting.

• Compatibility testing allows application users to obtain applications from disparate sources and deploy them with confidence.

• Compatibility testing benefits Java platform implementors by ensuring a level playing field for all Java platform ports.

1.3 About the JSR 352 TCK

The Batch Applications for the Java platform TCK is designed as a portable, configurable and automated test suite for verifying the compatibility of an implementation of JSR-352: Batch Applications for the Java platform

1.3.1 JSR 352 TCK Specifications and Requirements

This section lists the applicable requirements and specifications for the JSR-352 TCK

- **JSR 352 API** - The Java API defined in the JSR-352 specification and provided by the reference implementation

- **Reference implementation** - The designated SE reference implementation for compatibility testing of the JSR-352 specification can be obtained as a java.net zip download here http://java.net/projects/jbatch/downloads

1.3.2 JSR 352 TCK Components

The JSR 352 TCK includes the following components:

- **TestNG -** The JSR-352 TCK requires version 6.8 of TestNG.

- **Dependency injection implementation -** The JSR-352 TCK requires an implementation of JSR 330 (javax.inject.jar).

- **XMLUnit -** The JSR352 TCK requires version 1.1 of XMLUnit.

- **Test suite -** a collection of TestNG tests, the TestNG test suite descriptor and supplemental resources, such as the test artifacts used.

- **TCK documentation –** README file, and this document.

Appeals Process

If a test is determined to be invalid in function or if its basis in the specification is suspect, the test may be challenged by any implementor of the JSR-352 specification. Each test validity issue must be covered by a separate test challenge. Test validity or invalidity will be determined based on its technical correctness, such as:

- The test itself has bugs (i.e., program logic errors)
- Specification item covered by the test is ambiguous
- Test does not match the specification
- Test assumes unreasonable software requirements/configuration
- Test is biased to a particular implementation

Challenges based upon issues unrelated to technical correctness as defined by the specification will normally be rejected, as the specification document is controlled by a separate process.

Tests found to be invalid will either be placed on the Exclude List for that version of the JSR352-TCK or have a corrected or alternate test made available.

2.1. Who can make challenges to the TCK?

Any implementor may submit an appeal to challenge one or more tests in the JSR-352 TCK.

2.2. What challenges to the TCK may be submitted?

Any test case (e.g., @Test method or an artifact class), test case configuration, annotations and other resources may be challenged by an appeal. Assertions made by the specification are controlled by a separate process.

2.3. How these challenges are submitted?

A test challenge must be made by creating a new issue in the issue tracker of the java.net/projects/jbatch project. Summary and description fields should be completed.

2.4. How and by whom challenges are addressed?

The challenges will be addressed in a timely fashion by a JSR-352 lead, as designated by Specification Lead, IBM.

2.4. How are challenges managed?

If the test challenge is approved and one or more tests are invalidated, the lead may correct or replace the invalidated tests, or place the invalidated tests on the Exclude List for that version of the TCK.

Accepted challenges will be acknowledged via comments in the issue tracker. The issue status will be set to resolved when the lead believes the issue is resolved.

The implementor should, within 30 days, either close the issue if he/she agrees, or reopen the issue if it is not considered to be resolved. Resolved issues not addressed for 30 days will be closed.

Installation

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

3.1 Obtaining the Software

You can obtain a release of the Batch Applications for the Java Platform (JSR-352) TCK as a java.net zip download.

The JSR-352 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 also download the current source code from the Git repository here: http://java.net/projects/jbatch/sources

The JSR-352: Batch Applications for the Java Platform reference implementation (RI) can be obtained from the java.net download page [http://java.net/projects/jbatch/downloads]

3.2 The TCK Environment

The software can simply be extracted. It's recommended that you create a folder named `jsr352` to hold all of the related items. Then, extract the TCK distribution into a subfolder named tck. If you have downloaded the JSR-352 SE reference implementation distribution, extract it into a sub folder named runtime. The resulting folder structure is shown here (Note: this layout is assumed through all descriptions in this reference guide)

```
jsr352/
      tck/
      runtime/
```

Configuration

4.1 TCK Properties

In order to run the TCK, you must define a property pointing to the JSR-352 runtime implementation that you are running the TCK against.

You will need to set one required property prior to running the JSR-352 TCK. This property is defined in the `jsr352-tck.properties` as follows:

Property = Required/Example Value	Description
jsr352.impl.runtime=/path/to/runtime	The location where the SE JSR-352 runtime implementation (that you are running the TCK against) was extracted

4.2 The Porting Package

To run the TCK, you must also implement the porting package SPI. An implementation of this SPI for the JSR-352 reference implementation is available in the TCK zip.

The Batch Applications for the Java Platform (JSR-352) TCK relies on an implementation of the porting package to function. There are times when the tests need to access the spec implementation directly to verify results.

The porting package includes a set of SPIs that provide the TCK this level of access, without tying the tests to a particular implementation.

The four SPI classes in the JSR-352 TCK are as follows:

```
com.ibm.jbatch.tck.spi.BatchContainerServiceProvider

com.ibm.jbatch.tck.spi.JobEndCallback

com.ibm.jbatch.tck.spi.JobEndCallbackManager

com.ibm.jbatch.tck.spi.JSLInheritanceMerger
```

A cursory description of of these interfaces follows.

