Technology Compatibility Kit User's Guide for Jakarta EE

Table of Contents

Eclipse Foundation

Technology Compatibility Kit User's Guide for Debugging Support for Other Languages

Release 1.0 for Jakarta EE

September 2019

Preface

The Technology Compatability Kit (TCK) documentation is part of the Java Enterprise Edition contribution to the Eclipse Foundation and is not intended for use in relation to Java Enterprise Edition or Java Licensee requirements. The documentation is in the process of being revised to reflect the new Jakarta EE branding. Additional changes will be made as requirements and procedures evolve for Jakarta EE. Where applicable, references to Java EE or Java Enterprise Edition should be considered references to Jakarta EE.

Please see the Title page for additional license information.

This guide describes how to install, configure, and run the Technology Compatibility Kit (TCK) that is used to test the Debugging Support for Other Languages (Debugging Support for Other Languages 1.0) (JSR 045) technology.

The Debugging Support for Other Languages TCK is a portable, configurable automated test suite for verifying the compatibility of a licensee's implementation of the Debugging Support for Other Languages 1.0 Specification (hereafter referred to as the licensee implementation). The Debugging Support for Other Languages TCK uses the JavaTest harness version T.T.T to run the test suite

Note All references to specific Web URLs are given for the sake of your convenience in locating the resources quickly. These references are always subject to changes that are in many cases beyond the control of the authors of this guide.

Who Should Use This Book

This guide is for developers of the Debugging Support for Other Languages 1.0 technology to assist them in running the test suite that verifies compatibility of their implementation of the Debugging Support for Other Languages 1.0 Specification.

Before You Read This Book

You should be familiar with the Debugging Support for Other Languages 1.0 Specification, which can be found at http://jcp.org/en/jsr/detail?id=045.

Before running the tests in the Debugging Support for Other Languages TCK, you should familiarize yourself with the JavaTest documentation that is included in the Debugging Support for Other Languages TCK documentation bundle.

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

Convention	Meaning	Example

Boldface	Boldface type indicates graphical user interface elements associated with an action, terms defined in text, or what you type, contrasted with onscreen computer output.	From the **File** menu, select **Open Project**.
		
		A **cache** is a copy that is stored locally.
		
		`machine_name% *su*`
		`Password:`

`Monospace`	Monospace type indicates the names of files and directories, commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
Italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.	Read Chapter 6 in the *User's Guide*. Do *not* save the file. The command to remove a file is `rm` *filename*.

Shell Prompts in Command Examples

The following table shows the default UNIX system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

Shell	Prompt
C shell	`machine_name%`
C shell for superuser	`machine_name#`
Bourne shell and Korn shell	`$`
Bourne shell and Korn shell for superuser	`#`
Bash shell	`shell_name-shell_version$`
Bash shell for superuser	`shell_name-shell_version#`

1 Introduction

This chapter provides an overview of the principles that apply generally to all Technology Compatibility Kits (TCKs) and describes the Debugging Support for Other Languages TCK 1.0 (JSR 045). It also includes a high level listing of what is needed to get up and running with the Debugging Support for Other Languages TCK.

This chapter includes the following topics:

1.1 Compatibility Testing

Compatibility testing differs from traditional product testing in a number of ways. The focus of compatibility testing is to test those features and areas of an implementation that are likely to differ across other implementations, such as those features that:

- Rely on hardware or operating system-specific behavior
- Are difficult to port
- Mask or abstract hardware or operating system behavior

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.

1.1.1 Why Compatibility Testing is Important

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

- Compatibility testing 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.
- Conformance testing benefits Java platform implementors by ensuring a level playing field for all

Java platform ports.

1.1.2 TCK Compatibility Rules

Compatibility criteria for all technology implementations are embodied in the TCK Compatibility Rules that apply to a specified technology. Each TCK tests for adherence to these Rules as described in Chapter 2, "Procedure for Certification."

1.1.3 TCK Overview

A TCK is a set of tools and tests used to verify that a licensee's implementation of a Java EE technology conforms to the applicable specification. All tests in the TCK are based on the written specifications for the Java platform. A TCK tests compatibility of a licensee's implementation of the technology to the applicable specification of the technology. Compatibility testing is a means of ensuring correctness, completeness, and consistency across all implementations developed by technology licensees.

