TCK User’s Guide for Technology Implementors

Table of Contents

Eclipse Foundation . . . 1
Preface . . . 2
- Who Should Use This Book . . . 2
- Before You Read This Book . . . 2
- Typographic Conventions . . . 3
- Shell Prompts in Command Examples . . . 3

1 Introduction . . . 4
- 1.1 Compatibility Testing . . . 4
- 1.2 About the TCK . . . 6
- 1.3 Getting Started With the TCK . . . 8

2 Procedure for Certification . . . 10
- 2.1 Certification Overview . . . 10
- 2.2 Compatibility Requirements . . . 10
- 2.3 Test Appeals Process . . . 15
- 2.4 Specifications for Jakarta Expression Language . . . 17
- 2.5 Libraries for Jakarta Expression Language . . . 17

3 Installation . . . 18
- 3.1 Obtaining a Compatible Implementation . . . 18
- 3.2 Installing the Software . . . 18

4 Setup and Configuration . . . 20
- 4.1 Configuring Your Environment to Run the TCK Against a Compatible Implementation . . . 20
- 4.2 Configuring Your Environment to Repackage and Run the TCK Against the Vendor Implementation . . . 22
- 4.3 Publishing the Test Applications . . . 22

5 Executing Tests . . . 23
- 5.1 Starting the tests . . . 23
- 5.2 Running a Subset of the Tests . . . 24
- 5.3 Running Signature Test . . . 25
- 5.4 Running the TCK Against another CI . . . 25
- 5.5 Running the TCK Against a Vendor’s Implementation . . . 26
- 5.6 Test Reports . . . 26

6 Debugging Test Problems . . . 27
- 6.1 Overview . . . 27
- 6.2 Test Information . . . 27
- 6.3 Configuration Failures . . . 28
- 6.7 Troubleshooting Tips . . . 28

A Frequently Asked Questions 29
A.1 Where do I start to debug a test failure? 29
A.2 How do I restart a crashed test run? 29
A.3 What would cause tests be added to the exclude list? 29

Eclipse Foundation

Technology Compatibility Kit User's Guide for Jakarta Expression Language

Release 6.0 for Jakarta EE

January 2024

Technology Compatibility Kit User's Guide for Jakarta Expression Language, Release 6.0 for Jakarta EE

Copyright © 2017, 2024 Oracle and/or its affiliates. All rights reserved.

This program and the accompanying materials are made available under the terms of the Eclipse Public License v. 2.0, which is available at http://www.eclipse.org/legal/epl-2.0.

SPDX-License-Identifier: EPL-2.0

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

References in this document to EL refer to the Jakarta Expression Language unless otherwise noted.

Preface

This guide describes how to install, configure, and run the Technology Compatibility Kit (TCK) that is used to test the Jakarta Expression Language (Expression Language 6.0) technology.

The Expression Language TCK is a portable, configurable automated test suite for verifying the compatibility of a vendor's implementation of the Expression Language 6.0 Specification (hereafter referred to as the vendor implementation or VI).



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.

Jakarta EE is a community sponsored and community run program. Organizations contribute, along side individual contributors who use, evolve and assist others. Commercial support is not available through the Eclipse Foundation resources. Please refer to the Eclipse EE4J project site (https://projects.eclipse.org/projects/ee4j). There, you will find additional details as well as a list of all the associated sub-projects (Implementations and APIs), that make up Jakarta EE and define these specifications. If you have questions about this Specification you may send inquiries to el-dev@eclipse.org. If you have questions about this TCK, you may send inquiries to jakartaee-tck-dev@eclipse.org.

Who Should Use This Book

This guide is for vendors that implement the Expression Language 6.0 technology to assist them in running the test suite that verifies compatibility of their implementation of the Expression Language 6.0 Specification.

Before You Read This Book

You should be familiar with the Expression Language 6.0, version 6.0 Specification, which can be found at https://jakarta.ee/specifications/expression-language/6.0/.

Before running the tests in the Expression Language TCK, you should familiarize yourself with the JavaTest documentation which can be accessed at the JT Harness web site.

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 Jakarta Expression Language TCK (Expression Language 6.0 TCK). It also includes a high level listing of what is needed to get up and running with the Expression Language TCK.

