Jakarta EE Platform, Enterprise Edition 10 Test Compatibility Kit User's Guide, Release 10 for Jakarta EE

Table of Contents

Eclipse Foundation™

Jakarta Platform, Enterprise Edition 10 Test Compatibility Kit User's Guide

Release 10 for Jakarta EE

May 2021

Provides detailed instructions for obtaining, installing, configuring, and using the Eclipse Jakarta, Enterprise Edition 10 Compatibility Test Suite for the Full Profile and the Web Profile.

Jakarta Platform, Enterprise Edition 10 Test Compatibility Kit User's Guide, Release 10 for Jakarta EE

Emeritus Author: Eric Jendrock

> References in this document to Java EE refer to the Jakarta EE unless otherwise noted.

Preface

> ℹ️ The Jakarta Enterprise Edition documentation is part of the Jakarta Enterprise Edition contribution to the Eclipse Foundation and is not intended for use in relation to Java Enterprise Edition or Java Licensee requirements. This documentation is in the process of being revised to reflect the new Jakarta EE requirements and branding. Additional changes will be made as requirements and procedures evolve for Jakarta EE. Where applicable, references to Java EE or Java Enterprise Edition should be considered references to Jakarta EE.
>
> Please see the Title page for additional license information.

This book introduces the Test Compatibility Kit (TCK) for the Jakarta Platform, Enterprise Edition 10 (Jakarta EE 10) and Jakarta Platform, Enterprise Edition 10 Web Profile (Jakarta EE 10 Web Profile), and explains how to configure and run the test suite. It also provides information for troubleshooting problems you may encounter as you run the test suite.

The Jakarta Platform, Enterprise Edition 10 Test Compatibility Kit (Jakarta EE 10 TCK) is a portable, configurable automated test suite for verifying the compatibility of an implementer's compliance with the Jakarta EE 10 Specification (hereafter referred to as the implementer's implementation, or VI). The Jakarta EE 10 Platform TCK uses the JavaTest harness version 5.0 to run the test suite.

> ℹ️ URLs are provided so you can locate resources quickly. However, these URLs are subject to changes that are beyond the control of the authors of this guide.

Who Should Use This Book

This guide is for developers of the Jakarta EE 10 technology to assist them in running the test suite that verifies compatibility of their implementation of the Jakarta EE 10 Specification.

Before You Read This Book

Before reading this guide, you should familiarize yourself with the Java programming language, the Jakarta Platform, Enterprise Edition 10 (Jakarta EE 9) Specification, and the JavaTest documentation.

The Jakarta Platform, Enterprise Edition 10 (Jakarta EE 10) Specification can be downloaded from https://projects.eclipse.org/projects/ee4j.jakartaee-platform.

For documentation on the test harness used for running the Jakarta EE 10 Platform TCK test suite, see https://wiki.openjdk.java.net/display/CodeTools/Documentation.

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	`$`
Bourne shell and Korn shell for superuser	`#`
Bash shell	`shell_name-shell_version$`

Shell	Prompt
Bash shell for superuser	`shell_name-shell_version#`

Shell	Prompt
	`shell_name-shell_version#`

1 Introduction

This document provides instructions for installing, configuring, and running the Jakarta Platform, Enterprise Edition 10 Test Compatibility Kit (Jakarta EE 10 Platform TCK).

This chapter includes the following topics:

- Compatibility Testing

- About Jakarta EE 10 Platform TCK

- Hardware Requirements

- Software Requirements

- Additional Jakarta EE 10 Platform TCK Requirements

- Getting Started With the Jakarta EE 10 Platform TCK Test Suite

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 compabitle implementation for that feature. Compatibility testing is not primarily concerned with robustness, performance, or ease of use.

1.1.1 Why Compatibility Testing is Important

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 Java programming language, allowing them to write applications once and then to deploy them across heterogeneous computing environments without porting.

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

- Conformance testing benefits Jakarta Platform implementors by ensuring a level playing field for all Jakarta Platform ports.

1.1.2 Compatibility Rules

Compatibility criteria for all technology implementations are embodied in the Compatibility Rules that apply to a specified technology. The Jakarta EE 10 Platform TCK tests for adherence to these Rules as described in Chapter 2, "Procedure for Jakarta Platform, Enterprise Edition 10 Certification," for Jakarta EE 10 and Chapter 3, "Procedure for Jakarta Platform, Enterprise Edition 10 Web Profile Certification," for Jakarta EE 10 Web Profile.

1.1.3 TCK Overview

A Jakarta EE 10 Platform TCK is a set of tools and tests used to verify that a Implementer's implementation of Jakarta EE 10 technology conforms to the applicable specification. All tests in the TCK are based on the written specifications for the Jakarta Platform. The TCK tests compatibility of a Implementer's implementation of a 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 Implementers.

The set of tests included with the Jakarta EE 10 Platform TCK is called the test suite. All tests in the TCK test suite are self-checking, but some tests may require tester interaction. Most tests return either a Pass or Fail status. For a given platform to be certified, all of the required tests must pass. The definition of required tests may change from platform to platform.

The definition of required tests will change over time. Before your final certification test pass, be sure to download the latest Exclude List for the Jakarta EE 10 Platform TCK. The definition of required tests will change over time. See Section 1.2.5, "Exclude Lists," for more information.

1.1.4 Jakarta Specification Community Process 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 specification project committer group:

- Technology Specification

- A Compatible Implementation

- 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 Jakarta EE 10 Platform TCK

Jakarta EE 10 Platform TCK is a portable, configurable, automated test suite for verifying the compliance of a Implementer's implementation of the Jakarta EE 10 technologies. Jakarta EE 10 Platform TCK includes version 5.0 of the JavaTest harness.

For documentation on the test harness used for running the Jakarta EE 10 Platform TCK test suite, see `https://wiki.openjdk.java.net/display/CodeTools/Documentation`.

1.2.1 Jakarta EE 10 Technologies Tested with Jakarta EE 10 Platform TCK

The Jakarta EE 10 Platform TCK test suite includes compatibility tests for the following required and optional Jakarta EE 10 technologies:

- Jakarta Enterprise Beans 4.0

- Jakarta Server Pages 3.1

- Jakarta Servlet 6.0

- Jakarta Expression Language 5.0

- Jakarta Messaging 3.0

- Jakarta Transactions 2.0

- Jakarta Mail 2.0

- Jakarta Connectors 2.0

- Jakarta Enterprise Web Services 2.0 (optional)

- Jakarta RESTful Web Services 3.0

- Jakarta WebSocket 2.1

- Jakarta JSON Processing 2.0

- Jakarta JSON Binding 2.0

- Jakarta Concurrency 2.0

- Jakarta Authorization 2.0

- Jakarta Authentication 2.0

- Jakarta Standard Tag Library 2.0

- Jakarta Faces 3.0

- Jakarta Security 2.0

- Jakarta Annotations 2.0

- Jakarta Persistence 3.0

- Jakarta Bean Validation 3.0

- Jakarta Managed Beans 2.0

- Jakarta Interceptors 2.0

- Jakarta Contexts and Dependency Injection 3.0

- Jakarta Dependency Injection 2.0

- Jakarta Debugging Support for Other Languages 2.0

- Jakarta Enterprise Beans 3.2 and earlier entity beans and associated Jakarta Enterprise Beans QL (optional)

- Jakarta Enterprise Beans 2.x API group (optional)

- Jakarta Enterprise Web Services 2.0 (optional)

- Jakarta SOAP with Attachments 2.0 (optional)

- Jakarta Web Services Metadata 3.0 (optional)

- Jakarta XML Web Services 3.0 (optional)

1.2.2 Jakarta EE 10 Web Profile Technologies Tested With Jakarta EE 10 Platform TCK

The Jakarta EE 10 Platform TCK test suite can also be used to test compatibility for the following required Jakarta EE 10 Web Profile technologies:

- Jakarta Server Pages 3.1

- Jakarta Servlet 6.0

- Jakarta Expression Language 5.0

- Jakarta Standard Tag Library 2.0

- Jakarta Faces 3.0

- Jakarta RESTful Web Services 3.0

- Jakarta WebSocket 2.1

- Jakarta JSON Processing 2.0

- Jakarta JSON Binding 2.0

- Jakarta Common Annotations 2.0

- Jakarta Enterprise Beans 4.0, Lite

- Jakarta Transactions 2.0

- Jakarta Persistence 3.0

- Jakarta Bean Validation 3.0

- Jakarta Managed Beans 2.0

- Jakarta Interceptors 2.0

- Jakarta Contexts and Dependency Injection 3.0

- Jakarta Dependency Injection 2.0

- Jakarta Security 2.0

- Jakarta Authentication 2.0, Servlet Container Profile

- Jakarta Debugging Support for Other Languages 2.0

1.2.3 TCK Tests

The Jakarta EE 10 Platform TCK contains API tests and enterprise edition tests, which are tests that start in the Jakarta EE 10 platform and use the underlying enterprise service or services as specified. For example, a JDBC enterprise edition test connects to a database, uses SQL commands and the JDBC 4.2 API to populate the database tables with data, queries the database, and compares the returned results against the expected results.

