

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	2
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	11
2 Procedure for Certification	13
2.1 Certification Overview	13
2.2 Compatibility Requirements	13
2.3 Test Appeals Process	17
2.4 Specifications for Jakarta Authentication	19
2.5 Libraries for Jakarta Authentication	20
3 Installation	21
3.1 Obtaining a Compatible Implementation	21
3.2 Installing the Software	21
4 Setup and Configuration	23
4.1 Download the Jakarta Authentication TCK	23
4.2 Setting up an Environment for the Maven Surefire based TCK Against a Compatible Implementation	23
5 Executing Tests	25
5.1 Starting the tests	25
5.2 Running a Subset of the Tests	26
5.3 Running the TCK Against GlassFish	26
6 Debugging Test Problems	27
6.1 Overview	27
6.2 Configuration Failures	27
A Frequently Asked Questions	28
A.1 Where do I start to debug a test failure?	28
A.2 What would cause tests be added to the exclude list?	28

Eclipse Foundation

Technology Compatibility Kit User's Guide for Jakarta Authentication

Release 3.1 for Jakarta EE

June 2024

Technology Compatibility Kit User's Guide for Jakarta Authentication, Release 3.1 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 JASPIC refer to the Jakarta Authentication, unless otherwise noted.

References in this document to JWS refer to Jakarta Web Services Metadata, unless otherwise noted.

Reference in this document to SAAJ refer to Jakarta SOAP Attachments, unless otherwise noted.

References in this document to JAXWS refer to Jakarta XML Web Services, unless otherwise noted.

References in this document to JACC refer to Jakarta Authorization, 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 Authentication (Authentication 3.1) technology.

The Authentication TCK is a portable, configurable automated test suite for verifying the compatibility of a vendor's implementation of the Authentication 3.1 Specification (hereafter referred to as the vendor implementation or VI). The Authentication TCK uses the JUnit 4.13.2 and Maven surefire harness version 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.

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 jaspic-dev@eclipse.org. If you have questions about this TCK, you may send inquiries to jaspic-dev@eclipse.org.

Who Should Use This Book

This guide is for vendors that implement the Authentication 3.1 technology to assist them in running the test suite that verifies compatibility of their implementation of the Authentication 3.1 Specification.

Before You Read This Book

You should be familiar with the Authentication 3.1, version 3.1 Specification, which can be found at <https://jakarta.ee/specifications/authentication/3.1/>.

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. <code>machine_name% *su*</code> <code>Password:</code>
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 <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name% you have mail.</code>
<i>Italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.	Read Chapter 6 in the <i>User's Guide</i> . Do <i>not</i> save the file. The command to remove a file is <code>rm filename</code> .

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	<code>machine_name%</code>
C shell for superuser	<code>machine_name#</code>
Bourne shell and Korn shell	<code>\$</code>
Bourne shell and Korn shell for superuser	<code>#</code>
Bash shell	<code>shell_name-shell_version\$</code>
Bash shell for superuser	<code>shell_name-shell_version#</code>

1 Introduction

This chapter provides an overview of the principles that apply generally to all Technology Compatibility Kits (TCKs) and describes the Jakarta Authentication TCK (Authentication 3.1 TCK). It also includes a high level listing of what is needed to get up and running with the Authentication 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 Authentication TCK 3.1 is designed as a portable, configurable, automated test suite for verifying the compatibility of a vendor's implementation of the Authentication 3.1 Specification.

1.2.1 TCK Specifications and Requirements

This section lists the applicable requirements and specifications.

- **Specification Requirements:** Software requirements for a Authentication implementation are described in detail in the Authentication 3.1 Specification. Links to the Authentication specification and other product information can be found at <https://jakarta.ee/specifications/authentication/3.1/>.
- **Authentication Version:** The Authentication 3.1 TCK is based on the Authentication Specification, Version 3.1.
- **Compatible Implementation:** One Authentication 3.1 Compatible Implementation, Eclipse GlassFish 8.0 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 Authentication TCK Release Notes for more specific information about Java SE version requirements, supported platforms, restrictions, and so on.