This chapter includes the following topics:

- Compatibility Testing
- About the TCK
- Getting Started With the TCK

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 compatible implementation (CI) for that feature. Compatibility testing is not primarily concerned with robustness, performance, nor ease of use.

1.1.1 Why Compatibility Testing is Important

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

- Compatibility testing ensures that the Jakarta platform does not become fragmented as it is ported to different operating systems and hardware environments.
- Compatibility testing benefits developers working in the Jakarta 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 Jakarta platform implementors by ensuring a level playing field for all Jakarta 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 vendor's compatible implementation of a Jakarta EE technology conforms to the applicable specification. All tests in the TCK are based on the written specifications for the Jakarta EE platform. A TCK tests compatibility of a vendor's compatible 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 version of this TCK.

1.1.4 Jakarta EE Specification Process (JESP) Program and Compatibility Testing

The Jakarta EE Specification Process (JESP) program is the formalization of the open process that has been used since 2019 to develop and revise Jakarta EE technology specifications in cooperation with the international Jakarta EE community. The JESP program specifies that the following three major components must be included as deliverables in a final Jakarta EE technology release under the direction of the responsible Expert Group:

- Technology Specification
- Compatible Implementation (CI)
- Technology Compatibility Kit (TCK)

For further information about the JESP program, go to Jakarta EE Specification Process community

page https://jakarta.ee/specifications.

1.2 About the TCK

The Expression Language TCK 6.0 is designed as a portable, configurable, automated test suite for verifying the compatibility of a vendor's implementation of the Expression Language 6.0 Specification.

1.2.1 TCK Specifications and Requirements

This section lists the applicable requirements and specifications.

- Specification Requirements: Software requirements for a Expression Language implementation are described in detail in the Expression Language 6.0 Specification. Links to the Expression Language specification and other product information can be found at https://jakarta.ee/specifications/expression-language/6.0/.
- Expression Language Version: The Expression Language 6.0 TCK is based on the Expression Language Specification, Version 6.0.
- Compatible Implementation: One Expression Language 6.0 Compatible Implementation, Eclipse GlassFish 6.0 is available from the Eclipse EE4J project (https://projects.eclipse.org/projects/ee4j). See the CI documentation page at https://projects.eclipse.org/projects/ee4j.glassfish for more information.

See the Expression Language TCK Release Notes for more specific information about Java SE version requirements, supported platforms, restrictions, and so on.

1.2.2 TCK Components

The Expression Language TCK 6.0 includes the following components:

- Expression Language TCK signature tests; check that all public APIs are supported and/or defined as specified in the Expression Language Version 6.0 implementation under test.
- If applicable, an exclude list, which provides a list of tests that your implementation is not required to pass.
- API tests for all of the Expression Language API in all related packages:
  - `jakarta.el`

The Expression Language TCK tests run on the following platforms:

- Debian GNU/Linux 10

1.2.3 TCK Compatibility Test Suite

The test suite is the collection of tests used by the JavaTest harness to test a particular technology implementation. In this case, it is the collection of tests used by the Expression Language TCK 6.0 to test a Expression Language 6.0 implementation. The tests are designed to verify that a vendor's runtime implementation of the technology complies with the appropriate specification. The individual tests correspond to assertions of the specification.

The tests that make up the TCK compatibility test suite are precompiled and indexed within the TCK test directory structure. When a test run is started, the JavaTest harness scans through the set of tests that are located under the directories that have been selected. While scanning, the JavaTest harness selects the appropriate tests according to any matches with the filters you are using and queues them up for execution.

1.2.4 Exclude Lists

Each version of a TCK includes an Exclude List contained in a `TCK-Exclude-List.txt` file. This is a list of test file URLs that identify tests which do not have to be run for the specific version of the TCK being used. Whenever tests are run, the Junit framework automatically excludes these tests from being executed as those are disabled using *@Disabled* tag in JUnit.

A vendor's compatible implementation is not required to pass or run any test on the Exclude List. The Exclude List file, `docs/TCK-Exclude-List.txt`, is documented in the Expression Language TCK. Please note this file is not parsed to exclude any test and is only for documentation purpose.