The set of tests included with each TCK is called the test suite. Most tests in a TCK's test suite are self-checking, but some tests may require tester interaction. Most tests return either a Pass or Fail status. For a given platform to be certified, all of the required tests must pass. The definition of required tests may change from platform to platform.

The definition of required tests will change over time. Before your final certification test pass, be sure to download the latest Exclude List for the TCK you are using.

1.1.4 Java Community Process (JCP) Program and Compatibility Testing

The Java Community Process (JCP) program is the formalization of the open process that has been used since 1995 to develop and revise Java technology specifications in cooperation with the international Java community. The JCP program specifies that the following three major components must be included as deliverables in a final Java technology release under the direction of the responsible Expert Group:

- Technology Specification
- Reference Implementation
- Technology Compatibility Kit (TCK)

For further information about the JCP program, go to Java Community Process (http://jcp.org/en/home/index).

1.2 About the TCK

The Debugging Support for Other Languages TCK 1.0 is designed as a portable, configurable, automated test suite for verifying the compatibility of a licensee's implementation of the Debugging Support for Other Languages 1.0 Specification.

The Debugging Support for Other Languages does not define APIs, but instead defines a data format and process. As a result, the TCK is different than most, it verifies the data format, and thus indirectly the process. The input to the process is source code in an arbitrary language, and thus the process cannot be directly tested by the TCK.

1.2.1 TCK Specifications and Requirements

This section lists the applicable requirements and specifications.

- Specification Requirements: Software requirements for a Debugging Support for Other Languages implementation are described in detail in the Debugging Support for Other Languages 1.0 Specification. Links to the Debugging Support for Other Languages specification and other product information can be found at http://jcp.org/en/jsr/detail?id=045.
- Debugging Support for Other Languages Version: The Debugging Support for Other Languages TCK 1.0 is based on the Debugging Support for Other Languages Specification, Version 1.0.
- Reference Implementation: See the RI documentation page at http://javaee.github.io/glassfish for more information.

2 Procedure for Certification

This chapter describes the compatibility testing procedure and compatibility requirements for Debugging Support for Other Languages. This chapter contains the following sections:

2.1 Certification Overview

The certification process for Debugging Support for Other Languages 1.0 consists of the following activities:

- Install the appropriate version of the Technology Compatibility Kit (TCK) and execute it in accordance with the instructions in this User's Guide.
- Ensure that you meet the requirements outlined in ???Compatibility Requirements,??? below.
- Certify to the Java Partner organization that you have finished testing and that you meet all of the compatibility requirements.

2.2 Compatibility Requirements

The compatibility requirements for Debugging Support for Other Languages 1.0 consist of meeting the requirements set forth by the rules and associated definitions contained in this section.

2.2.1 Definitions

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

Table 2-1 Definitions??

Term	Definition
Conformance Tests	All tests in the Test Suite for an indicated Technology Under Test, as distributed by the Maintenance Lead.
Documented	Made technically accessible and made known to users, typically by means such as marketing materials, product documentation, usage messages, or developer support programs.
Exclude List	The most current list of tests, distributed by the Maintenance Lead, that are not required to be passed to certify conformance. The Maintenance 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.
Maintenance Lead	The Java Community Process member responsible for maintaining the Specification, reference implementation, and TCK for the Technology. Oracle is the Maintenance Lead for Debugging Support for Other Languages.
Operating Mode	Any Documented option of a Product that can be changed by a user in order to modify the behavior of the Product. For example, an Operating Mode can be binary (enable/disable optimization), an enumeration (select from a list of protocols), or a range (set the maximum number of active threads). Note that an Operating Mode may be selected by a command line switch, an environment variable, a GUI user interface element, a configuration or control file, etc.
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.
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.

Term	Definition
Specifications	The documents produced through the Java Community Process that define a particular Version of a Technology. The Specifications for the Technology Under Test are referenced later in this chapter.
Technology	Specifications and a reference implementation produced through the Java Community Process.
Technology Under Test	Specifications and the reference implementation for Debugging Support for Other Languages.
Test Suite	The requirements, tests, and testing tools distributed by the Maintenance Lead as applicable to a given Version of the Technology.
Version	A release of the Technology, as produced through the Java Community Process.