Figure 1-1 Typical Jakarta Platform, Enterprise Edition Workflow
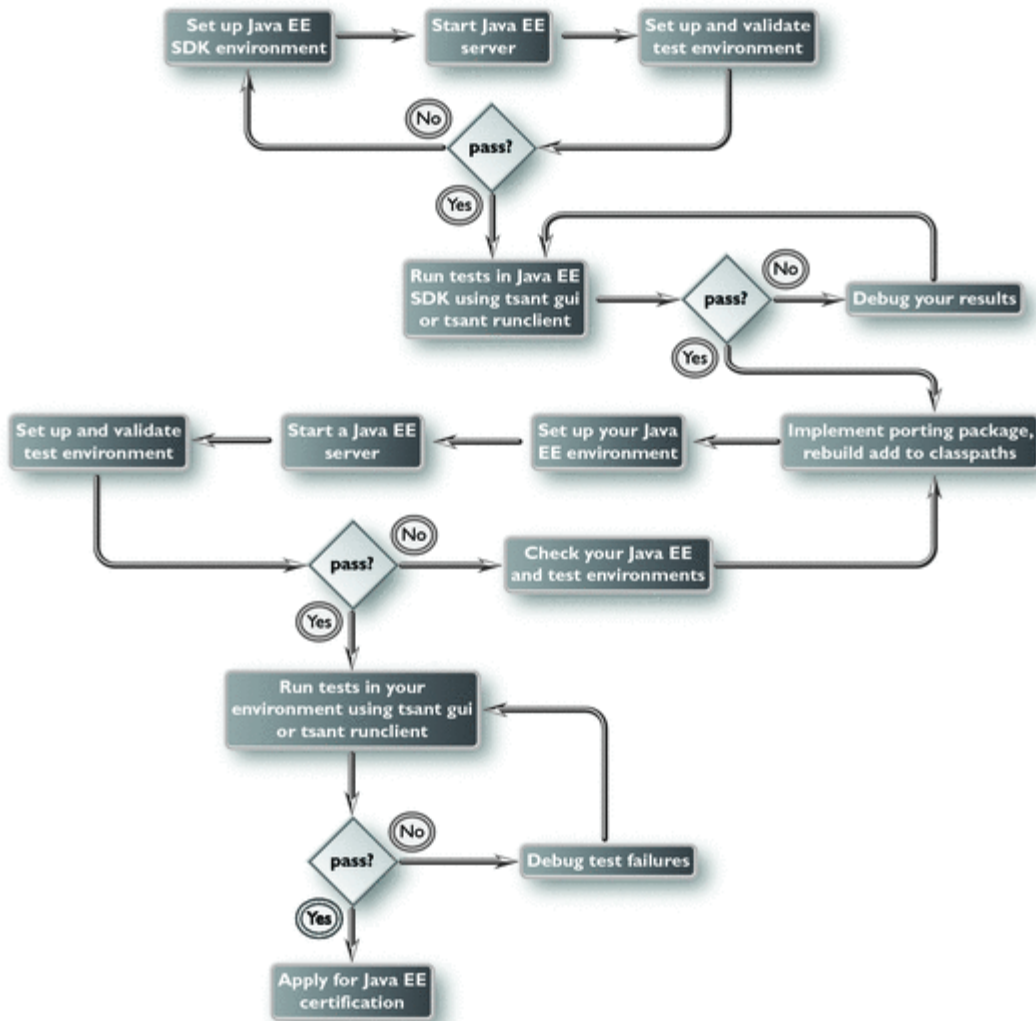
Note: References in diagram to Java EE refer to Jakarta EE.

Figure 1-1 shows how most Implementers will use the test suite. They will set up and run the test suite with the Jakarta Platform, Enterprise Edition 10 Compatible Implementation (Jakarta EE 10 CI) first to become familiar with the testing process. Then they will set up and run the test suite with their own Jakarta EE 10 implementation. This is called the Vendor Implementation, or VI in this document. When they pass all of the tests, they will apply for and be granted certification.

- Before you do anything with the test suite, read the rules in Chapter 2, "Procedure for Jakarta Platform, Enterprise Edition 8 Certification," or Chapter 3, "Procedure for Jakarta Platform, Enterprise Edition 10 Web Profile Certification." These chapters explain the certification process and provides a definitive list of certification rules for Jakarta EE 10 and Jakarta EE 10 Web Profile implementations.

- Next, take a look at the test assertions in the Assertion List, which you can find in the Jakarta EE 10 Platform TCK documentation bundle. The assertions explain what each test is testing. When you run the tests with the JavaTest GUI, the assertion being tested as part of the test description of the currently selected test is displayed.

- Third, install and configure the Jakarta EE 10 Platform TCK software and the Jakarta EE 10 CI or

Jakarta EE 10 Web Profile CI and run the tests as described in this guide. This will familiarize you with the testing process.

- Finally, set up and run the test suite with your own Jakarta EE 10 or Jakarta EE 10 Web Profile implementation.

> In the instructions in this document, variables in angle brackets need to be expanded for each platform. For example, `<TS_HOME>` becomes `$TS_HOME` on Solaris/Linux and `%TS_HOME%` on Windows. In addition, the forward slashes (`/`) used in all of the examples need to be replaced with backslashes (`\`) for Windows.

1.2.4 JavaTest Harness

The JavaTest harness version 5.0 is a set of tools designed to run and manage test suites on different Java platforms. The JavaTest harness can be described as both a Java application and a set of compatibility testing tools. It can run tests on different kinds of Java platforms and it allows the results to be browsed online within the JavaTest GUI, or offline in the HTML reports that the JavaTest harness generates.

The JavaTest harness includes the applications and tools that are used for test execution and test suite management. It supports the following features:

- Sequencing of tests, allowing them to be loaded and executed automatically

- Graphic user interface (GUI) for ease of use

- Automated reporting capability to minimize manual errors

- Failure analysis

- Test result auditing and auditable test specification framework

- Distributed testing environment support

To run tests using the JavaTest harness, you specify which tests in the test suite to run, how to run them, and where to put the results as described in Chapter 7, "Executing Tests."

The tests that make up the TCK 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.

1.2.5 Exclude Lists

The Jakarta EE 10 Platform TCK includes an Exclude List contained in a `.jtx` 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 JavaTest harness automatically excludes any test on the Exclude List from being executed.

A implementor is not required to pass or run any test on the Exclude List. The Exclude List file, `<TS_HOME>/bin/ts.jtx`, is included in the Jakarta EE 10 TCK.

> **ℹ** Always make sure you are using an up-to-date copy of the Exclude List before running the Jakarta EE 10 Platform TCK test suite 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 cannot be run on a compatible Jakarta Platform may be put on the Exclude List if the Specification project team agrees the test is invalid. 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.

> **ℹ** Implementers 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, "Jakarta Platform, Enterprise Edition TCK Test Appeals Steps," and Section 3.3.1, "Jakarta Platform, Enterprise Edition TCK Test Appeals Steps."

1.2.6 Apache Ant

The Jakarta EE 10 CI, Jakarta EE 10 Web Profile CI, and Jakarta EE 10 Platform TCK require implementations of Apache Ant 1.9.7 from the Apache Ant Project (`http://ant.apache.org/`). Apache Ant is a free, open-source, Java-based build tool, similar in some ways to the make tool, but more flexible, cross-platform compatible, and centered around XML-based configuration files.

Ant is invoked in the Jakarta EE 10 CI, Jakarta EE 10 Web Profile CI, and Jakarta EE 10 TCK in conjunction with various XML files containing Ant targets. These Ant targets provide a convenient way to automate various configuration tasks for Jakarta EE 10 Platform TCK. For example, the initial configuration of the Jakarta EE 10 CI or Jakarta EE 10 Web Profile CI for TCK is done by means of the `config.vi` Ant target.

The Ant configuration targets are there for your convenience. When configuring your Jakarta EE 10 or Jakarta EE 10 Web Profile implementation for the Jakarta EE 10 Platform TCK, you can either set up

your environment to use the Ant tools, or you can perform some or all of your configuration procedures manually. Jakarta EE 10 Platform TCK includes the Ant Contrib package, and the tasks included with Ant Contrib are used within the TCK build files. See `http://ant-contrib.sourceforge.net/` for more information about Ant Contrib.