From time to time, updates to the Exclude List are made available. The exclude list is included in the Jakarta TCK ZIP archive. Each time an update is approved and released, the version number will be incremented. You should always make sure you are using an up-to-date copy of the Exclude List before running the Expression Language TCK to verify your implementation.

A test might be in the Exclude List for reasons such as:

- An error in an underlying implementation API has been discovered which does not allow the test to execute properly.
- An error in the specification that was used as the basis of the test has been discovered.
- An error in the test itself has been discovered.
- The test fails due to a bug in the tools (such as the JavaTest harness, for example).

In addition, all tests are run against the compatible implementations. Any tests that fail when run on a compatible Jakarta platform are put on the Exclude List. Any test that is not specification-based, or for which the specification is vague, may be excluded. Any test that is found to be implementation dependent (based on a particular thread scheduling model, based on a particular file system behavior, and so on) may be excluded.



Vendors are not permitted to alter or modify Exclude Lists. Changes to an Exclude List can only be made by using the procedure described in Section 2.3.1, "TCK Test Appeals Steps."

1.2.5 TCK Configuration

You need to set several variables in your test environment, and run the Expression Language tests, as described in Chapter 4, "Setup and Configuration."



The Jakarta EE Specification Process support multiple compatible implementations. These instructions explain how to get started with the Eclipse GlassFish 6.0 CI. If you are using another compatible implementation, refer to material provided by that implementation for specific instructions and procedures.

1.3 Getting Started With the TCK

This section provides an general overview of what needs to be done to install, set up, test, and use the Expression Language TCK. These steps are explained in more detail in subsequent chapters of this guide.

1. Make sure that the following software has been correctly installed on the system :

- A Jakarta EE 11 Compatible Implementation
- Java SE 21
- A CI for Expression Language 6.0. One example is Eclipse GlassFish 6.0.
- Expression Language TCK version 6.0
- The Expression Language 6.0 Vendor Implementation (VI)
  See the documentation for each of these software applications for installation instructions. See Chapter 3, "Installation," for instructions on installing the Expression Language TCK.

  1. Set up the Expression Language TCK software.
     See Chapter 4, "Setup and Configuration," for details about the following steps.

     1. Set up your shell environment.

2. Set the required System properties.

2. Test the Expression Language 6.0 implementation.
   Test the Expression Language implementation installation by running the test suite. See Chapter 5, "Executing Tests."

2 Procedure for Certification

This chapter describes the compatibility testing procedure and compatibility requirements for Jakarta Expression Language. This chapter contains the following sections:

- Certification Overview
- Compatibility Requirements
- Test Appeals Process
- Specifications for Jakarta Expression Language
- Libraries for Jakarta Expression Language

2.1 Certification Overview

The certification process for Expression Language 6.0 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 Eclipse Foundation that you have finished testing and that you meet all of the compatibility requirements, as required by the Eclipse Foundation TCK License.

2.2 Compatibility Requirements

The compatibility requirements for Expression Language 6.0 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
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.
Computational Resource	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. 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.
Configuration Descriptor	Any file whose format is well defined by a specification and which contains configuration information for a set of Java classes, archive, or other feature defined in the specification.
Conformance Tests	All tests in the Test Suite for an indicated Technology Under Test, as released and distributed by the Eclipse Foundation, excluding those tests on the published Exclude List for the Technology Under Test.
Container	An implementation of the associated Libraries, as specified in the Specifications, and a version of a Java Platform, Standard Edition Runtime Product, as specified in the Specifications, or a later version of a Java Platform, Standard Edition Runtime Product that also meets these compatibility requirements.
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, released and distributed by the Eclipse Foundation, that are not required to be passed to certify conformance. The Jakarta EE Specification Committee may add to the Exclude List for that Test Suite as needed at any time, in which case the updated TCK version supplants any previous Exclude Lists for that Test Suite.

Term	Definition
Libraries	The class libraries, as specified through the Jakarta EE Specification Process (JESP), for the Technology Under Test. The Libraries for Jakarta Expression Language are listed at the end of this chapter.
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.
Maintenance Lead	The corresponding Jakarta EE Specification Project is responsible for maintaining the Specification, and the TCK for the Technology. The Specification Project Team will propose revisions and updates to the Jakarta EE Specification Committee which will approve and release new versions of the specification and TCK.
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 vendor’s 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.
Rebuildable Tests	Tests that must be built using an implementation-specific mechanism. This mechanism must produce specification-defined artifacts. Rebuilding and running these tests against a known compatible implementation verifies that the mechanism generates compatible artifacts.