2.2.2 Rules for Debugging Support for Other Languages Products

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

OL1 The Product must be able to satisfy all applicable compatibility requirements, including passing all Conformance Tests, in every Product Configuration and in every combination of Product Configurations, except only as specifically exempted by these Rules.

For example, if a Product provides distinct Operating Modes to optimize performance, then that Product must satisfy all applicable compatibility requirements for a Product in each Product Configuration, and combination of Product Configurations, of those Operating Modes.

OL1.1 If an Operating Mode controls a Resource necessary for the basic execution of the Test Suite, testing may always use a Product Configuration of that Operating Mode providing that Resource, even if other Product Configurations do not provide that Resource. Notwithstanding such exceptions, each Product must have at least one set of Product Configurations of such Operating Modes that is able to pass all the Conformance Tests.

For example, a Product with an Operating Mode that controls a security policy (i.e., Security Resource) which has one or more Product Configurations that cause Conformance Tests to fail may be tested using a Product Configuration that allows all Conformance Tests to pass.

OL1.2 A Product Configuration of an Operating Mode that causes the Product to report only version, usage, or diagnostic information is exempted from these compatibility rules.

OL2 Some Conformance Tests may have properties that may be changed. Properties that can be

changed are identified in the configuration interview. Apart from changing such properties and other allowed modifications described in this User's Guide (if any), no source or binary code for a Conformance Test may be altered in any way without prior written permission. Any such allowed alterations to the Conformance Tests would be posted to the Java Licensee Engineering web site and apply to all licensees.

OL4 The Exclude List associated with the Test Suite cannot be modified.

OL5 The Maintenance Lead can define exceptions to these Rules. Such exceptions would be made available to and apply to all licensees.

OL6 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.

OL7 The Product???s generated SMAPs must be as defined by the Specifications.

OL8 Except for tests specifically required by this TCK to be rebuilt (if any), the binary Conformance Tests supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

2.3 Test Appeals Process

Oracle has a well established process for managing challenges to its Java technology Test Suites and plans to continue using a similar process in the future. Oracle, as Debugging Support for Other Languages Maintenance Lead, will authorize representatives from the Java Partner Engineering group to be the point of contact for all test challenges. Typically this will be the engineer assigned to a company as part of its Debugging Support for Other Languages TCK support.

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 licensee of the Debugging Support for Other Languages TCK. 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:

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

Challenges based upon issues unrelated to technical correctness as defined by the specification will normally be rejected.

Test challenges must be made in writing to Java Partner Engineering and include all relevant information as described in Example 2-1, "Test Challenge Form". The process used to determine the validity or invalidity of a test (or related group of tests) is described in Section 2.3.1, "TCK Test Appeals Steps."

All tests found to be invalid will either be placed on the Exclude List for that version of the JAX-RS TCK or have an alternate test made available.

- Tests that are placed on the Exclude List will be placed on the Exclude List within one business day after the determination of test validity. The new Exclude List will be made available to all Debugging Support for Other Languages licensees on the Debugging Support for Other Languages website.
- Oracle, as Maintenance Lead has the option of creating alternative tests to address any challenge. Alternative tests (and criteria for their use) will be made available on the Debugging Support for Other Languages TCK website.

Passing an alternative test is deemed equivalent to passing the original test.

2.3.1 TCK Test Appeals Steps

1. Debugging Support for Other Languages TCK licensee writes a test challenge to Java Licensee Engineering contesting the validity of one or a related set of Debugging Support for Other Languages tests.
   A detailed justification for why each test should be invalidated must be included with the challenge as described in Example 2-1, "Test Challenge Form".
2. Java Licensee Engineering evaluates the challenge.
   If the appeal is incomplete or unclear, it is returned to the submitting licensee for correction. If all is in order, Java Licensee Engineering will check with the responsible test developers to review the purpose and validity of the test before writing a response as described in Example 2-2, "Test Challenge Response Form". Java Licensee Engineering will attempt to complete the response within 5 business days. If the challenge is similar to a previously rejected test challenge (i.e., same test and justification), Java Licensee Engineering will send the previous response to the licensee.
3. The challenge and any supporting materials from test developers is sent to the specification engineers for evaluation.
   A decision of test validity or invalidity is normally made within 15 working days of receipt of the challenge. All decisions will be documented with an explanation of why test validity was maintained or rejected.
4. The licensee is informed of the decision and proceeds accordingly.
   If the test challenge is approved and one or more tests are invalidated, Oracle places the tests on

