

Technology Compatibility Kit Reference Guide for Jakarta Core Profile

Table of Contents

Preface	1
Who Should Use This Book	2
Before You Read This Book	2
How This Book Is Organized	2
1. Introduction (Core Profile TCK)	3
1.1. TCK Primer	3
1.2. Compatibility Testing	3
1.3. Compatibility Requirements	4
1.4. About the Jakarta Core Profile TCK	8
2. Appeals Process	9
2.1. What challenges to the TCK may be submitted?	9
2.2. How these challenges are submitted?	9
2.3. How and by whom challenges are addressed?	10
2.4. How accepted challenges to the TCK are managed?	10
3. Installation	10
3.1. Obtaining the Software	10
3.2. The TCK Environment	10
4. Core Profile TCK Configuration	11
4.1. TCK Properties	11
4.2. Standalone TCK Configuration	11
5. Reporting	12
5.1. Maven, Failsafe, Surefire and TestNG	12
6. Running the Signature Tests	12
6.1. Running the Component Specification Signature Tests	13
7. Executing (Core Profile TCK)	13
8. WildFly Example	13

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 Core Profile.

The Core Profile TCK is built atop JUnit5 framework and Arquillian platform. The Core Profile TCK uses the Arquillian version *1.7.0.Alpha10* to execute the test suite.

The Core Profile TCK is provided under [Apache Public License 2.0](#).

Who Should Use This Book

This guide is for implementors of the Jakarta Core Profile 10.0 technology to assist in running the test suite that verifies the compatibility of their implementation.

Before You Read This Book

Before reading this guide, you should familiarize yourself with the Jakarta EE programming model, specifically the Jakarta Restful Webservices 3.1 and the Jakarta Contexts and Dependency Injection 4.0 specifications. A good resource for the Jakarta EE programming model is the [Jakarta EE](#) web site.

The Core Profile TCK is based on the following Jakarta technologies:

- Jakarta EE Core Profile [Core Profile 10.0](#)
- Jakarta Annotations [Annotations 2.1](#)
- Jakarta Contexts and Dependency Injection Lite [CDI 4.0](#).
- Jakarta Dependency Injection [DI 2.0](#)
- Jakarta Interceptors [Interceptors 2.1](#)
- Jakarta JSON Binding [JSON-B 3.0](#)
- Jakarta JSON Processing [JSON-B 2.1](#)
- Jakarta RESTful Web Services [Rest 3.1](#)

Before running the tests in the Core Profile TCK, optionally read and become familiar with the Arquillian testing platform. A good starting point could be a series of [Arquillian Guides](#).

How This Book Is Organized

If you are running the Core Profile TCK for the first time, read [Introduction \(Core Profile TCK\)](#) completely for the necessary background information about the TCK. Once you have reviewed that material, perform the steps outlined in the remaining chapters.

- [Introduction \(Core Profile TCK\)](#) gives an overview of the principles that apply generally to all Technology Compatibility Kits (TCKs), outlines the appeals process and describes the Core Profile TCK architecture and components. It also includes a broad overview of how the TCK is executed and lists the platforms on which the TCK has been tested and verified.
- [Appeals Process](#) explains the process to be followed by an implementor, who wish to challenge any test in the TCK.
- [Installation](#) explains where to obtain the required software for the Core Profile TCK and how to

install it. It covers both the primary TCK components as well as tools useful for troubleshooting tests.

- [Core Profile TCK Configuration](#) details the configuration of the JBoss Test Harness, how to create a TCK runner for the TCK test suite and the mechanics of how an in-container test is conducted.
- [Reporting](#) explains the test reports that are generated by the TCK test suite and introduces the TCK audit report as a tool for measuring the completeness of the TCK in testing the CDI specification and in understanding how testcases relate to the specification.
- [Executing \(Core Profile TCK\)](#) documents how the TCK test suite is executed. It covers both modes supported by the TCK, standalone and in-container, and shows how to dump the generated test artifacts to disk.

1. Introduction (Core Profile TCK)

This chapter explains the purpose of a TCK and identifies the foundation elements of the Jakarta Core Profile TCK.

1.1. TCK Primer

A TCK, or Technology Compatibility Kit, is one of the three required pieces for any specification (the other two being the specification document and a compatible 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.