Term	Definition
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.
Runtime	The Containers specified in the Specifications.
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 (JESP) that define a particular Version of a Technology. The Specifications for the Technology Under Test are referenced later in this chapter.
Technology	Specifications and one or more compatible implementations produced through the Jakarta EE Specification Process (JESP).
Technology Under Test	Specifications and a compatible implementation for Jakarta Expression Language Version 6.0.
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 Jakarta EE Specification Process (JESP).

2.2.2 Rules for Jakarta Expression Language Products

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

ExpressionLanguage1 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.

ExpressionLanguage1.1 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.

ExpressionLanguage1.2 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.

ExpressionLanguage1.3 An API Definition Product is exempt from all functional testing requirements defined here, except the signature tests.

ExpressionLanguage2 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 identified in the JavaTest Environment (.jte) files in the Test Suite installation. 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 will be provided via the Jakarta EE Specification Project website and apply to all vendor compatible implementations.

ExpressionLanguage3 The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

ExpressionLanguage4 The Exclude List associated with the Test Suite cannot be modified.

ExpressionLanguage5 The Maintenance Lead can define exceptions to these Rules. Such exceptions would be made available as above, and will apply to all vendor implementations.

ExpressionLanguage6 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.

ExpressionLanguage7 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.

ExpressionLanguage7.1 If a Product includes Technologies in addition to the Technology Under Test, then it must contain the full set of combined public and protected classes and interfaces. The API of the

Product must contain the union of the included Technologies. No further modifications to the APIs of the included Technologies are allowed.

ExpressionLanguage8 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.

ExpressionLanguage9 The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.

2.3 Test Appeals Process

Jakarta has a well established process for managing challenges to its TCKs. Any implementor may submit a challenge to one or more tests in the Expression Language TCK as it relates to their implementation. Implementor means the entity as a whole in charge of producing the final certified release. **Challenges filed should represent the consensus of that entity**.

2.3.1 Valid Challenges

Any test case (e.g., test class, @Test method), test case configuration (e.g., deployment descriptor), test beans, annotations, and other resources considered part of the TCK may be challenged.

The following scenarios are considered in scope for test challenges:

- Claims that a test assertion conflicts with the specification.
- Claims that a test asserts requirements over and above that of the specification.
- Claims that an assertion of the specification is not sufficiently implementable.
- Claims that a test is not portable or depends on a particular implementation.

2.3.2 Invalid Challenges

The following scenarios are considered out of scope for test challenges and will be immediately closed if filed:

- Challenging an implementation's claim of passing a test. Certification is an honor system and these issues must be raised directly with the implementation.
- Challenging the usefulness of a specification requirement. The challenge process cannot be used to bypass the specification process and raise in question the need or relevance of a specification requirement.
- Claims the TCK is inadequate or missing assertions required by the specification. See the Improvement section, which is outside the scope of test challenges.
- Challenges that do not represent a consensus of the implementing community will be closed until

such time that the community does agree or agreement cannot be made. The test challenge process is not the place for implementations to initiate their own internal discussions.

- Challenges to tests that are already excluded for any reason.
- Challenges that an excluded test should not have been excluded and should be re-added should be opened as a new enhancement request

Test challenges must be made in writing via the Expression Language specification project issue tracker as described in Section 2.3.3, "TCK Test Appeals Steps."

All tests found to be invalid will be placed on the Exclude List for that version of the Expression Language TCK.