1.2.2 TCK Components

The Authentication TCK 3.1 includes the following components:

- 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 Authentication API in all related packages:
 - `jakarta.security.auth.message`
 - `jakarta.security.auth.message.callback`
 - `jakarta.security.auth.message.config`
 - `jakarta.security.auth.message.module`

The Authentication TCK tests run on the following platforms:

- CentOS Linux 7
- Alpine Linux v3.12

1.2.3 TCK Compatibility Test Suite

The test suite is the collection of JUnit tests used with the Maven Surefire plugin to test a particular technology implementation. In this case, it is the collection of tests used by the Authentication TCK 3.1 to test a Authentication 3.1 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.

1.2.4 TCK Configuration

You need to set several variables in your test environment, modify properties and run the Authentication tests, as described in [Chapter 4, "Setup and Configuration."](#)

1.2.7 Authentication Technology Overview

The Authentication 3.1 Specification defines a service provider interface (SPI) by which authentication providers implementing message authentication mechanisms can be integrated in client and server message processing runtimes or containers.

The Authentication TCK uses a Test Suite SPI Verifier (TSSV) to verify whether the vendor's message processing runtimes invoke the correct SPI in the proper order.

TSSV includes test suite implementations of:

- `AuthConfigFactory`
- `AuthConfigProvider`
- `AuthConfigClient`, `AuthConfigServer`
- `AuthContextClient`, `AuthContextServer`
- `AuthenticationModulesClient`, `AuthenticationModules Server`

TSSV gets loaded into vendor's message processing runtime using one of the following ways, as defined by the Authentication 1.1 Specification:

- By defining a property in `JAVA_HOME/jre/lib/security/java.security` as follows:
`authconfigprovider.factory=com.sun.ts.tests.jaspic.tssv.config.TSAuthConfigFactory`
- By calling the `registerConfigProvider()` method in a vendor's `AuthConfigFactory` with the following values:
 - Test Suite Provider ClassName
 - Map of properties
 - Message Layer (such as `SOAP` or `HttpServlet`)

- Application Context Identifier
- A description of the provider



For the Authentication TCK, more than one provider is registered in the vendor's message processing runtime.

In a typical test scenario (for each profile of Servlet or SOAP), an application is deployed into a vendor's runtime, and a client invokes the service. The message policies required for the secure invocations are built into TSSV implementations, and the runtime is analyzed to see whether it invokes the correct SPIs at the proper time.

TSSV uses Java logging APIs to log the client and server invocation into a log file (`TSSVLog.txt`), this log file is used by the TCK tests to validate actual logged runtime information against expected results to ensure that the runtime is compliant. The `jaspic_util_web.war` file contains the Authentication log file processor, which writes output to the `TSSVLog.txt` file. The `TSSVLog.txt` file is put into the location defined by the `log.file.location` property in the `ts.jte` file.

1.2.8 Authentication TSSV Files

The following sections describe the `tssv.jar`, `ProviderConfiguration.xml`, and `provider-configuration.xsd` files that are used by the Authentication TCK tests.

1.2.8.1 tssv.jar file

The `tssv.jar` file contains classes necessary for populating a vendor implementation with a CTS AuthConfigFactory (ACF) as well as information used to register CTS providers. The `tssv.jar` file contains the class files for the Test Suite SPI Verifier. The `tssv.jar` file classes need to be loaded by the vendor implementation runtime during startup.

1.2.8.2 ProviderConfiguration.xml file

The format is a test suite-specific format. The file was designed to contain test provider information the test suite uses to populate the ACF with a list of providers for testing. The file needs to be copied to the location specified in the `ts.jte` file by the `provider.configuration.file` property. An edit to the `ProviderConfiguration.xml` file may be required for the vendor implementation. The current application context Ids are generic and should work as is, but there could be some scenarios in which the application Context Ids may need to be modified.