the Exclude List for that version of the Debugging Support for Other Languages TCK (effectively removing the test(s) from the Test Suite). All tests placed on the Exclude List will have a bug report written to document the decision and made available to all licensees through the bug reporting database. If the test is valid but difficult to pass due to hardware or operating system limitations, Oracle may choose to provide an alternate test to use in place of the original test (all alternate tests are made available to the licensee community).

5. If the test challenge is rejected, the licensee may choose to escalate the decision to the Executive Committee (EC), however, it is expected that the licensee would continue to work with Oracle to resolve the issue and only involve the EC as a last resort.

2.3.2 Test Challenge and Response Forms

Example 2-1 shows the test challenge information you must provide to Java Licensee Engineering to initiate a challenge, and Example 2-2 shows the test challenge response format.

Example 2-1 Test Challenge Form

```
Test Challenger Name and Company:
Specification Name(s) and Version(s):
Test Suite Name and Version:
Exclude List Version:
Test Name:
Complaint (argument for why test is invalid):
.jtr file of the failing test:
Console log of the JavaTest harness and device with all debugging flags turned on (if
applicable):
.jti file for the test run:
Startup scripts for the JavaTest harness and agent (if applicable):
```

Example 2-2 Test Challenge Response Form

```
Test Defender Name and Company:
Test Defender Role in Defense (e.g., test developer, Maintenance Lead, etc.):
Specification Name(s) and Version(s):
Test Suite Name and Version:
Test Name:
Defense (argument for why test is valid):
[Multiple challenges and corresponding responses may be listed here.]
Implications of test invalidity (e.g., other affected tests and test framework code,
creation or exposure of ambiguities in spec (due to unspecified requirements),
invalidation of the reference implementation, creation of serious holes in test suite):
Alternatives (e.g., are alternate test(s) appropriate?):
```

2.4 Specifications for Debugging Support for Other Languages

The Debugging Support for Other Languages specification is available on the JSR 045 Web site at http://jcp.org/en/jsr/detail?id=045 or on the Java Community Process (http://jcp.org/en/home/index) site.

2.5 Reference Runtime for Debugging Support for Other Languages 1.0

Designated Reference Runtimes for compatibility testing of Debugging Support for Other Languages 1.0 are the Sun Software JRE release 1.4 for Solaris OE/SPARC, and win32.

3 Installation

This chapter explains how to install the Debugging Support for Other Languages TCK software. This chapter contains the following sections:

3.1 Obtaining the Reference Implementation

The Reference Implementation for Debugging Support for Other Languages 1.0 is Java 2 Platform, Enterprise Edition ("J2EE") version 1.4. Follow the instructions for installing J2EE.

3.2 Installing the Software

Copy the test (`dsol-tck.jar`) and/or configure the class path such that the test is available on the class path. Adding options to the ???java??? command line to accomplish this is acceptable.

Debugging Debugging Support for Other Languages TCK Contents

The top most Debugging Support for Other Languages TCK installation directory, is referred to as TCK_DIRECTORY throughout the Debugging Support for Other Languages TCK documentation. You can name this directory whatever you want.

Once the Debugging Support for Other Languages TCK is installed, several directories will be created under the TCK_DIRECTORY/. The contents of these directories are as follows (on Win32 platforms assume backslashes in directory paths, instead of forward slashes used here).

Table 3-1 Definitions??

File or Directory	Contents
`dsol-tck.jar`	Contains class files for the Debugging Support for Other Languages TCK.
`doc/`	This directory and its subdirectories contain all of the documentation for the Debugging Support for Other Languages TCK.
`doc/dsol-tck/`	Contains the Debugging Support for Other Languages TCK User???s Guide (dsol-tck.pdf).
`src/`	Contains the source files of the TCK.