2.3.3 TCK Test Appeals Steps

1. Challenges should be filed via the Jakarta Expression Language specification project's issue tracker using the label `challenge` and include the following information:
    - The relevant specification version and section number(s)
    - The coordinates of the challenged test(s)
    - The exact TCK and exclude list versions
    - The implementation being tested, including name and company
    - The full test name
    - A full description of why the test is invalid and what the correct behavior is believed to be
    - Any supporting material; debug logs, test output, test logs, run scripts, etc.
2. Specification project evaluates the challenge.
   Challenges can be resolved by a specification project lead, or a project challenge triage team, after a consensus of the specification project committers is reached or attempts to gain consensus fails. Specification projects may exercise lazy consensus, voting or any practice that follows the principles of Eclipse Foundation Development Process. The expected timeframe for a response is two weeks or less. If consensus cannot be reached by the specification project for a prolonged period of time, the default recommendation is to exclude the tests and address the dispute in a future revision of the specification.
3. Accepted Challenges.
   A consensus that a test produces invalid results will result in the exclusion of that test from certification requirements, and an immediate update and release of an official distribution of the TCK including the new exclude list. The associated `challenge` issue must be closed with an `accepted` label to indicate it has been resolved.
4. Rejected Challenges and Remedy.
   When a `challenge` issue is rejected, it must be closed with a label of `invalid` to indicate it has been rejected. There appeal process for challenges rejected on technical terms is outlined in Escalation

Appeal. If, however, an implementer feels the TCK challenge process was not followed, an appeal issue should be filed with specification project's TCK issue tracker using the label `challenge-appeal`. A project lead should escalate the issue with the Jakarta EE Specification Committee via email (jakarta.ee-spec@eclipse.org). The committee will evaluate the matter purely in terms of due process. If the appeal is accepted, the original TCK challenge issue will be reopened and a label of `appealed-challenge` added, along with a discussion of the appeal decision, and the `challenge-appeal` issue with be closed. If the appeal is rejected, the `challenge-appeal` issue should closed with a label of `invalid`.

5. Escalation Appeal.
   If there is a concern that a TCK process issue has not been resolved satisfactorily, the Eclipse Development Process Grievance Handling procedure should be followed to escalate the resolution. Note that this is not a mechanism to attempt to handle implementation specific issues.

2.4 Specifications for Jakarta Expression Language

The Jakarta Expression Language specification is available from the specification project web-site: https://jakarta.ee/specifications/expression-language/6.0/.

2.5 Libraries for Jakarta Expression Language

The following is a list of the packages comprising the required class libraries for Expression Language 6.0:

- `jakarta.el`

For the latest list of packages, also see:

https://jakarta.ee/specifications/expression-language/6.0/

3 Installation

This chapter explains how to install the Jakarta Expression Language TCK software.

After installing the software according to the instructions in this chapter, proceed to Chapter 4, "Setup and Configuration," for instructions on configuring your test environment.



Although the Expression Language 6.0 TCK is not depended on any particular build tool, it is convenient to install Apache Maven 3.6.3+ for setup and execution of tests. Any other build tools like Gradle and JUnit 5 Console Runner can also be used that is Jupiter API compatible.

3.1 Obtaining a Compatible Implementation

Each compatible implementation (CI) will provide instructions for obtaining their implementation. Eclipse GlassFish 6.0 is a compatible implementation which may be obtained from https://projects.eclipse.org/projects/ee4j.glassfish

3.2 Installing the Software

Before you can run the Expression Language TCK tests, you must install and set up the following software components:

- A Jakarta EE 11 Compatible Implementation
- Java SE 21
- A CI for Expression Language 6.0, one example is Eclipse GlassFish 6.0
- Expression Language TCK version 6.0
- The Expression Language 6.0 Vendor Implementation (VI)
- Any Jupiter API compatible test runner (eg. Apache Maven 3.6.3+)

Follow these steps:

1. Install the Java SE 21 software, if it is not already installed.
   Download and install the Java SE 21 software from http://www.oracle.com/technetwork/java/javase/downloads/index.html. Refer to the installation instructions that accompany the software for additional information.
2. Install the build tool that will be used to run the TCK, if it is not already installed.
   It will be convenient to use Apache Maven 3.6.3+ for running the tests. The test kit is not depended

on Maven, any build tool compatible with Jupiter API is suffitient.