The value of the `<app-context-id>` element in the `ProviderConfiguration.xml` file should reflect what the vendor implementation will use for its internal representation of the application context identifier for a registered provider. Said differently, the test suite registers its providers with information from the `ProviderConfiguration.xml` file but every implementation is not guaranteed to use the application context identifier that is used in the call to register the configuration provider.

This value of the `<app-context-id>` element corresponds to the `appContext` argument in the `AuthConfigFactory.registerConfigProvider()` API. The API documentation for this method indicates that the `appContext` argument may be used but is not guaranteed to be used.

The default `ProviderConfiguration.xml` file should work without modification, but a vendor may need to alter the value of the `<app-context-id>` element as previously described to accommodate the implementation under test. Vendors need to find the correct application context identifier for their implementation.

Vendors should enable two levels of logging output to get finer levels of debugging and tracing information than is turned on by default. This is done by setting the `traceflag` property in the `ts.jte` file to "true" and setting the `HARNESS_DEBUG` environment variable to "true". If both of these are set, the debug output should contain application context identifier information.

1.2.8.3 provider-configuration.xsd file

The `provider-configuration.xsd` file is a schema file that resides in the same directory as the `ProviderConfiguration.xml` file and describes the `ProviderConfiguration.xml` file. This file should not be edited.

1.2.9 Baseline Compatibility Requirements

To obtain Baseline compliance, a vendor must meet the Baseline Compatibility requirements.

1.2.10 Servlet Profile Tests

To obtain Servlet Profile compliance, a vendor must meet the Baseline Compatibility requirements as well as the Servlet Profile requirements.

1.2.11 SOAP Profile Tests

Since various SOAP implementations are possible in a vendor's message processing runtime, the Authentication TCK considers the following SOAP implementations:

- SOAP implementation in a Jakarta EE environment
- SOAP implementation in standalone container (Java SE only)
- Non-container based SOAP implementation

For SOAP profile tests, the client invocations of webservice have been abstracted into two different types:

- Invocations of Service in a Jakarta EE environment (for example, using JAXWS annotations `@WebServiceRef` for looking up the service and `@WebService` for service definition).

- Invocations of Service in a standalone (i.e. Java SE only) environment (this includes standalone container and non-container based implementation).

The following are used to get the service reference:

- WSDL
- Service `QName` (for example, `QName(NAMESPACEURI, SERVICENAME)`)
- Service Class (such as `HelloService.class`)
- PORT `QName` (for example, `QName(NAMESPACEURI, PORT_NAME)`)
- Service Endpoint Interface class (for example, `Hello.class`)

The deployment abstraction for handling various SOAP implementations are handled in the following ways:

- A Jakarta Deployment version 1.2 deployment is used for Jakarta EE based implementations. This is differentiated by using a different deliverable class, `deliverable.class=com.sun.ts.lib.deliverable.jaspic.JaspicJakartaEEDeliverable`, which is configurable in the `ts.jte` file. Vendors need to write their own `Deliverable` class that can be used to deploy in their environment.
- For standalone implementations (this includes container and non-container based implementations), a different deliverable class is used, `deliverable.class=com.sun.ts.lib.deliverable.jaspic.JaspicDeliverable`. Along with this deliverable class an Ant file, `TS_HOME/bin/xml/deploy.xml`, is used to deploy in GlassFish Server. Vendors are expected to implement the `deploy` and `undeploy` targets in `deploy.xml` to suite their environment.



Two deliverable implementations are provided with the GlassFish server. One implementation, for Java SE only servers, turns off auto deployment and leaves the deployment up to the licensee by way of an Ant target.

- Along with the deliverable class, a configurable property in the `ts.jte` file, `platform.mode`, is used to distinguish the different SOAP implementations.
 - `platform.mode=jakartaEE` (for Jakarta EE based implementations)
 - `platform.mode=standalone`



A deployable EAR , WAR, or JAR file is created, based on the value specified by the `platform.mode` property in the `ts.jte` file.

- For non-container based standalone SOAP implementations, vendors are expected to deploy the service and make it available for client invocations. For this purpose, a no-op for `deploy` and `undeploy` targets can be implemented in the `deploy.xml` file.