This User's Guide does not provide in-depth instruction on Ant internals or how to configure Ant targets for your particular Jakarta EE 10 or Jakarta EE 8 Web Profile implementation. For complete information about Ant, refer to the extensive documentation on the Apache Ant Project site. The Apache Ant Manual is available at `http://ant.apache.org/manual/index.html`.

Apache Ant is protected under the Apache Software, License 2.0, which is is available on the Apache Ant Project license page at `http://ant.apache.org/license.html`.

Installing Apache Ant

- Download the Apache Ant 1.9.7 binary bundle from the Apache Ant Project.

- Change to the directory in which you want to install Apache Ant and extract the bundle

- Set the `ANT_HOME` environment variable to point to the `apache-ant-<version>` directory

- Add `<ANT_HOME>/bin` directory to the environment variable `PATH`

1.3 Hardware Requirements

The following section lists the hardware requirements for the Jakarta EE 10 TCK software, using the Jakarta EE 10 CI or Jakarta EE 10 Web Profile CI. Hardware requirements for other compatible implementations will vary.

All systems should meet the following recommended hardware requirements:

- CPU running at 2.0 GHz or higher

- 4 GB of RAM or more

- 2 GB of swap space , if required

- 6 GB of free disk space for writing data to log files, the Jakarta EE 10 repository, and the database

- Network access to the Internet

1.4 Software Requirements

You can run the Jakarta EE 10 Platform TCK software on platforms running the Linux software that meet the following software requirements:

- Operating Systems:
    - CentOS Linux 7
    - Alpine Linux v3.12
- Java SE 8 (1.8) or 11
- Jakarta EE 10 CI or Jakarta EE 10 Web Profile CI
- Mail server that supports the IMAP and SMTP protocols
- One of the following databases:
    - MySQL
    - Apache Derby

1.5 Additional Jakarta EE 10 Platform TCK Requirements

In addition to the instructions and requirements described in this document, all Jakarta EE 10 and Jakarta EE 10 Web Profile implementations must also pass the standalone TCKs for the following technologies:

- Jakarta Batch 2.1
- Jakarta Contexts and Dependency Injection 4.0
- Jakarta Dependency Injection 2.0
- Jakarta Bean Validation 3.0
- Jakarta XML Binding 3.0
- Jakarta JSON Binding 3.0
- Jakarta JSON Processing 2.1
- Jakarta RESTful Web Services 3.1

For more information about the Jakarta Batch technology, see the specification at https://jakarta.ee/specifications/batch/2.1/

For more information about the Jakarta Contexts and Dependency Injection technology, see the specification at https://jakarta.ee/specifications/cdi/4.0/

For more information about the Jakarta Dependency Injection, see the specification at https://jakarta.ee/specifications/dependency-injection/2.0/

For more information about the Jakarta Bean Validation technology, see the specification at https://jakarta.ee/specifications/bean-validation/3.0/

For more information about the Jakarta XML Binding technology, see the specification at https://jakarta.ee/specifications/xml-binding/3.0/

For more information about the Jakarta JSON Binding technology, see the specification at https://jakarta.ee/specifications/jsonb/3.0/

For more information about the Jakarta JSON Processing technology, see the specification at https://jakarta.ee/specifications/jsonp/2.1/

For more information about the Jakarta RESTful Web Services technology, see the specification at https://jakarta.ee/specifications/restful-ws/3.1/

1.6 Getting Started With the Jakarta EE 10 Platform TCK Test Suite

Installing, configuring, and using the Jakarta EE 10 Platform TCK involves the following general steps:

1. Download, install, and configure a Jakarta EE 10 CI or Jakarta EE 10 Web Profile CI. For example Eclipse GlassFish 6.1.

2. Download and install the Jakarta EE 10 Platform TCK package.

3. Configure your database to work with your CI.

4. Configure TCK to work with your database and CI.

5. Run the TCK tests.

The remainder of this guide explains these steps in detail. If you just want to get started quickly with the Jakarta EE 10 Platform TCK using the most basic test configuration, refer to Chapter 4, "Installation."

Procedure for Jakarta Platform, Enterprise Edition 10.0 Certification

2 Procedure for Jakarta Platform, Enterprise Edition 10.0 Certification

This chapter describes the compatibility testing procedure and compatibility requirements for Jakarta Platform, Enterprise Edition Version 10.

This chapter contains the following sections:

- Certification Overview

- Compatibility Requirements

- Jakarta Platform, Enterprise Edition Version 10 Test Appeals Process

- Specifications for Jakarta Platform, Enterprise Edition Version 10

- Libraries for Jakarta Platform, Enterprise Edition Version 8

2.1 Certification Overview

The certification process for Jakarta EE 10.0 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 Section 2.2, "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 Jakarta EE 10.0 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.
Application	A collection of components contained in a single application package (such as an EAR file or JAR file) and deployed at the same time using a Deployment Tool.
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.
Deployment Tool	A tool used to deploy applications or components in a Product, as described in the Specifications.
Development Kit	A software product that implements or incorporates a Compiler, a Schema Compiler, a Schema Generator, a Java-to-WSDL Tool, a WSDL-to-Java Tool, and an RMI Compiler.

Term	Definition
Documented	Made technically accessible and made known to users, typically by means such as marketing materials, product documentation, usage messages, or developer support programs.
Edition	A Version of the Java Platform. Editions include Java Platform Standard Edition and Java Platform Enterprise Edition.
Endorsed Standard	A Java API defined through a standards process other than the Jakarta Enterprise Specification Process. The Endorsed Standard packages are listed later in this chapter.
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.
Java-to-WSDL Output	Output of a Java-to-WSDL Tool that is required for Web service deployment and invocation.
Java-to-WSDL Tool	A software development tool that implements or incorporates a function that generates web service endpoint descriptions in WSDL and XML schema format from Source Code as specified by the JAXWS Specification.
Jakarta Server Page	A text-based document that uses Jakarta Server Pages technology.
Jakarta Server Page Implementation Class	A program constructed by transforming the Jakarta Server Page text into a Java language program using the transformation rules described in the Specifications.
Libraries	The class libraries, as specified through the Jakarta EE Specification Process (JESP), for the Technology Under Test. The Libraries for Jakarta Platform, Enterprise Edition Version 10 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.

Term	Definition
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. Eclipse Jakarta EE Specification Committee is the Maintenance Lead for Jakarta Platform, Enterprise Edition Version 10.
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 of a Runtime can be binary (enable/disable optimization), an enumeration (select from a list of localizations), or a range (set the initial Runtime heap size). 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.
Compatible Implementation (CI)	A verified compatible implementation of a Specification.
Resource	A Computational Resource, a Location Resource, or a Security Resource.
Rules	These definitions and rules in this Compatibility Requirements section of this User's Guide.
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.

Term	Definition
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 Platform, Enterprise Edition Version 10.
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).
WSDL-to-Java Output	Output of a WSDL-to-Java tool that is required for Web service deployment and invocation.
WSDL-to-Java Tool	A software development tool that implements or incorporates a function that generates web service interfaces for clients and endpoints from a WSDL description as specified by the JAXWS Specification.

2.2.2 Rules for Jakarta Platform, Enterprise Edition Version 10 Products

The following rules apply for each implementation:

EE1 The Product must be able to satisfy all applicable compatibility requirements, including passing all required TCK tests.

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.

EE1.1 Each implementation must have at least one configuration that can be used to pass all required TCK Tests, although such configuration may need adjustment (e.g. whether statically or via administrative tooling).

EE1.2 An implementation may have mode(s) that provide compatibility with previous Jakarta EE versions.

EE1.3 An API Definition Product is exempt from all functional testing requirements defined here, except the signature tests.

EE2 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 lib directory of 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.

EE3 The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

EE4 The Exclude List associated with the Test Suite cannot be modified.

EE5 The Maintenance Lead may define exceptions to these Rules. Such exceptions would be made available as above, and will apply to all vendor implementations.

EE6 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.

EE7 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.

EE7.1 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.

EE7.2 A Product may provide a newer version of an Endorsed Standard. Upon request, the Maintenance Lead will make available alternate Conformance Tests as necessary to conform with such newer version of an Endorsed Standard. Such alternate tests will be made available to and apply to all implementers. If a Product provides a newer version of an Endorsed Standard, the version of the Endorsed Standard supported by the Product must be Documented.

EE7.3 The Maintenance Lead may authorize the use of newer Versions of a Technology included in the Technology Under Test. A Product that provides a newer Version of a Technology must meet the Compatibility Requirements for that newer Version, and must Document that it supports the newer Version.

For example, the Jakarta Platform, Enterprise Edition Maintenance Lead could authorize use of a newer version of a Java technology such as Jakarta XML Web Services.