The tests in the TCK are derived from assertions in the written specification document. The assertions are itemized in an XML document, where they each get assigned a unique identifier, and materialize as a suite of automated tests that collectively validate whether an implementation complies with the aforementioned assertions, and in turn the specification. For a particular implementation to be certified, all the required tests must pass (i.e., the provided test suite must be run unmodified).

A TCK is entirely implementation agnostic. Ideally, it should validate assertions by consulting the specification's public API. However, when the information returned by the public API is not low-level enough to validate the assertion, the implementation must be consulted directly. In this case, the TCK provides an independent API as part of a porting package that enables this transparency. The porting package must be implemented for each Core Profile implementation.

1.2. Compatibility Testing

The goal of any specification is to eliminate portability problems so long as the program which uses the implementation also conforms to the rules laid out in the specification.

Executing the TCK is a form of compatibility testing. It's important to understand that compatibility testing is distinctly different from product testing. The TCK is not concerned with robustness, performance or ease of use, and therefore cannot vouch for how well an implementation meets these criteria. What a TCK can do is to ensure the exactness of an implementation as it relates to the

specification.

Compatibility testing of any feature relies on both a complete specification and a complete compatible implementation. The compatible implementation demonstrates how each test can be passed and provides additional context to the implementor during development for the corresponding assertion.

1.2.1. Why Compatibility Is Important

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

- Compatibility testing is the means by which the Jakarta ensures that the Java platform does not become fragmented as it's ported to different operating systems and hardware.
- Compatibility testing benefits developers working in the Java programming language, enabling them to write applications once and deploy them across heterogeneous computing environments without porting.
- Compatibility testing enables application users to obtain applications from disparate sources and deploy them with confidence.
- Conformance testing benefits Java platform implementors by ensuring the same extent of reliability for all Java platform ports.

The CDI specification goes to great lengths to ensure that programs written for Jakarta EE are compatible and the TCK is rigorous about enforcing the rules the specification lays down.

1.3. Compatibility Requirements

The compatibility requirements for Jakarta Contexts and Dependency Injection Version 3.0 consist of meeting the requirements set forth by the rules and associated definitions contained in this section.

1.3.1. Definitions

These definitions are for use only with these compatibility requirements and are not intended for any other purpose.

Table 1. Definitions

Term	Definition
API Definition Product	A Product for which the only Java class files contained in the product are those corresponding to the application programming interfaces defined by the Specifications, and which is intended only as a means for formally specifying the application programming interfaces defined by the Specifications.

Term	Definition
Computational Resource	<p>A piece of hardware or software that may vary in quantity, existence, or version, which may be required to exist in a minimum quantity and/or at a specific or minimum revision level so as to satisfy the requirements of the Test Suite.</p> <p>Examples of computational resources that may vary in quantity are RAM and file descriptors. Examples of computational resources that may vary in existence (that is, may or may not exist) are graphics cards and device drivers. Examples of computational resources that may vary in version are operating systems and device drivers.</p>
Conformance Tests	<p>All tests in the Test Suite for an indicated Technology Under Test, as distributed by the Platform Group, excluding those tests on the Exclude List for the Technology Under Test.</p>
Documented	<p>Made technically accessible and made known to users, typically by means such as marketing materials, product documentation, usage messages, or developer support programs.</p>
Edition	<p>A Version of the Java Platform. Editions include Java Platform Standard Edition and Jakarta Platform Enterprise Edition.</p>
Exclude List	<p>The most current list of tests, distributed by the Platform Group or TCK Lead, that are not required to be passed to certify conformance. The Platform Group or TCK Lead may add to the Exclude List for that Test Suite as needed at any time, in which case the updated Exclude List supplants any previous Exclude Lists for that Test Suite.</p>
Libraries	<p>The class libraries for the Technology Under Test. The Libraries for Jakarta Contexts and Dependency Injection Version 4.0 are listed in libraries.</p>