3. Install the Expression Language TCK 6.0 software.
    1. Copy or download the Expression Language TCK software to your local system.
    You can obtain the Expression Language TCK software from the Jakarta EE site https://jakarta.ee/specifications/expression-language/6.0/.
    2. Use the `unzip` command to extract the bundle in the directory of your choice:
    `unzip jakarta-expression-language-tck-6.0.0.zip`
4. Install the Jakarta EE 11 CI software (the servlet Web container used for running the Expression Language TCK with the Expression Language 6.0 CI), if it is not already installed.
Download and install the Servlet Web container with the Expression Language 6.0 CI used for running the Expression Language TCK 6.0, represented by the Jakarta EE 11 CI. You may obtain a copy of this CI by downloading it from https://projects.eclipse.org/projects/ee4j.glassfish.
5. Install a Expression Language 6.0 Compatible Implementation.
A Compatible Implementation is used to validate your initial configuration and setup of the Expression Language TCK 6.0 tests, which are explained further in Chapter 4, "Setup and Configuration."
The Compatible Implementations for Expression Language are listed on the Jakarta EE Specifications web site: https://jakarta.ee/specifications/expression-language/6.0/.
6. Install the Expression Language VI to be tested.
Follow the installation instructions for the particular VI under test.

4 Setup and Configuration



The Jakarta EE Specification process provides for any number of compatible implementations. As additional implementations become available, refer to project or product documentation from those vendors for specific TCK setup and operational guidance.

This chapter describes how to set up the Expression Language TCK. Before proceeding with the instructions in this chapter, be sure to install all required software, as described in Chapter 3, "Installation."

After completing the instructions in this chapter, proceed to Chapter 5, "Executing Tests," for instructions on running the Expression Language TCK.



The Expression Language TCK is not depended on any particular build tool to run the tests. It will be convenient and advisable to create a Apache Maven project to setup and run the TCK. This chapter will henceforth use instructions and steps to provide setup with Apache Maven as a build tool.

4.1 Configuring Your Environment to Run the TCK Against a Compatible Implementation

After configuring your environment as described in this section, continue with the instructions in Chapter 5, "Executing Tests."



In these instructions, variables in angle brackets need to be expanded for each platform. For example, `<JAVA_HOME>` becomes `$JAVA_HOME` on Solaris/Linux and `%JAVA_HOME%` on Windows. In addition, the forward slashes (`/`) used in all of the examples need to be replaced with backslashes (`\`) for Windows. Finally, be sure to use the appropriate separator for your operating system when specifying multiple path entries (`;` on Windows, `:` on UNIX/Linux).

On Windows, you must escape any backslashes with an extra backslash in path separators used in any of the following properties, or use forward slashes as a path separator instead.

1. Set the following environment variables in your shell environment:
    1. `JAVA_HOME` to the directory in which Java SE 21 is installed
    2. `M2_HOME` to the directory in which the Apache Maven build tool is installed.
    3. `PATH` to include the following directories: `JAVA_HOME/bin`, and `M2_HOME/bin`

2. Set the following System properties:
    1. Set the `webServerHost` property to the name of the host on which Jakarta EE 11 CI is running.
    The default setting is `localhost`.
    2. Set the `webServerPort` property to the port number of the host on which Jakarta EE 11 CI is running.
    The default setting is `8080`.
    3. Set the `el.classes` property to the directory of Jakarta EE 11 CI with implementation jar and api jar.
    4. Set the `porting.ts.url.class.1` property to your porting implementation class that is used for obtaining URLs.
    The default setting for this property is `com.sun.ts.lib.implementation.sun.common.SunRIURL`.
3. Set the below jars to the classpath
    1. JAR file for the Expression Language 6.0 API.
    eg. `${webServerHost}/modules/jakarta.el-api.jar`.
    2. JUnit 5 jars (5.7.2+) Maven cordinates :

```
<dependency>
   <groupId>org.junit</groupId>
   <artifactId>junit-bom</artifactId>
   <version>5.7.2</version>
   <type>pom</type>
   <scope>import</scope>
</dependency>
```

    3. sigtest-maven-plugin (1.4) to run the signature tests. Maven cordinates :

```
<dependency>
      <groupId>jakarta.tck</groupId>
      <artifactId>sigtest-maven-plugin</artifactId>
      <version>2.2</version>
 </dependency>
```

    4. Eclipse GlassFish 6.0 CI jars
    For eg, if you are using the Eclipse GlassFish 6.0 CI below jars need to be added to Classpath

```
${webServerHost}/modules/expressly.jar:
```

4. Provide compatible implementation of the porting package interface provided with the Expression Language TCK.
    The porting package interface, `TSURLInterface.java`, obtains URL strings for web resources in an

implementation-specific manner. API documentation for the `TSURLInterface.java` porting package interface is available in the Expression Language TCK documentation bundle.