The Authentication TCK uses Jakarta Web Services Metadata, 3.0 metadata based annotations to define web service applications. Although Jakarta Web Services Metadata, 3.0 support is not required in a vendor's SOAP implementation, using web services metadata simplifies the definition of web services and the linking between various artifacts of web services (the WSDL, `ServiceEndpoint`, and implementation and their associations). Using other forms of web services

implementation will lead to separate binding files, web services description files (`webservices.xml`) which are different for different SOAP implementations, such as a Jakarta EE based SOAP implementation, standalone implementation, and so on.

Since vendors are already expected to generate web service artifacts using `wsgen` and `wsimport` tools, writing an annotation processor to support Jakarta Web Services Metadata, 3.0 based annotations is just a step further towards making a better SOAP implementation. Also having annotated web services helps vendors generate different artifacts that suit their SOAP implementation.



For Jakarta EE based SOAP implementations, Jakarta Web Services Metadata, 3.0 support is required.



The Jakarta EE Specification Process support multiple compatible implementations. These instructions explain how to get started with the Eclipse GlassFish 8.0 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 Authentication 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 hosting the Maven Surefire plugin run:

- Java SE 11
- A CI for Authentication 3.1. One example is Eclipse GlassFish 8.0.
- Authentication TCK version 3.1, which includes:
 - JDOM 1.1.3
 - Apache Commons HTTP Client 3.1
 - Apache Commons Logging 1.1.3
 - Apache Commons Codec 1.9
- The Authentication 3.1 Vendor Implementation (VI)
See the documentation for each of these software applications for installation instructions. See [Chapter 3, "Installation,"](#) for instructions on installing the Authentication TCK.

1. Set up the Authentication TCK software.

See [Chapter 4, "Setup and Configuration,"](#) for details about the following steps.

1. Set up your shell environment.

2. Test the Authentication 3.1 implementation.

Test the Authentication 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 Authentication. This chapter contains the following sections:

- [Certification Overview](#)
- [Compatibility Requirements](#)
- [Test Appeals Process](#)
- [Specifications for Jakarta Authentication](#)
- [Libraries for Jakarta Authentication](#)

2.1 Certification Overview

The certification process for Authentication 3.1 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 Authentication 3.1 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	<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.</p> <p>Examples of computational resources that may vary in existence (that is, may or may not exist) are graphics cards and device drivers.</p> <p>Examples of computational resources that may vary in version are operating systems and device drivers.</p>
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.
Libraries	<p>The class libraries, as specified through the Jakarta EE Specification Process (JESP), for the Technology Under Test.</p> <p>The Libraries for Jakarta Authentication are listed at the end of this chapter.</p>

Term	Definition
Location Resource	<p>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.</p> <p>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.</p>
Maintenance Lead	<p>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.</p>
Operating Mode	<p>Any Documented option of a Product that can be changed by a user in order to modify the behavior of the Product.</p> <p>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).</p> <p>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.</p>
Product	<p>A vendor's product in which the Technology Under Test is implemented or incorporated, and that is subject to compatibility testing.</p>
Product Configuration	<p>A specific setting or instantiation of an Operating Mode.</p> <p>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.</p>
Rebuildable Tests	<p>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.</p>
Resource	<p>A Computational Resource, a Location Resource, or a Security Resource.</p>
Rules	<p>These definitions and rules in this Compatibility Requirements section of this User's Guide.</p>
Runtime	<p>The Containers specified in the Specifications.</p>
Security Resource	<p>A security privilege or policy necessary for the proper execution of the Test Suite.</p> <p>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.</p>

Term	Definition
Specifications	<p>The documents produced through the Jakarta EE Specification Process (JESP) that define a particular Version of a Technology.</p> <p>The Specifications for the Technology Under Test are referenced later in this chapter.</p>
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 Authentication Version 3.1.
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 Authentication Products

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

Authentication1 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.

Authentication1.1 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.

Authentication1.2 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.