Term	Definition
Location Resource	A location of classes or native libraries that are components of the test tools or tests, such that these classes or libraries may be required to exist in a certain location in order to satisfy the requirements of the test suite. For example, classes may be required to exist in directories named in a CLASSPATH variable, or native libraries may be required to exist in directories named in a PATH variable.
Product	A licensee product in which the Technology Under Test is implemented or incorporated, and that is subject to compatibility testing.
Product Configuration	A specific setting or instantiation of an Operating Mode. For example, a Product supporting an Operating Mode that permits user selection of an external encryption package may have a Product Configuration that links the Product to that encryption package.
Compatible Implementation (CI)	The prototype or "proof of concept" implementation of a Specification.
Resource	A Computational Resource, a Location Resource, or a Security Resource.
Rules	These definitions and rules in this Compatibility Requirements section of this User's Guide.
Security Resource	A security privilege or policy necessary for the proper execution of the Test Suite. For example, the user executing the Test Suite will need the privilege to access the files and network resources necessary for use of the Product.
Specifications	The documents produced through the Jakarta EE Specification Process that define a particular Version of a Technology. The Specifications for the Technology Under Test are referenced later in this chapter.
TCK Lead	Person responsible for maintaining TCK for the Technology. TCK Lead is representative of Red Hat Inc.
Technology	Specifications and a compatible implementation produced through the Jakarta EE Specification Process.

Term	Definition
Technology Under Test	Specifications and the compatible implementation for Jakarta Contexts and Dependency Injection Version 3.0.
Test Suite	The requirements, tests, and testing tools distributed by the Platform Group or TCK Lead as applicable to a given Version of the Technology.
Version	A release of the Technology, as produced through the Jakarta EE Specification Process.

1.3.2. Rules for Jakarta Core Profile Version 10.0 Products

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

CORE_PROFILE-1 The Product must be able to satisfy all applicable compatibility requirements, including passing all Conformance Tests Rules.

CORE_PROFILE-2 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 specified in [TCK Properties](#). 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.

CORE_PROFILE-3 The testing tools supplied as part of the Test Suite or as updated by the must be used to certify compliance.

CORE_PROFILE-4 The Exclude List associated with the Test Suite cannot be modified.

CORE_PROFILE-6 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.

CORE_PROFILE-7 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.

CORE_PROFILE-8 Except for tests specifically required by this TCK to be recompiled (if any), the binary Conformance Tests supplied as part of the Test Suite must be used to certify compliance.

CORE_PROFILE-9 The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.

CORE_PROFILE-10 In addition to the instructions and requirements the Core Profile TCK, each Product must pass the following standalone TCKs for the following technologies:

- Jakarta Contexts and Dependency Injection, Lite [CDI 4.0](#).
 - This TCK includes compatibility and signature tests for Jakarta Jakarta Interceptors.
 - The Jakarta Contexts and Dependency Injection Language Model TCK included in the CDI distribution.
- Jakarta Dependency Injection [DI 2.0](#)
- Jakarta JSON Binding [JSON-B 3.0](#)
- Jakarta JSON Processing [JSON-B 2.1](#)
- Jakarta RESTful Web Services [Rest 3.1](#)
- Signature tests for Jakarta Common Annotations [Annotations 2.1.1](#)

1.4. About the Jakarta Core Profile TCK

The Jakarta Core Profile TCK is designed as a portable, configurable and automated test suite for verifying the compatibility of an implementation of the Jakarta CDI specification. The test suite is built atop Junit5 framework and Arquillian platform.

Each test class in the suite acts as a deployable unit. The deployable units, or artifacts are packaged as a WAR.

NOTE

The test archives are built with ShrinkWrap, a Java API for creating archives. ShrinkWrap is a part of the Arquillian platform ecosystem.

1.4.1. Jakarta Core Profile TCK Specifications and Requirements

This section lists the applicable requirements and specifications for the Jakarta Core Profile TCK.

- **Specification requirements** - Software requirements for a Jakarta Core Profile implementation include a Java SE 11 or newer compatible runtime.
- **Jakarta Contexts and Dependency Injection API** - The Java API defined in the CDI specification and provided by the compatible implementation.
- **Testing platform** - The Jakarta Core Profile TCK requires version 1.7.0.Alpha10 of the Arquillian (<http://arquillian.org>) and an Arquillian container implementation that can deploy ShrinkWrap [WebArchives](#) of the test contents. The TCK test suite is based on Junit 5.8.2 (<http://junit.org>).
- **Porting Package** - An implementation of SPIs that are required for the test suite to run the in-container tests and at times extend the Jakarta Core Profile 4.0 API to provide extra information to the TCK.
- **Compatible implementation** - A compatible implementation runtime for compatibility testing of the Jakarta Platform Enterprise Edition Core Profile 10.
- **Jakarta Container and Dependency Injection (CDI)** - CDI builds on Jakarta Annotations and Jakarta Interceptors, and the standalone CDI TCK provides signature and unit testing for those

component specifications. Passing the CDI TCK is sufficient for validation of compatibility of Jakarta Annotations and Jakarta Interceptors.