EE8 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.

EE9 The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.

EE9.1 A Product may contain Operating Modes that meet all of these requirements, except Rule EE9, provided that:

1. The Operating Modes must not violate the Java Platform, Standard Edition Rules.

2. Some Product Configurations of such Operating Modes may provide only a subset of the functional programmatic behavior required by the Specifications. The behavior of applications that use more than the provided subset, when run in such Product Configurations, is unspecified.

3. The functional programmatic behavior of any binary class or interface in the above defined subset must be that defined by the Specifications.

4. Any Product Configuration that invokes this rule must be clearly Documented as not fully meeting the requirements of the Specifications.

EE10 Each Container must make technically accessible all Java SE Runtime interfaces and functionality, as defined by the Specifications, to programs running in the Container, except only as specifically exempted by these Rules.

EE10.1 Containers may impose security constraints, as defined by the Specifications.

EE11 A web Container must report an error, as defined by the Specifications, when processing a Jakarta Server Page that does not conform to the Specifications.

EE12 The presence of a Java language comment or Java language directive in a Jakarta Server Page that specifies "java" as the scripting language, when processed by a web Container, must not cause the functional programmatic behavior of that Jakarta Server Page to vary from the functional programmatic behavior of that Jakarta Server Page in the absence of that Java language comment or Java language directive.

EE13 The contents of any fixed template data (defined by the Specifications) in a Jakarta Server Page, when processed by a web Container, must not affect the functional programmatic behavior of that Jakarta Server Page, except as defined by the Specifications.

EE14 The functional programmatic behavior of a Jakarta Server Page that specifies "java" as the scripting language must be equivalent to the functional programmatic behavior of the Jakarta Server Page Implementation Class constructed from that Jakarta Server Page.

EE15 A Deployment Tool must report an error when processing a Configuration Descriptor that does not conform to the Specifications.

EE16 The presence of an XML comment in a Configuration Descriptor, when processed by a Deployment Tool, must not cause the functional programmatic behavior of the Deployment Tool to vary from the functional programmatic behavior of the Deployment Tool in the absence of that

comment.

EE17 A Deployment Tool must report an error when processing an Jakarta Enterprise Beans deployment descriptor that includes an Jakarta Enterprise Beans QL expression that does not conform to the Specifications.

EE18 The Runtime must report an error when processing a Configuration Descriptor that does not conform to the Specifications.

EE19 An error must be reported when processing a configuration descriptor that includes a Java Persistence QL expression that does not conform to the Specifications.

EE20 The presence of an XML comment in a Configuration Descriptor, when processed by the Runtime, must not cause the functional programmatic behavior of the Runtime to vary from the functional programmatic behavior of the Runtime in the absence of that comment.

EE21 Compliance testing for Jakarta EE 10.0 consists of running Jakarta EE 10.0 TCK and the following Technology Compatibility Kits (TCKs):

- Jakarta Authentication 3.0
- Jakarta Batch 2.1
- Jakarta Bean Validation 3.0
- Jakarta Concurrency 3.0
- Jakarta Contexts and Dependency Injection 4.0
- Jakarta Dependency Injection 2.0
- Jakarta Faces 4.0
- Jakarta JSON Processing 2.1
- Jakarta JSON Binding 3.0
- Jakarta RESTful Web Services 3.1
- Jakarta Security 3.0
- Jakarta XML Binding 4.0 (If XML Binding is supported)

In addition to the compatibility rules outlined in this TCK User's Guide, Jakarta EE 10.0 implementations must also adhere to all of the compatibility rules defined in the User's Guides of the aforementioned TCKs.

EE21.1 If the Jakarta EE 10 implementation uses a runtime which has already been validated by the Technology Compatibility Kit, the Jakarta EE 10 implementation may use result of such validation to claim its compliance with the Technology Compatibility Kit.

EE22 Source Code in WSDL-to-Java Output when compiled by a Reference Compiler must execute properly when run on a Reference Runtime.

EE23 Source Code in WSDL-to-Java Output must be in source file format defined by the Java Language Specification (JLS).

EE24 Java-to-WSDL Output must fully meet W3C requirements for the Web Services Description Language (WSDL) 1.1.

EE25 A Java-to-WSDL Tool must not produce Java-to-WSDL Output from source code that does not conform to the Java Language Specification (JLS).

2.3 Jakarta Platform, Enterprise Edition 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 Jakarta EE 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 {TechnologyShortName} 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 {TechnologyShortName} TCK.

2.3.3 TCK Test Appeals Steps

1. Challenges should be filed via the Jakarta EE Platform 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 Platform, Enterprise Edition Version 10.0

The Specifications for Jakarta Platform, Enterprise Edition 10.0 are found on the Eclipse Foundation, Jakarta EE Specifications web site at `https://jakarta.ee/specifications/platform/10/`. You may also find information available from the EE4J Jakarta EE Platform project page, at `https://projects.eclipse.org/projects/ee4j.jakartaee-platform`.

2.5 Libraries for Jakarta Platform, Enterprise Edition Version 10.0

The following list constitutes the complete list of packages that are required for Jakarta EE 10.0:

- jakarta.annotation
- jakarta.annotation.security
- jakarta.annotation.sql
- jakarta.batch.api
- jakarta.batch.api.chunk
- jakarta.batch.api.chunk.listener
- jakarta.batch.api.listener
- jakarta.batch.api.partition
- jakarta.batch.operations

- jakarta.batch.runtime
- jakarta.batch.runtime.context
- jakarta.decorator
- jakarta.ejb
- jakarta.ejb.embeddable
- jakarta.ejb.spi
- jakarta.el
- jakarta.enterprise.concurrent
- jakarta.enterprise.context
- jakarta.enterprise.context.control
- jakarta.enterprise.context.spi
- jakarta.enterprise.event
- jakarta.enterprise.inject
- jakarta.enterprise.inject.literal
- jakarta.enterprise.inject.se
- jakarta.enterprise.inject.spi
- jakarta.enterprise.inject.spi.configurator
- jakarta.enterprise.util
- jakarta.faces
- jakarta.faces.annotation
- jakarta.faces.application
- jakarta.faces.bean
- jakarta.faces.component
- jakarta.faces.component.behavior
- jakarta.faces.component.html
- jakarta.faces.component.search
- jakarta.faces.component.visit
- jakarta.faces.context
- jakarta.faces.convert
- jakarta.faces.el
- jakarta.faces.event
- jakarta.faces.flow

- jakarta.faces.flow.builder
- jakarta.faces.lifecycle
- jakarta.faces.model
- jakarta.faces.push
- jakarta.faces.render
- jakarta.faces.validator
- jakarta.faces.view
- jakarta.faces.view.facelets
- jakarta.faces.webapp
- jakarta.inject
- jakarta.interceptor
- jakarta.jms
- jakarta.json
- jakarta.json.bind
- jakarta.json.bind.adapter
- jakarta.json.bind.annotation
- jakarta.json.bind.config
- jakarta.json.bind.serializer
- jakarta.json.bind.spi
- jakarta.json.spi
- jakarta.json.stream
- jakarta.mail
- jakarta.mail.event
- jakarta.mail.internet
- jakarta.mail.search
- jakarta.mail.util
- jakarta.persistence
- jakarta.persistence.criteria
- jakarta.persistence.metamodel
- jakarta.persistence.spi
- jakarta.resource
- jakarta.resource.cci

- jakarta.resource.spi
- jakarta.resource.spi.endpoint
- jakarta.resource.spi.security
- jakarta.resource.spi.work
- jakarta.security.auth.message
- jakarta.security.auth.message.callback
- jakarta.security.auth.message.config
- jakarta.security.auth.message.module
- jakarta.security.enterprise
- jakarta.security.enterprise.authentication.mechanism.http
- jakarta.security.enterprise.credential
- jakarta.security.enterprise.identitystore
- jakarta.security.jacc
- jakarta.servlet
- jakarta.servlet.annotation
- jakarta.servlet.descriptor
- jakarta.servlet.http
- jakarta.servlet.jsp
- jakarta.servlet.jsp.el
- jakarta.servlet.jsp.jstl.core
- jakarta.servlet.jsp.jstl.fmt
- jakarta.servlet.jsp.jstl.sql
- jakarta.servlet.jsp.jstl.tlv
- jakarta.servlet.jsp.tagext
- jakarta.transaction
- javax.transaction.xa
- jakarta.validation
- jakarta.validation.bootstrap
- jakarta.validation.constraints
- jakarta.validation.constraintvalidation
- jakarta.validation.executable
- jakarta.validation.groups

- jakarta.validation.metadata
- jakarta.validation.spi
- jakarta.validation.valueextraction
- jakarta.websocket
- jakarta.websocket.server
- jakarta.ws.rs
- jakarta.ws.rs.client
- jakarta.ws.rs.container
- jakarta.ws.rs.core
- jakarta.ws.rs.ext
- jakarta.ws.rs.sse
- jakarta.xml.bind (optional)
- jakarta.xml.bind.annotation (optionsl)
- jakarta.xml.bind.annotation.adapters (optional)
- jakarta.xml.bind.attachment (optional)
- jakarta.xml.bind.util (optional)

Procedure for Jakarta Platform, Enterprise Edition 10 Web Profile Certification

3 Procedure for Jakarta Platform, Enterprise Edition 10 Web Profile Certification

This chapter describes the compatibility testing procedure and compatibility requirements for Jakarta Platform, Enterprise Edition Version 10 Web Profile.