Authentication1.3 An API Definition Product is exempt from all functional testing requirements defined here, except the signature tests.

Authentication2 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 Maven pom.xml file of the test module. 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.

Authentication3 The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

Authentication4 The Exclude List associated with the Test Suite cannot be modified.

Authentication5 The Maintenance Lead can define exceptions to these Rules. Such exceptions would be made available as above, and will apply to all vendor implementations.

Authentication6 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.

Authentication7 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.

Authentication7.1 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.

Authentication8 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.

Authentication9 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 Authentication 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 Authentication 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 Authentication TCK.

2.3.3 TCK Test Appeals Steps

1. Challenges should be filed via the Jakarta Authentication 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 will be closed. If the appeal is rejected, the **challenge-appeal** issue should be 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 Authentication

The Jakarta Authentication specification is available from the specification project web-site: <https://jakarta.ee/specifications/authentication/3.1/>.

2.5 Libraries for Jakarta Authentication

The following is a list of the packages comprising the required class libraries for Authentication 3.1:

- `jakarta.security.auth.message`
- `jakarta.security.auth.message.callback`
- `jakarta.security.auth.message.config`
- `jakarta.security.auth.message.module`

For the latest list of packages, also see:

<https://jakarta.ee/specifications/authentication/3.1/>

3 Installation

This chapter explains how to install the Jakarta Authentication 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.

3.1 Obtaining a Compatible Implementation

Each compatible implementation (CI) will provide instructions for obtaining their implementation. Eclipse GlassFish 8.0 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 Authentication TCK tests, you must install and set up the following software components:

- Java SE 11
- Apache Maven 3.8.6+
- A CI for Authentication 3.1, one example is Eclipse GlassFish 8.0
- Authentication TCK version 3.1
- The Authentication 3.1 Vendor Implementation (VI)

Follow these steps:

1. Install the Java SE 11 software, if it is not already installed.
Download and install the Java SE 11 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 Apache Maven 3.8.6+ software, if it is not already installed.
Download and install Apache Maven 3.8.6+ software from Apache Maven Project.
3. Install the Authentication TCK 3.1 software.
 1. Copy or download the Authentication TCK software to your local system.
You can obtain the Authentication TCK software from the Jakarta EE site <https://jakarta.ee/specifications/authentication/3.1/>.
 2. Use the `unzip` command to extract the bundle in the directory of your choice:
`unzip jakarta-authentication-tck-3.1.0.zip`
This creates the TCK directory. The TCK is the test suite home, `<TS_HOME>`.
4. Install the Jakarta EE 11.0 CI software (the servlet Web container used for running the

Authentication TCK with the Authentication 3.1 CI), if it is not already installed.

Download and install the Servlet Web container with the Authentication 3.1 CI used for running the Authentication TCK 3.1, represented by the Jakarta EE 11.0 CI. One CI is Eclipse GlassFish 8.0. You may obtain a copy of this CI by downloading it from <https://projects.eclipse.org/projects/ee4j.glassfish>.

5. Install a Authentication 3.1 Compatible Implementation.

A Compatible Implementation is used to validate your initial configuration and setup of the Authentication TCK 3.1 tests, which are explained further in [Chapter 4, "Setup and Configuration."](#)

The Compatible Implementations for Authentication are listed on the Jakarta EE Specifications web site: <https://jakarta.ee/specifications/authentication/3.1/>.

6. Install a Web server on which the Authentication TCK test applications can be published for testing the VI.

7. Install the Authentication 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 Authentication 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 Authentication TCK.



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.

4.1 Download the Jakarta Authentication TCK

The Jakarta Authentication TCK is retrieved from

```
https://download.eclipse.org/jakartaee/authentication/3.1/jakarta-authentication-tck-3.1.0.zip
```

4.2 Setting up an Environment for the Maven Surefire based TCK Against a Compatible Implementation

The current JUnit based set of tests runs under a Maven surefire runner and needs a profile configured for your compatible implementation to test. The top level tck/pom.xml includes example profiles including one for GlassFish. To configuration a profile for your compatible implementation, start with the GlassFish glassfish-ci-managed profile, and modify the dependencies and configuration to support your implementation and Arquillian container implementation. The Arquillian container implementation starts up your container and deploys the test wars into it.