5. If the Expression Language TCK test applications are published on a Servlet 5.0-compliant Web container to run the CI, the `servlet_adaptor` property needs to be set as System property, and CI-specific WAR files containing the Servlet information need to be created for publishing.
   The CI-specific WAR files should never override any existing files that come with the TCK. Refer to Appendix B, "Packaging the Test Applications in Servlet-Compliant WAR Files With VI-Specific Information," for more information.

4.2 Configuring Your Environment to Repackage and Run the TCK Against the Vendor Implementation

Not needed for the EL TCK.

4.3 Publishing the Test Applications

Not needed for the EL TCK.

5 Executing Tests

The Expression Language TCK uses the Junit framework to execute the tests.

This chapter includes the following topics:

- Starting the tests
- Running a Subset of the Tests
- Running the TCK Against your selected CI
- Running the TCK Against a Vendor's Implementation
- Test Reports



The instructions in this chapter assume that you have installed and configured your test environment as described in Chapter 3, "Installation," and Chapter 4, "Setup and Configuration,", respectively.



The Jakarta Expression Language TCK is not depended on any particular build tool to run the tests. It will be convenient and advisable to create a Apache Maven project to setup and run the TCK. This chapter will henceforth use instructions and steps to provide setup with Apache Maven as a build tool.

5.1 Starting the tests

The Expression Language TCK can be run from the command line in your shell environment by executing the TCK jar.



The `mvn` command referenced in the following two procedures and elsewhere in this guide is the Apache Maven build tool, which will need to be downloaded separately.

5.1.1 To Start Tests in Command-Line Mode

Start the Junit tests using the following command:

```
mvn verify
```

Example 5-1 Expression Language TCK Signature Tests

To run the Expression Language TCK signature tests, use the sigtest-maven-plugin in the TCK runner as

below.

```
<plugin> <groupId>jakarta.tck</groupId> <artifactId>sigtest-maven-plugin</artifactId> <version>2.2</version> <configuration> <sigfile>path-to-sig-file-provided-with-TCK</sigfile> <packages>jakarta.el</packages> <classes>path-to-api-jar</classes> <report>target/sig-report.txt</report> </configuration> <executions> <execution> <id>sigtest</id> <goals> <goal>check</goal> </goals> <phase>verify</phase> </execution> </executions> </plugin>
```

Example 5-2 Single Test Directory

To run a single test directory, enter the following commands:

```
mvn verify -Dit.test=com/sun/ts/tests/el/api/client.**
```

Example 5-3 Subset of Test Directories

To run a subset of test directories, enter the following commands:

```
mvn verify -Dit.test=com/sun/ts/tests/el/api.**
```

5.2 Running a Subset of the Tests

Use the following modes to run a subset of the tests:

- Section 5.2.1, "To Run a Subset of Tests in Command-Line Mode"

5.2.1 To Run a Subset of Tests in Command-Line Mode

Start the test run by executing the following command:

```
mvn verify -Dit.test=com/sun/ts/tests/el/api.**
```

The tests in the directory and its subdirectories are run.



Currently there are no Junit test tags(groups) available in the TCK.

5.2.2 To Run a Group of Tests in Command-Line Mode

Start the test run by executing the following command:

```
mvn verify -Dgroups={groupsExample}
```

Only the tests in the group `{groupsExample}` is run. Multiple groups can be separated by comma.

5.2.3 To Exclude a Group of Tests in Command-Line Mode

Start the test run by executing the following command:

```
mvn verify -DexcludedGroups={groupsExample}
```

The tests in the group `{groupsExample}` is exclued from the run. Multiple groups can be separated by comma.

5.3 Running Signature Test

sigtest-maven-plugin is used to run the EL signature test. Please refer the sample provided in the platform TCK repository.