This chapter contains the following sections:

- Certification Overview

- Compatibility Requirements

- Jakarta Platform, Enterprise Edition Version 10 Test Appeals Process

- Specifications for Jakarta Platform, Enterprise Edition Version 10, Web Profile

- Libraries for Jakarta Platform, Enterprise Edition Version 10, Web Profile

3.1 Certification Overview

The certification process for Jakarta EE 10, Web Profile 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.

3.2 Compatibility Requirements

The compatibility requirements for Jakarta EE 10, Web Profile consist of meeting the requirements set forth by the rules and associated definitions contained in this section.

3.2.1 Definitions

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

Table 3-1 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.
Application	A collection of components contained in a single application package (such as an EAR file or JAR file) and deployed at the same time using a Deployment Tool.
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 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.
Deployment Tool	A tool used to deploy applications or components in a Product, as described in the Specifications.
Documented	Made technically accessible and made known to users, typically by means such as marketing materials, product documentation, usage messages, or developer support programs.

Term	Definition
Edition	A Version of the Java Platform. Editions include Java Platform Standard Edition and Java Platform Enterprise Edition.
Endorsed Standard	A Java API defined through a standards process other than the Jakarta Enterprise Specification Process. The Endorsed Standard packages are listed later in this chapter.
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.
Java-to-WSDL Output	Output of a Java-to-WSDL Tool that is required for Web service deployment and invocation.
Java-to-WSDL Tool	A software development tool that implements or incorporates a function that generates web service endpoint descriptions in WSDL and XML schema format from Source Code as specified by the JAXWS Specification.
Jakarta Server Page	A text-based document that uses Jakarta Server Pages technology.
Jakarta Server Page Implementation Class	A program constructed by transforming the Jakarta Server Page text into a Java language program using the transformation rules described in the Specifications.
Libraries	The class libraries, as specified through the Jakarta EE Specification Process (JESP), for the Technology Under Test. The Libraries for Jakarta Platform, Enterprise Edition Version 10 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. Eclipse Jakarta EE Specification Committee is the Maintenance Lead for Jakarta Platform, Enterprise Edition Version 10, Web Profile.

Term	Definition
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.
Compatible Implementation (CI)	A verified compatible implementation of a Specification.
Resource	A Computational Resource, a Location Resource, or a Security Resource.
Rules	These definitions and rules in this Compatibility Requirements section of this User's Guide.
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.

Term	Definition
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 Platform, Enterprise Edition Version 10, Web Profile.
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).
WSDL-to-Java Output	Output of a WSDL-to-Java tool that is required for Web service deployment and invocation.
WSDL-to-Java Tool	A software development tool that implements or incorporates a function that generates web service interfaces for clients and endpoints from a WSDL description as specified by the JAXWS Specification.

3.2.2 Rules for Jakarta Platform, Enterprise Edition Version 10 Products

The following rules apply for each implementation:

EE-WP1 The Product must be able to satisfy all applicable compatibility requirements, including passing all required TCK tests.

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.

EE-WP1.1 Each implementation must have at least one configuration that can be used to pass all required TCK Tests, although such configuration may need adjustment (e.g. whether statically or via administrative tooling).

EE-WP1.2 An implementation may have mode(s) that provide compatibility with previous Jakarta EE versions.

EE-WP1.3 An API Definition Product is exempt from all functional testing requirements defined here, except the signature tests.

EE-WP2 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 lib directory of 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.

EE-WP3 The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

EE-WP4 The Exclude List associated with the Test Suite cannot be modified.

EE-WP5 The Maintenance Lead may define exceptions to these Rules. Such exceptions would be made available as above, and will apply to all vendor implementations.

EE-WP6 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.

EE-WP7 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.

EE-WP7.1 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.

EE-WP7.2 A Product may provide a newer version of an Endorsed Standard. Upon request, the Maintenance Lead will make available alternate Conformance Tests as necessary to conform with such newer version of an Endorsed Standard. Such alternate tests will be made available to and apply to all implementers. If a Product provides a newer version of an Endorsed Standard, the version of the Endorsed Standard supported by the Product must be Documented.

EE-WP7.3 The Maintenance Lead may authorize the use of newer Versions of a Technology included in the Technology Under Test. A Product that provides a newer Version of a Technology must meet the Compatibility Requirements for that newer Version, and must Document that it supports the newer Version.

EE-WP8 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.

EE-WP9 The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.

EE-WP9.1 A Product may contain Operating Modes that meet all of these requirements, except Rule EE-WP9, provided that:

1. The Operating Modes must not violate the Java Platform, Standard Edition Rules.

2. Some Product Configurations of such Operating Modes may provide only a subset of the functional programmatic behavior required by the Specifications. The behavior of applications that use more than the provided subset, when run in such Product Configurations, is unspecified.

3. The functional programmatic behavior of any binary class or interface in the above defined subset must be that defined by the Specifications.

4. Any Product Configuration that invokes this rule must be clearly Documented as not fully meeting the requirements of the Specifications.

EE-WP10 Each Container must make technically accessible all Java SE Runtime interfaces and functionality, as defined by the Specifications, to programs running in the Container, except only as specifically exempted by these Rules.

EE-WP10.1 Containers may impose security constraints, as defined by the Specifications.

EE-WP11 A web Container must report an error, as defined by the Specifications, when processing a Jakarta Server Page that does not conform to the Specifications.

EE-WP12 The presence of a Java language comment or Java language directive in a Jakarta Server Page that specifies "java" as the scripting language, when processed by a web Container, must not cause the functional programmatic behavior of that Jakarta Server Page to vary from the functional programmatic behavior of that Jakarta Server Page in the absence of that Java language comment or Java language directive.

EE-WP13 The contents of any fixed template data (defined by the Specifications) in a Jakarta Server Page, when processed by a web Container, must not affect the functional programmatic behavior of that Jakarta Server Page, except as defined by the Specifications.

EE-WP14 The functional programmatic behavior of a Jakarta Server Page that specifies "java" as the scripting language must be equivalent to the functional programmatic behavior of the Jakarta Server Page Implementation Class constructed from that Jakarta Server Page.

EE-WP15 A Deployment Tool must report an error when processing a Configuration Descriptor that does not conform to the Specifications.

EE-WP16 The presence of an XML comment in a Configuration Descriptor, when processed by a Deployment Tool, must not cause the functional programmatic behavior of the Deployment Tool to vary from the functional programmatic behavior of the Deployment Tool in the absence of that comment.

EE-WP17 A Deployment Tool must report an error when processing an Jakarta Enterprise Beans deployment descriptor that includes an Jakarta Enterprise Beans QL expression that does not conform to the Specifications.

EE-WP18 The Runtime must report an error when processing a Configuration Descriptor that does not conform to the Specifications.

EE-WP19 The presence of an XML comment in a Configuration Descriptor, when processed by the Runtime, must not cause the functional programmatic behavior of the Runtime to vary from the functional programmatic behavior of the Runtime in the absence of that comment.

EE-WP20 Compatibility testing for the Jakarta EE 10 Web Profile consists of running the tests for the technologies defined in Section 1.2.2, "Jakarta EE 10 Web Profile Technologies Tested With Jakarta EE 10 Platform TCK." If optional technologies defined in the Jakarta EE 10 Web Profile platform are implemented in addition to the required Jakarta EE 10 Web Profile technologies, corresponding tests within this TCK for those additional technologies must also be run.

EE-WP21 Compliance testing for Jakarta EE 10 Web Profile consists of running the Jakarta EE 10 Web Profile tests and the following Technology Compatibility Kits (TCKs):

- Jakarta Authentication 3.0
- Jakarta Bean Validation 3.0
- Jakarta Concurrency 3.0
- Jakarta Contexts and Dependency Injection 4.0
- Jakarta Dependency Injection 2.0
- Jakarta Faces 4.0
- Jakarta JSON Processing 2.1
- Jakarta JSON Binding 3.0
- Jakarta RESTful Web Services 3.1
- Jakarta Security 3.0
- Jakarta XML Binding 4.0 (If XML Binding is supported)

In addition to the compatibility rules outlined in this TCK User's Guide, Jakarta EE 10 implementations must also adhere to all of the compatibility rules defined in the User's Guides of the aforementioned TCKs.