`BatchContainerServiceProvider`
This interface allows the TCK to obtain instances of the `JobEndCallbackManager` and `JSLInheritanceMerger`. The `BatchContainerServiceProvider` implementation needs to be available on the path specified earlier (jsr352.impl.runtime). For example, the RI specifies this as:
`META-INF/services/com.ibm.jbatch.tck.spi.BatchContainerServiceProvider`

and the file contains:
`com.ibm.jbatch.container.tck.bridge.BatchContainerServiceProviderImpl`

`JobEndCallback`
Used by the `JobEndCallbackManager`

`JobEndCallbackManager`
This interface allows the TCK to register and deregister a job end callback. A job end callback is invoked by the batch runtime when a job ends. The majority of TCK tests require notification that a job has ended to allow it to continue and use the job status values to determine success or failure.

`JSLInheritanceMerger`
TODO

4.3 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].

The `artifacts/jsr299-tck-impl-suite.xml` artifact provided in the TCK distribution must be run by TestNG 6.9 (described by the TestNG documentation as "with a testng.xmlfile") unmodified for an implementation to pass the TCK. This file also allows tests to be excluded from a run.

Executing Tests

Running the Signature Test
One of the requirements of an implementation passing the TCK is for it to pass the signature test. This section describes how the signature file is generated and how to run it against your implementation.

5.1 Obtaining the Signature Test Tool
You can obtain the Sigtest tool from the Sigtest home page at http://sigtest.java.net

5.2 Creating the Signature File

The TCK package contains the file signature file `jsr352-api-sigtest.sig (`in the `artifacts` directory) which was created using the following command:

(Example syntax is shown running from the `jsr352/tck` directory. Modify the sigtestdev jar location in the example for your environment)

```
java -jar sigtestdev.jar Setup -static -package javax.batch -filename
jsr352-api-sigtest.sig -classpath
$JAVA_HOME/jre/lib/rt.jar:lib/javax.inject.jar:../runtime/javax.batch
.api.jar
```

In order to pass the JSR352 TCK you have to make sure that your API passes the signature tests.

5.3 Running the Signature Tests

To run the signature test, change the execution command from `Setup` to `SignatureTest` as follows:

```
java -jar sigtestdev.jar SignatureTest -static -package javax.batch
-filename jsr352-api-sigtest.sig -classpath
$JAVA_HOME/jre/lib/rt.jar:lib/javax.inject.jar:../runtime/javax.batch
.api.jar
```

You must chose the right version of the signature file depending on your Java version. In order to run against your own API replace `javax.batch.api.jar` with your own API jar. You should get the message "STATUS:Passed."

5.4 Forcing a Signature Test failure

To confirm that the signature test is working correctly, a failure can be forced by doing the following:

- Edit `jsr352-api-sigtest.sig`
- Modify one of the class signatures. For example, change this:

```
CLSS public abstract interface
javax.batch.api.chunk.CheckpointAlgorithm
meth public abstract boolean isReadyToCheckpoint() throws
java.lang.Exception
meth public abstract int checkpointTimeout() throws
java.lang.Exception
meth public abstract void beginCheckpoint() throws
java.lang.Exception
meth public abstract void endCheckpoint() throws java.lang.Exception
```

to the following:

(changing the `isReadyToCheckpoint` method to accept a `java.lang.String` parameter)

```
CLSS public abstract interface
javax.batch.api.chunk.CheckpointAlgorithm
meth public abstract boolean isReadyToCheckpoint(java.lang.String)
throws java.lang.Exception
meth public abstract int checkpointTimeout() throws
java.lang.Exception
meth public abstract void beginCheckpoint() throws
java.lang.Exception
meth public abstract void endCheckpoint() throws java.lang.Exception
```

When the signature test is then run, it will fail with the following error:

```
Missing Methods
---------------
javax.batch.api.chunk.CheckpointAlgorithm:                    method
public abstract boolean
javax.batch.api.chunk.CheckpointAlgorithm.isReadyToCheckpoint(java.la
ng.String) throws java.lang.Exception

Added Methods
-------------
javax.batch.api.chunk.CheckpointAlgorithm:                    method
public abstract boolean
javax.batch.api.chunk.CheckpointAlgorithm.isReadyToCheckpoint() throw
s java.lang.Exception

duplicate messages suppressed: 1

STATUS:Failed.3 errors
```

Executing the Test Suite

The `build.xml` file is used for running the test suite in standalone mode with ant.
The default target (run) target will invoke TestNG, running the tests specified in the suite xml file at `artifacts/jsr352-tck-impl-suite.xml` (described
by the TestNG documenation as "with a testng.xmlfile"). A report will be generated by TestNG in the `results` directory.

The list of test cases to run can be customized by modifying the the TestNG suite xml file at `artifacts/jsr352-tck-impl-suite.xml`. (RNote that an implementation must run against that provided suite.xml file as-is, to pass the TCK.

Building the TCK

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)

The TCK source is included with the TCK zip, and can be located in `jsr352-tck-impl-src.jar`. Extract this archive to a directory, and note that location. Modify the “tck-src” property to point to the directory to which you've extracted the source. The “compile” target can then be used to build the TCK from source, with the resulting class files being located in the “build” directory.

Spec Coverage

The tests themselves are commented to indicate its test assertion and test strategy.

Timeouts

The JobOperatorBridge makes use of the following system property:

```
junit.jobOperator.sleep.time
```

with 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 job end callback never occurs). The test also can't end too soon, causing a test failure because the wait was not long enough (or to kill the JVM when debugging).

This timeout value can be customized (say, to increase when debugging or decrease to make tests run slightly faster). The default value of 900 seconds is then optimized for debugging, rather than “fast failure”. Note that some of the tests (e.g. the chunk tests) should take at least 15-25 seconds to run, so any value less than that for the whole TCK will cause some false negatives due to timing.