Use below configuration while using sigtest-maven-plugin with goal as `check`.

```
<configuration>
    <sigfile>path-to-sig-file-provided-with-TCK</sigfile>
    <packages>jakarta.el</packages>
    <classes><path-to-api-jar</classes>
    <report>sig-report.txt</report>
</configuration>
```

5.4 Running the TCK Against another CI

Some test scenarios are designed to ensure that the configuration and deployment of all the prebuilt Expression Language TCK tests against one Compatible Implementation are successful operating with other compatible implementations, and that the TCK is ready for compatibility testing against the

Vendor and Compatible Implementations.

1. Verify that you have followed the configuration instructions in Section 4.1, "Configuring Your Environment to Run the TCK Against the Compatible Implementation."
2. If required, verify that you have completed the steps in Section 4.3.2, "Deploying the Prebuilt Archives."
3. Run the tests, as described in Section 5.1, "Starting the tests," and, if desired, Section 5.2, "Running a Subset of the Tests."

5.5 Running the TCK Against a Vendor's Implementation

This test scenario is one of the compatibility test phases that all Vendors must pass.

1. Verify that you have followed the configuration instructions in Section 4.2, "Configuring Your Environment to Repackage and Run the TCK Against the Vendor Implementation."
2. If required, verify that you have completed the steps in Section 4.3.3, "Deploying the Test Applications Against the Vendor Implementation."
3. Run the tests, as described in Section 5.1, "Starting the tests," and, if desired, Section 5.2, "Running a Subset of the Tests."

5.6 Test Reports

A set of report files is created for every test run. These report files can be found in the target directory that the test is run. After a test run is completed, the Junit framework writes reports for the test run.

6 Debugging Test Problems

There are a number of reasons that tests can fail to execute properly. This chapter provides some approaches for dealing with these failures. Please note that most of these suggestions are only relevant when running the test harness in GUI mode.

This chapter includes the following topics:

- Overview
- Test Tree
- Folder Information
- Test Information
- Report Files
- Configuration Failures

6.1 Overview

The goal of a test run is for all tests in the test suite that are not filtered out to have passing results. If the root test suite folder contains tests with errors or failing results, you must troubleshoot and correct the cause to satisfactorily complete the test run.

- Errors: Tests with errors could not be executed by the Junit framework. These errors usually occur because the test environment is not properly configured.
- Failures: Tests that fail were executed but had failing results.

For every test run, the Junit framework creates a set of report files in the target directory.

If a large number of tests failed, you should read Configuration Failures to see if a configuration issue is the cause of the failures.



You can set `junit.log.traceflag=true` as System property to get more debugging information.

6.2 Test Information

If you need more information to identify the cause of the error or failure, use the Junit reports generated after running the tests.

6.3 Configuration Failures

Configuration failures are easily recognized because many tests fail the same way. When all your tests begin to fail, you may want to stop the run immediately and start viewing individual test output.

6.7 Troubleshooting Tips

None for the Expression Language TCK.

A Frequently Asked Questions

This appendix contains the following questions.

- Where do I start to debug a test failure?
- How do I restart a crashed test run?
- What would cause tests be added to the exclude list?

A.1 Where do I start to debug a test failure?

From the JavaTest GUI, you can view recently run tests using the Test Results Summary, by selecting the red Failed tab or the blue Error tab. See Chapter 6, "Debugging Test Problems," for more information.

A.2 How do I restart a crashed test run?

If you need to restart a test run, you can figure out which test crashed the test suite by looking at the `harness.trace` file. The `harness.trace` file is in the report directory that you supplied to the JavaTest GUI or parameter file. Examine this trace file, then change the JavaTest GUI initial files to that location or to a directory location below that file, and restart. This will overwrite only `.jtr` files that you rerun. As long as you do not change the value of the GUI work directory, you can continue testing and then later compile a complete report to include results from all such partial runs.

A.3 What would cause tests be added to the exclude list?

The exclude file (`docs/TCK-Exclude-List.txt`) contains all tests that are not required to be run. The file is used only for documentation purpose. The tests are excluded using @Disabled tag in Junit when necessary. The following is a list of reasons for a test to be included in the Exclude List:

- An error in a Compatible Implementation that does not allow the test to execute properly has been discovered.
- An error in the specification that was used as the basis of the test has been discovered.
- An error in the test has been discovered.

Appendix B is not used for the Expression Language TCK.