EE-WP21.1 If the Jakarta EE 10 Web Profile implementation uses a runtime which has already been validated by the Technology Compatibility Kit, the Jakarta EE 10 Web Profile implementation may use result of such validation to claim its compliance with the Technology Compatibility Kit.

EE-WP22 Source Code in WSDL-to-Java Output when compiled by a Reference Compiler must execute properly when run on a Reference Runtime.

EE-WP23 Source Code in WSDL-to-Java Output must be in source file format defined by the Java Language Specification (JLS).

EE-WP24 Java-to-WSDL Output must fully meet W3C requirements for the Web Services Description Language (WSDL) 1.1.

EE-WP25 A Java-to-WSDL Tool must not produce Java-to-WSDL Output from source code that does not

conform to the Java Language Specification (JLS).

3.3 Jakarta Platform, Enterprise Edition 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 Jakarta EE 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**.

3.3.1 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.

3.3.2 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 {TechnologyShortName} 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 {TechnologyShortName} TCK.

3.3.3 TCK Test Appeals Steps

1. Challenges should be filed via the Jakarta EE Platform 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.

3.4 Specifications for Jakarta Platform, Enterprise Edition Version 10, Web Profile

The Specifications for Jakarta Platform, Enterprise Edition 10, Web Profile are found on the Eclipse Foundation, Jakarta EE Specification web site at `https://jakarta.ee/specifications/webprofile/10/`. You may also find information available from the EE4J Jakarta EE Platform project page, at `https://projects.eclipse.org/projects/ee4j.jakartaee-platform`.

[[3libraries-for-jakarta-platform-enterprise-edition-version-10-web-profile]]

3.5 Libraries for Jakarta Platform, Enterprise Edition Version 10, Web Profile

The following location provides a list of packages that constitute the required class libraries for the full Java EE 10 platform:

```https://projects.eclipse.org/projects/ee4j.jakartaee-platform`

The following list constitutes the subset of Jakarta EE 10 packages that are required for the Jakarta EE 10 Web Profile:

- jakarta.annotation
- jakarta.annotation.security
- jakarta.annotation.sql
- jakarta.decorator
- jakarta.ejb
- jakarta.ejb.embeddable
- jakarta.ejb.spi
- jakarta.el
- jakarta.enterprise.context
- jakarta.enterprise.context.spi

- jakarta.enterprise.event
- jakarta.enterprise.inject
- jakarta.enterprise.inject.spi
- jakarta.enterprise.util
- jakarta.faces
- jakarta.faces.application
- jakarta.faces.bean
- jakarta.faces.component
- jakarta.faces.component.behavior
- jakarta.faces.component.html
- jakarta.faces.component.visit
- jakarta.faces.context
- jakarta.faces.convert
- jakarta.faces.el
- jakarta.faces.event
- jakarta.faces.flow
- jakarta.faces.flow.builder
- jakarta.faces.lifecycle
- jakarta.faces.model
- jakarta.faces.render
- jakarta.faces.validator
- jakarta.faces.view
- jakarta.faces.view.facelets
- jakarta.faces.webapp
- jakarta.inject
- jakarta.interceptor
- jakarta.json
- jakarta.json.spi
- jakarta.json.stream
- jakarta.persistence
- jakarta.persistence.criteria
- jakarta.persistence.metamodel

- jakarta.persistence.spi
- jakarta.servlet
- jakarta.servlet.annotation
- jakarta.servlet.descriptor
- jakarta.servlet.http
- jakarta.servlet.jsp
- jakarta.servlet.jsp.el
- jakarta.servlet.jsp.jstl.core
- jakarta.servlet.jsp.jstl.fmt
- jakarta.servlet.jsp.jstl.sql
- jakarta.servlet.jsp.jstl.tlv
- jakarta.servlet.jsp.tagext
- jakarta.transaction
- javax.transaction.xa
- jakarta.validation
- jakarta.validation.bootstrap
- jakarta.validation.constraints
- jakarta.validation.constraintvalidation
- jakarta.validation.executable
- jakarta.validation.groups
- jakarta.validation.metadata
- jakarta.validation.spi
- jakarta.websocket
- jakarta.websocket.server
- jakarta.ws.rs
- jakarta.ws.rs.client
- jakarta.ws.rs.container
- jakarta.ws.rs.core
- jakarta.ws.rs.ext
- jakarta.json.bind
- jakarta.json.bind.adapter
- jakarta.json.bind.annotation

- jakarta.json.bind.config
- jakarta.json.bind.serializer
- jakarta.json.bind.spi
- jakarta.security.enterprise
- jakarta.security.enterprise.authentication.mechanism.http
- jakarta.security.enterprise.credential
- jakarta.security.enterprise.identitystore
- jakarta.xml.bind (optional)
- jakarta.xml.bind.annotation (optional)
- jakarta.xml.bind.annotation.adapters (optional)
- jakarta.xml.bind.attachment (optional)
- jakarta.xml.bind.util (optional)

4 Installation

This chapter explains how to install the Jakarta EE 10 Platform TCK software and perform a sample test run to verify your installation and familiarize yourself with the TCK. Installation instructions are provided for Eclipse GlassFish 6.1, a Compatible Implementation (CI) of Jakarta EE. If you are using another compatible implementation, refer to instructions provided with that implementation.

After installing the software according to the instructions in this chapter, proceed to Chapter 5, "Setup and Configuration," for instructions on configuring your test environment.

This chapter covers the following topics:

- Installing the Jakarta EE 10 Compatible Implementation
- Installing the Jakarta EE 10 Platform TCK
- Verifying Your Installation (Optional)

4.1 Installing the Jakarta EE 10 Compatible Implementation

Before You Begin

If a Jakarta EE 10 Compatible Implementation (CI) is already installed, it is recommended that you shut it down and start with a new, clean CI installation.

1. Install the Java SE 8 JDK bundle, if it is not already installed.
   Refer to the JDK installation instructions for details. The JDK bundle can be downloaded from http://www.oracle.com/technetwork/java/javase/downloads/index.html

2. Create or change to the directory in which you will install the Jakarta EE 10 CI.

3. Copy or download the Jakarta EE 10 CI, for example, Eclipse GlassFish 6.1.

4. Unzip the Jakarta EE 10 CI bundle.

5. For Eclipse GlassFish 6.1, set the following environment variables:

   - `JAVAEE_HOME` to the CI directory you just created

   - `JAVA_HOME` to the JDK you want to use

6. Start the Jakarta EE 10 CI, Eclipse GlassFish 6.1, by executing the following command:

```
<JAVAEE_HOME>/bin/asadmin start-domain
```

4.2 Installing the Jakarta EE 10 Platform TCK

Complete the following procedure to install the Jakarta EE 10 Platform TCK on a system running the Solaris, Linux, or Windows operating system.

> ℹ️ When installing in the Windows environment, the Jakarta EE 10 CI, Java SE 8 JDK, and TCK should be installed on the same drive. If you must install these components on different drives, be sure to change the `ri.applicationRoot` and `s1as.applicationRoot` properties as needed in the `<TS_HOME>/bin/ts.jte` TCK configuration file. See Section 5.4.2, "Windows-Specific Properties," for more information.

1. Copy or download the TCK 10 software.

2. Change to the directory in which you want to install the Jakarta EE 10 TCK software and use the `unzip` command to extract the bundle:

    ```
    cd install_directory
    unzip jakartaeetck-nnn.zip
    ```

    This creates the `jakartaeetck` directory. The install_directory`/jakartaeetck` directory will be `TS_HOME`.

3. Set the `TS_HOME` environment variable to point to the `javaeetck` directory.

After you complete the installation, follow the directions in Chapter 5, "Setup and Configuration," to set up and configure the Jakarta EE 10 Platform TCK test suite.

4.3 Verifying Your Installation (Optional)

This procedure is optional. After installing the Jakarta EE 10 CI and Jakarta EE 10 TCK, you may want to verify your installation by running the TCK samples against the Jakarta EE 10 CI. See Chapter 5, "Setup and Configuration," for complete configuration instructions.

1. For the Jakarta EE 10 CI, Eclipse GlassFish 6.1, set the following properties in your `<TS_HOME>/bin/ts.jte` file:

```
javaee.home=<JAVAEE_HOME>
javaee.home.ri=<JAVAEE_HOME>
mailHost=<mail-server-host> // the name of an accessible SMTP server
mailuser1=firstname.lastname@xyz.com // a valid email address
javamail.password=password // password for mailuser1
orb.host=localhost // the name of the machine running the Jakarta EE 10 CI
orb.host.ri=localhost // the name of the machine running the Jakarta EE 10 CI
```

2. Use these commands to run the Jakarta EE 10 Platform TCK sample tests:

```
cd <TS_HOME>/bin
ant start.javadb.asadmin
ant config.vi
cd <TS_HOME>/src/com/sun/ts/tests/samples
ant runclient
```

5 Setup and Configuration

This chapter describes how to set up the Jakarta EE 10 Platform TCK test suite and configure it to work with your test environment. It is recommended that you first set up the testing environment using the Jakarta EE 10 CI and then with your Jakarta EE 10 server.

This chapter includes the following topics:

- Allowed Modifications
- Configuring the Test Environment
- Configuring a Jakarta EE 10 Server
- Modifying Environment Settings for Specific Technology Tests
- Using the JavaTest Harness Configuration GUI

5.1 Allowed Modifications

You can modify the following test suite components only:

- Your implementation of the porting package
- `ts.jte` environment file
- The vendor-specific SQL files in `<TS_HOME>/sql`
- Any files in `<TS_HOME>/bin` and `<TS_HOME>/bin/xml` (except for `ts.*` files)

5.2 Configuring the Test Environment

The instructions in this section and in Section 5.3.3, "Configuring Your Application Server as the VI," step you through the configuration process for the Solaris, Microsoft Windows, and Linux platforms.

All TCK test configuration procedures are based on running the Ant scripts against a set of build targets. The primary location of any configuration settings you are likely to make is the `<TS_HOME>/bin/ts.jte` environment file. You may also want to modify the `javaee_vi.xml` and `initdb.xml` Ant configuration files and the vendor-specific SQL files. These two files contain predefined Ant targets that are implemented such that they automatically configure the Jakarta EE 10 CI and its associated database in order to pass the TCK. A Implementer may choose to implement these targets to work with their server environment to perform the steps described in Section 5.3.3, "Configuring Your Application Server as the VI."

Ant targets are available at all levels of the TCK source tree that allow you to execute a wide variety of test scenarios and configurations.

Before You Begin

In these instructions, variables in angle brackets need to be expanded for each platform. For example, `<TS_HOME>` becomes `$TS_HOME` on Solaris/Linux and `%TS_HOME%` on Windows. In addition, the forward slashes (`/`) used in all of the examples need to be replaced with backslashes (`\`) for Windows.

1. Identify the software pieces and assemble them into the Jakarta EE 10 platform to be tested for certification.

2. Implement the porting package APIs.
   Some functionality in the Jakarta EE 10 platform is not completely specified by an API. To handle this situation, the Jakarta EE 10 Platform TCK test suite defines a set of interfaces in the `com.sun.cts.porting` package, which serve to abstract any implementation-specific code. You must create your own implementations of the porting package interfaces to work with your particular Jakarta EE 10 server environment. See Section 11.2, "Porting Package APIs," for additional information about the porting APIs. API documentation for the porting package interfaces is available in the `<TS_HOME>/docs/api` directory.

3. Set up the Jakarta Platform, Enterprise Edition Compatible Implementation (CI) server.
   See Section 5.3.2, "Configuring the Jakarta EE 10 CI as the VI," for a list of the modifications that must be made to run CTS against the Jakarta EE 10 CI.

4. Set up the vendor's Jakarta EE 10 server implementation (VI).
   See Section 5.3.3, "Configuring Your Application Server as the VI," for a list of the modifications that must be made to run CTS against the vendor's Jakarta EE 10 server.

5. Validate your configuration.
   Run the sample tests provided. If the tests pass, your basic configuration is valid. See Section 7.3, "Validating Your Test Configuration," for information about using JavaTest to run the sample tests.

6. Run the TCK tests.
   See Chapter 7, "Executing Tests," for information about Starting JavaTest and running tests.

5.3 Configuring a Jakarta EE 10 Server

This section describes how to configure the Jakarta EE 10 server under test. You can run the TCK tests against the Jakarta EE 10 CI or your own Jakarta Platform, Enterprise Edition server. When performing interoperability (interop) tests or web service-based tests, you will be running two Jakarta EE 10 CI servers, one of which must be a Jakarta EE 10 CI using, or configured to use a database. For example, Eclipse GlassFish 6.1 is bundled and configured to use the Apache Derby database.

For the purposes of this section, it is useful to clarify three terms as they are used here:

- Compatible Implementation (CI): Jakarta EE 10 CI, for example, Eclipse GlassFish 6.1

- Vendor Implemention (VI): Jakarta EE 10 implementation from a vendor other than Eclipse; typically, the goal of running the TCK is to certify a Jakarta EE 10 VI; in some cases, for purposes of familiarizing yourself with TCK, you may choose to run the Jakarta EE 10 CI as the VI

- Bundled Derby: Apache Derby database bundled with the Jakarta EE 10 CI, Eclipse GlassFish 6.1

5.3.1 Jakarta Platform, Enterprise Edition Server Configuration Scenarios

There are three general scenarios for configuring Jakarta EE 10 servers for Jakarta EE 10 Platform TCK testing (Note: in the following images, Java EE refers to Jakarta EE. RI should be replaced with CI for Compatible Implementation):

- Configure the Jakarta EE 10 CI as the server under test

Use the Jakarta EE 10 CI as the Jakarta EE 10 VI; you may want to do this as a sanity check to make sure you are comfortable with using the Jakarta EE 10 TCK against a known standard CI with certified sample applications before proceeding with tests against your Jakarta EE 10 VI. See Section 5.3.2, "Configuring the Jakarta EE 10 CI as the VI," for instructions.

- Configure your Jakarta EE 10 VI as Server Under Test

This is the primary goal of using the Jakarta EE 10 Platform TCK; you will eventually need to configure the Jakarta EE 10 implementation you want to certify. See Section 5.3.3, "Configuring Your Application Server as the VI," for instructions.

- Configure two Jakarta EE 10 servers for the purpose of interop testing
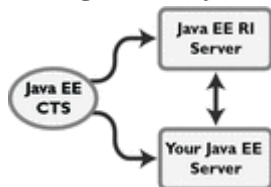
Rebuildable tests require that you configure two Jakarta EE 10 servers on one or two machines. One server will be your Jakarta EE 10 VI running a database of your choice with JDBC 4.1-compliant drivers. The second server must be the Jakarta EE 10 CI using the bundled Java DB database.

In terms of the Jakarta EE 10 Platform TCK, all TCK configuration settings are made in the `<TS_HOME>/bin/ts.jte` file. When configuring a Jakarta EE 10 server, the important thing is to make sure that the settings you use for your server match those in the `ts.jte` file.

These configuration scenarios are described in the following sections.

5.3.2 Configuring the Jakarta EE 10 CI as the VI

To configure the Jakarta EE 10 CI as the server under test (that is, to use the Jakarta EE 10 CI as the VI) follow the steps listed below. In this scenario, the goal is simply to test the Jakarta EE 10 CI against the CTS for the purposes of familiarizing yourself with TCK test procedures. You may also want to refer to the Quick Start guides included with the Jakarta EE 10 TCK for similar instructions.

1. Set server properties in your `<TS_HOME>/bin/ts.jte` file to suit your test environment.
   Be sure to set the following properties:

   a. Set the `webServerHost` property to the name of the host on which your Web server is running that is configured with the CI.
      The default setting is `localhost`.

   b. Set the `webServerPort` property to the port number of the host on which the Web server is running and configured with the CI.
      The default setting is `8001`.

   c. Set the `wsgen.ant.classname` property to the Vendor's implementation class that mimics the CI Ant task that in turn calls the `wsgen` Java-to-WSDL tool.
      The default setting is `com.sun.tools.ws.ant.WsGen`.

   d. Set the `wsimport.ant.classname` property to the Vendor's implementation class that mimics the CI Ant task that in turn calls the `wsimport` WSDL-to-Java tool.
      The default setting is `com.sun.tools.ws.ant.WsImport`.

   e. Set the `porting.ts.url.class` property to your porting implementation class that is used for obtaining URLs.
      The default setting for the CI porting implementation is `com.sun.ts.lib.implementation.sun.common.SunRIURL`.

   f. Set the database-related properties in the `<TS_HOME>/bin/ts.jte` file.
      Section C.3, "Database Properties in ts.jte," lists the names and descriptions for the database properties you need to set.

   g. Add the following JVM option to the `command.testExecuteAppClient` property to enable the Security Manager in the application client container:

   ```
   -Djava.security.manager
   ```

2. Install the Jakarta EE 10 CI and configure basic settings, as described in Chapter 4, "Installation."

3. Start the Jakarta EE 10 CI application server.
   Refer to the application server documentation for complete instructions.

4. Enable the Security Manager.

If you are using the Jakarta EE 10 CI, execute the following command from the command line:

```
asadmin create-jvm-options -Djava.security.manager
```

5. Stop and restart your application server so it is running with the Security Manager enabled.

6. Change to the `<TS_HOME>/bin` directory.

7. Start your backend database.
   If you are using Derby as your backend database, execute the `start.javadb` Ant target:

```
ant -f xml/impl/glassfish/s1as.xml start.javadb
```

Otherwise, refer to your backend database administration documentation for information about starting your database server.

8. Initialize your backend database.
   If you are using Derby as your backend database, execute the `init.derby` Ant target:

```
ant -f xml/init.xml init.derby
```

If you are not using Derby as your backend database, refer to Appendix C, "Configuring Your Backend Database."

> ℹ️ If you are using MySQL or MS SQL Server as your backend database, see Section 5.4.24, "Backend Database Setup," for additional database setup instructions.

9. Run the configuration Ant target.