4.3 Setting up an Environment Against a Jakarta EE 11.0 CI for SPI TCK tests ~~~~~

The SPI tests (in [tck]/spi which are part of the latest Jakarta Authentication TCK need additional configuration.

These SPI tests dive a little deeper into the assertions of the Jakarta Authentication SPI, and directly test for a number of technical expectations. In contrast, most other tests in the Jakarta Authentication TCK look more at observable system behavior and are more akin to Integration Tests (IT).

The common sub-module ([tck]/spi/module) of the SPI module primarily contains a special AuthConfigFactory (the TSSV) that is installed using a ServletContainerInitializer. This AuthConfigFactory traces a lot of operations on it to a logging file (saved to location set by system property "log.file.location"). This logging file is read by the (client side) tests, which verify whether specific entries are present or absent in said log file.

Both the CI (e.g. Eclipse GlassFish) and the client environment running the tests (junit) needs to have this "log.file.location" specified and pointing to the same location.

Among the SPI tests is a test for the "auth context ID", which can be obtained in a standard way, but contains an implementation specific value. In order to test that this value is correct, the expected value for a CI needs to be configured using the client environment system property "logical.hostname.servlet".

A Jakarta Authentication implementation provides an "AuthConfigFactory" implementation, which can be obtained in a standard way, but obviously is of a implementation specific type. In order to test various assumptions about this "vendor AuthConfigFactory", the exact type needs to be provided to the tests via the system property "vendor.authconfig.factory". Both the CI (e.g. GlassFish) and the client environment running the tests (junit) needs to have this "vendor.authconfig.factory" specified and set to the same value.

The `tck/pom.xml` file contains a full example for Eclipse GlassFish in the "glassfish-ci-managed" profile in the surefire configuration section.

5 Executing Tests

The Authentication TCK uses the JUnit and Maven Surefire harness to execute the tests in the test suite.

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.

5.1 Starting the tests

Run the Authentication TCK from the command line in your shell environment



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 the tests in Command-Line Mode

To run all tests against GlassFish, use following command:

```
mvn -Pglassfish-ci-managed clean test surefire-report:report
```

This will produce a `target/site/surefire-report.html` summary. Replace the `glassfish-ci-managed` profile name with the name of any other configured compatible implementation to run the tests against that implementation.

When doing testing with staged dependencies, you may need to add the `-Pstaging` profile:

```
mvn -Pstaging -Pglassfish-ci-managed clean test surefire-report:report
```

When running tests against Jakarta EE Compatible Implementation you need to specify the

-Pplatform profile: This will set the **KEYWORDS** parameter accordingly for choosing the tests to be run for Platform mode (Full profile in default).

```
mvn -Pplatform clean test surefire-report:report
```

When running tests against Jakarta EE Web Profile Compatible Implementation you need to specify the **-Pplatform,web** profiles: This will set the **KEYWORDS** parameter accordingly for choosing the tests to be run for Platform mode in web profile. Also the glassfish web profile bundle will be chosen to run the test against.

```
mvn -Pplatform,web clean test surefire-report:report
```

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

1. Change to the directory containing the tests you want to run.
2. Start the test run by executing the following command:

```
mvn clean verify
```

The tests in the directory are run.

5.3 Running the TCK Against GlassFish

All the tests can be run against GlassFish by invoking the following from the `tck/` directory: **mvn clean verify**.

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.

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.

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

6.2 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.

A Frequently Asked Questions

This appendix contains the following questions.

- [Where do I start to debug a test failure?](#)
- [What would cause tests be added to the exclude list?](#)

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

The Maven Surefire Plugin used to run the TCK tests, describes how to run the plugin under a debugger: <https://maven.apache.org/surefire/maven-surefire-plugin/examples/debugging.html>

A.2 What would cause tests be added to the exclude list?

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 Authentication TCK.