- **Jakarta Dependency Injection (DI)** - CDI builds on DI, and as such CDI implementations must additionally pass the Jakarta Dependency Injection TCK.

1.4.2. Core Profile TCK Components

The Core Profile TCK includes the following components:

- **Arquillian 1.7.0.Alpha10**
- **Junit 5.8.2**
- **TestNG 7.4.x**
- **Porting Package SPIs** - Extensions to the CDI SPIs to allow testing of a container.
- **The test suite**, which is a collection of Junit 5 tests, the TestNG test suite descriptor and supplemental resources that configure CDI and other software components.
- **TCK documentation** accompanied by release notes identifying updates between versions.

The Core Profile TCK has been tested on following platforms:

- WildFly 27 Preview using Eclipse Temurin Java SE 11 and Eclipse Temurin Java SE 17 on Linux based operating systems.

2. Appeals Process

While the Jakarta Core Profile TCK is rigorous about enforcing an implementation's conformance to the Jakarta Core Profile specification, it's reasonable to assume that an implementor may discover new and/or better ways to validate the assertions. The appeals process is defined by the Jakarta EE [Jakarta EE TCK Process 1.2](#)

2.1. What challenges to the TCK may be submitted?

Any test case (e.g., test class, @Test method), test case configuration (e.g., beans.xml), test beans, annotations and other resources may be challenged by an appeal.

What is generally not challengeable are the assertions made by the specification. The specification document is controlled by a separate process and challenges to it should be handled by sending an e-mail to [Platform Dev List](#)

2.2. How these challenges are submitted?

To submit a challenge, a new issue should be created in the [Jakarta Platform Project](#) using the label challenge. Any communication regarding the issue should be pursued in the comments of the filed issue for accurate record.

2.3. How and by whom challenges are addressed?

The challenges will be addressed in a timely fashion by the platform dev team.

2.4. How accepted challenges to the TCK are managed?

The workflow for TCK challenges is outlined in [Jakarta EE TCK Process 1.2](#).

Periodically, an updated TCK will be released, containing tests altered due to challenges - no new tests will be added. Implementations are required to pass the updated TCK. This release stream is named 10.0.x, where x will be incremented.

3. Installation

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

3.1. Obtaining the Software

You can obtain a release of the Jakarta Core Profile TCK project from the [Jakarta EE download site](#). The release stream for Jakarta Profile TCK is named 10.0.x. The Jakarta Core Profile TCK is distributed as a ZIP file, which contains the TCK artifacts (the test suite binary and source, test suite configuration file) in the **artifacts** directory, and documentation in **doc** directory.

You can also download the current source code from [GitHub repository](#).

Executing the TCK requires a Jakarta EE 10 or newer runtime environment (i.e., application server), to which the test artifacts are deployed and the individual tests are invoked. The TCK does not depend on any particular Jakarta EE implementation.

Naturally, to execute Java programs, you must have a Java SE runtime environment. The TCK requires Java SE 11 or newer, which you can obtain from the [Java Software](#) website.

3.2. The TCK Environment

The TCK requires the following software to be installed:

- Java SE 11 or newer
- Maven 3.6 or newer
- A Jakarta EE 10 implementation (e.g., WildFly 27.x)

You should refer to EE 10 implementation instructions for how to install the runtime environment.

Unzipping the Jakarta Core Profile TCK archive will create a **core-profile-tck-x.y.z** root folder which contains the TCK contents. To complete the installation, cd into the **artifacts** directory and install the standalone TCKs and Core Profile TCK artifacts using the maven **pom.xml** file. From that directory run:

`mvn install`

to populate the local maven repository with the necessary dependencies.

4. Core Profile TCK Configuration

This chapter lays out how to configure the TCK Harness by specifying the SPI implementation classes, defining the target container connection information, and various other switches. You then learn how to setup a TCK runner project that executes the TCK test suite, putting these settings into practice.