4 Running the TCK

This chapter describes how to use the Debugging Support for Other Languages TCK. This chapter contains the following sections:

Debugging Support for Other Languages TCK Operating Assumptions

The following is assumed:
* J2SE SDK version 1.4 or later is installed on the system hosting the test. * The Product to be tested (which implements Debugging Support for Other Languages 1.0) is installed on the system hosting the test.

4.1 Generating the SMAPs to be Tested

The input to the test is a set of SMAPs. The testing party must generate these SMAPs and they must be generated according to the following procedures. There are two forms of SMAP: an unresolved SMAP in an SMAP file and a resolved SMAP embedded in the SourceDebugExtension attribute of a class file. If unresolved SMAPs are exposed, this SMAP form must be tested. If SMAPs are embedded into class files, this SMAP form (class files containing a resolved SMAP) must be tested. If both forms are exposed, the tests must be repeated with each form.

The Product must be used to create the set of SMAPs to be tested. Generally, the Product is a translator. In this case, a set of test source programs must first be written — see “Generating the SMAPs from Test Source” on page 16. If the Product has more than one input language or more than one output language, the test must be repeated for each combination of input and output langauge. If the Product has no input language, an SMAP for each type of output must be used.

4.1.1 Generating the SMAPs from Test Source

Let us call the input language of the translator LI. A set of test source programs in LI must be written.

The set of test source programs in LI must exercise all control structures in LI, all subroutine invocation mechanisms in LI and all source inclusion mechanisms in LI.. Any of these which do not exist in LI are, of course, excepted.

For each test source program, the Product must be used to generate the output program and its

corresponding SMAP. These SMAPs will then be submitted to the TCK test.

4.2 Using the Debugging Support for Other Languages TCK to Test a Product

The following test is applied, one SMAP at a time, to each SMAP generated by the procedures above. The test is executed by launching the Java programming language class VerifySMAP (in dsol-tck.jar) with the SMAP as an argument:

```
java VerifySMAP -classpath TCK_DIRECTORY/dsol-tck.jar path_to_the_smap
```

For example, to test an unresolved SMAP file pass it to the test:
`java VerifySMAP my.smap` For example, to test a class file with an embedded SMAP pass it to the test:
`java VerifySMAP my.class`

If a test fails an exception will be thrown. If the test of any SMAP fails, the TCK has failed.

5 Assertions

This chapter includes the following topics:

5.1 Assertions Tested with the Debugging Support for Other Languages 1.0 TCK

1. Syntax must be valid, per the grammar in the specification.
2. A resolved SMAP must specify a *DefaultStratumId*.
3. A specified `DefaultStratumId` must either be "Java" or be the `StratumId` of a `StratumSection`.
4. No `StratumSection` may have a `StratumId` of "Java".
5. A `FileSection` may only occur after a `StratumSection`.
6. There must be exactly one `FileSection` after each `StratumSection`.
7. In a `FileSection`, each `FileId` must be unique within that `FileSection`.
8. In a `FileSection`, the `FileName` must be non empty.
9. In a `FileSection`, the `AbsoluteFileName`, if specified, must be non empty.
10. A `LineSection` may only occur after a `StratumSection`.
11. There must be exactly one `LineSection` after each `StratumSection`.
12. In a `LineSection`, `RepeatCount` must be greater than or equal to one.
13. In a `LineSection`, `OutputLineIncrement` must be greater than or equal to zero.
14. In a `LineSection`, `InputStartLine` must be greater than or equal to one.
15. In a `LineSection`, `OutputStartLine` must be greater than or equal to one.
16. In a `LineSection`, `LineFileId` must be a `FileId` in the `FileSection` after the same `StratumSection`.
17. In a `VendorSection`, the `VENDORID` must be well formed, per the specification.
18. `FutureSection` must not be used until defined in the maintenance phase of the JSR.
19. There must be at least one `StratumSection`.
20. An embedded SMAP must not occur in a resolved SMAP.
21. An `OpenEmbeddedSection` must be followed by at least one SMAP, and terminated with `CloseEmbeddedSection`.
22. `StratumId` of `CloseEmbeddedSection` must match `StratumId` of `OpenEmbeddedSection`.