4.1. TCK Properties

The various TCKs will have properties or configuration variables that need to be set in order to enable running the TCK against a compatible implementation. The `examples` directory in the TCK distribution illustrates sample properties setup using Maven.

NOTE

The JSON-B standalone TCK configuration example includes running the tests with the system property `java.locale.providers` set to `COMPAT`. This addresses a known inconsistency in a test when run under both Java SE 11 and Java SE 17.

NOTE

The RESTful TCK configuration expects the following system properties related to security to be set, even though the security tests are disabled. The value of these is meaningless:

- `user` .
- `password`
- `authuser`
- `authpassword`

In the WildFly runner example these are set to `unused`. These properties are validated by the initialization of the security related test classes before the tests are run. In addition, the Jakarta Core Profile TCK overrides the default RESTful TCK implicit test suite to remove tests that use XML binding but are not tagged with `xml_binding` and security tests that require some Jakarta Security related configuration on the server side.

4.2. Standalone TCK Configuration

Refer to the configuration section of the CDI, RESTful, JSON-P and JSON-B standalone TCK user guides for configuration specific to each TCK. The `examples` directory in the TCK distribution illustrates sample configuration setup using Maven.

4.2.1. Jakarta RESTful TCK Configuration

The Jakarta Core Profile TCK provides a JUnit 5 suite runner that excludes the tests tagged with

`xml_binding`, `servlet` and `security` as these specifications are not part of the Core Profile. The `examples/wf-core-tck-runner/rest-tck` project contains a `pom.xml` that configure surefire to run the `ee.jakarta.tck.coreprofile.rs.CoreProfileRestTCKSuite` class which configures the Core Profile RESTful TCK test suite.

NOTE

Implementations that support any of these technologies in their Core Profile implementation are free to remove these exclusions to enable these tests.

5. Reporting

This chapter covers the execution results.

5.1. Maven, Failsafe, Surefire and TestNG

When the Jakarta Core Profile TCK test suite is executed during the Maven test phase of the TCK runner project, TestNG is invoked indirectly through the Maven Surefire plugin. Surefire is a test execution abstraction layer capable of executing a mix of tests written for JUnit, TestNG, and other supported test frameworks.

Why is this relevant? It means two things. First, it means that you are going to get a summary of the test run on the commandline.

If the Maven reporting plugin that complements Surefire is configured properly, Maven will also generate a generic HTML test result report. That report is written to the file `test-report.html` in the `target/surefire-reports` directory of the TCK runner project. It shows how many tests were run, how many failed and the success rate of the test run.

The one drawback of the Maven Surefire report plugin is that it buffers the test failures and puts them in the HTML report rather than outputting them to the commandline. If you are running the test suite to determine if there are any failures, it may be more useful to get this information in the foreground. You can prevent the failures from being redirected to the report using the following commandline switch:

```
mvn test -Dsfire.useFile=false
```

The unit test reports will be placed into `target/failsafe` or `target/surefire` depending on which TCK is being run.

6. Running the Signature Tests

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

The Core Profile specification has no API artifact other than the utility `api.jar` that is a combination of the various component specifications that make up the Core Profile. As such, there is no Core

Profile signature test.

6.1. Running the Component Specification Signature Tests

Each required component TCK describes how to run its signature tests. The JSON-P, JSON-B and Restful standalone TCKs include a test that sets up and runs the signature tests as part of the Junit 5 tests. Running those standalone TCKs generates the signature test results.

The CDI TCK includes a pom file to execute the signature tests. See the CDI TCK user guide for how to run those tests.

7. Executing (Core Profile TCK)

The Jakarta Core Profile is designed to be executed in a framework like Maven where the Core Profile TCK along with each standalone TCK described as required in the [Introduction \(Core Profile TCK\)](#) are configured in a profile that defined the dependencies and configuration to run the associated TCK.

8. WildFly Example

In the examples directory of the TCK distribution, there is a wf-core-tck-runner maven project that illustrates running the standalone TCKs along with the Jakarta Core Profile TCK. The [wf-core-tck-runner/README-WFP.adoc](#) describes how to run these TCKs against WildFly.

The associated [pom.xml](#) Maven files in each TCK runner directory illustrate the configuration needed to run the tests using Maven and Arquillian.