```
ant config.vi
```

> ℹ️ By default, the `config.vi` Ant task configures the entire application server. Sometimes you may not want or need to configure everything, such as connector RAR files. If you are not performing connector-related tests, you can avoid the deployment and configuration of RAR files by using the Ant option `-Dskip.config.connector=true`. This will reduce configuration times, the deployment of unneeded RAR files, and the creation of unnecessary resources on the server under test. For example, the following command will do this.
>
> ```
> ant -Dskip.config.connector=true config.vi
> ```

10. Build the special web services clients.

    The special webservices tests under the `webservices12/specialcases` directory have prebuilt endpoints, but the clients are not prebuilt. The clients will be built after the endpoints are first predeployed to the application server under test.

    During the build, the clients import the WSDLs (by means of the Jakarta EE `wsimport` and `wsgen` tools) from the predeployed webservices endpoints. This process verifies that importing a WSDL from a predeployed webservice endpoint works properly.

    To build the special webservices clients, the following command must be executed:

    ```
    ant build.special.webservices.clients
    ```

    This predeploys all the special webservices endpoints, builds all the special webservices clients, and then undeploys the special webservices endpoints. See Section 11.2.2, "Ant-Based Deployment Interface," for more information about the Ant-based deployment interface, including guidelines for creating your own Ant-based deployment implementation.

11. Continue on to Chapter 7, "Executing Tests," for instructions on running tests.

5.3.3 Configuring Your Application Server as the VI

To use a Jakarta EE 10 server other than the Jakarta EE 10 CI, follow the steps below.

1. Set server properties in your `<TS_HOME>/bin/ts.jte` file to suit your test environment.
   Be sure to set the following properties:

   a. Set the `webServerHost` property to the name of the host on which your Web server is running that is configured with the CI.
      The default setting is `localhost`.

   b. Set the `webServerPort` property to the port number of the host on which the Web server is running and configured with the CI.
      The default setting is `8001`.

   c. Set the `wsgen.ant.classname` property to the Vendor's implementation class that mimics the CI Ant task that in turn calls the `wsgen` Java-to-WSDL tool.
      The default setting is `com.sun.tools.ws.ant.WsGen`.

   d. Set the `wsimport.ant.classname` property to the Vendor's implementation class that mimics the CI Ant task that in turn calls the `wsimport` WSDL-to-Java tool.
      The default setting is `com.sun.tools.ws.ant.WsImport`.

   e. Set the `porting.ts.url.class` property to your porting implementation class that is used for obtaining URLs.

   f. Set the database-related properties in the `<TS_HOME>/bin/ts.jte` file.
      Section C.3, "Database Properties in ts.jte," lists the names and descriptions for the database

properties you need to set.

g. Add the following JVM option to the `command.testExecuteAppClient` property to enable the Security Manager in the application client container:

```
-Djava.security.manager
```

Add this option to the list of other -D JVM options for this property.

These settings can vary, but must match whatever you used when setting up your Jakarta Platform, Enterprise Edition server.

2. Install the Jakarta Platform, Enterprise Edition VI and configure basic settings.
   If you want to configure your Jakarta Platform, Enterprise Edition server using Ant configuration target similar to the target for the Jakarta EE 10 CI, as described in Chapter 4, "Installation," you will need to modify the `<TS_HOME>/bin/xml/javaee_vi.xml` file to implement the defined Ant targets for your application server. Then run:

```
ant config.vi
```

The Ant configuration targets you implement, if any, may vary. Whichever configuration method you choose, make sure that all configuration steps in this procedure are completed as shown.

3. Enable the Security Manager and then stop and restart your application server so it is running with the Security Manager enabled.

4. Provide alternate endpoint and WSDL URLs, if necessary.
   The `<TS_HOME>/bin` directory contains the following `.dat` files:

   - `jaxws-url-props.dat`

   - `jws-url-props.dat`

   - `webservices12-url-props.dat`

   These files contain the webservice endpoint and WSDL URLs that the CTS tests use when running against the CI. In the porting package used by the TCK, the URLs are returned as is since this is the form that the CI expects. You may need an alternate form of these URLs to run the CTS tests in your environment. However, you MUST NOT modify the existing `.dat` files, but instead make any necessary changes in your own porting implementation class to transform the URLs appropriately for your environment.

5. Install and configure a database for the server under test.

6. Start your database.

7. Initialize your database for TCK tests.

   a. If you choose to not implement the `javaee_vi.xml` targets, execute the following command to specify the appropriate DML file:

(Derby DB Example)

```
ant -Dtarget.dml.file=tssql.stmt \
    -Ddml.file=javadb/javadb.dml.sql copy.dml.file
```

b. Execute the following command to initialize your particular database:

```
ant -f <TS_HOME>/bin/xml/initdb.xml init.Database
```

For example, to initialize a Derby DB database:

```
ant -f <TS_HOME>/bin/xml/initdb.xml init.javadb
```

Refer to Appendix C, "Configuring Your Backend Database," for detailed database configuration and initialization instructions and a list of database-specific initialization targets.

8. Start your Jakarta EE 10 server.

9. Set up required users and passwords.

a. Set up database users and passwords that are used for JDBC connections.
   The Jakarta EE 10 Platform TCK requires several user names, passwords, and user-to-role mappings. These need to match those set in your `ts.jte` file. By default, `user1`, `user2`, `user3`, `password1`, `password2`, and `password3` are set to `cts1`.

b. Set up users and passwords for your Jakarta Platform, Enterprise Edition server.
   For the purpose of running the TCK test suite, these should be set as follows:

User	Password	Groups
j2ee_vi	j2ee_vi	staff
javajoe	javajoe	guest
j2ee	j2ee	staff, mgr, asadmin
j2ee_ri	j2ee_ri	staff

Note that adding the `asadmin` group is only necessary when running against the Eclipse GlassFish 6.1 Jakarta EE 10 CI application server. It is required in this case because the management Jakarta Enterprise Bean (MEjb) in the Jakarta EE 10 server is protected with the `asadmin` group. Other appservers may or may not choose to protect their MEjb. If necessary for your appserver implementation, you should also add the group name with which your MEjb is protected.
Also make sure the principal to role-mappings that are specified in the runtime XML files are properly mapped in your environment (TODO: confirm that we can also delete mention of principal (user) ⇒ role-mappings). Note that the principal-to-role mappings may vary for each

application.

10. Make sure that the appropriate JDBC 4.1-compliant database driver class, any associated database driver native libraries, and the correct database driver URL are available.

11. Configure your Jakarta Platform, Enterprise Edition server to use the appropriate JDBC logical name (`jdbc/DB1`) when accessing your database server.

12. Configure your Jakarta EE 10 server to use the appropriate logical name (`jdbc/DBTimer`) when accessing your Jakarta Enterprise Beans timer.

13. Provide access to a JNDI lookup service.

14. Provide access to a Web server.

15. Provide access to a Jakarta Mail server that supports the SMTP protocol.

16. Execute the `add.interop.certs` Ant target.

    ℹ️ This step installs server side certificates for interoperability testing; that is, it installs the CI's server certificate to VI and VI's server certificate into the CI. This step is necessary for mutual authentication tests in which both the server and client authenticate to each other.

17. Install the client-side certificate in the `trustStore` on the Jakarta EE 10 server. Certificates are located `<TS_HOME>/bin/certificates`. Use the certificate that suits your environment.

    a. `cts_cert`: For importing the TCK client certificate into a `truststore`

    b. `clientcert.jks`: Used by the Java SE 8 runtime to identify the CTS client's identity

    c. `clientcert.p12`: Contains TCK client certificate in `pkcs12` format

    d. Append the file `<TS_HOME>/bin/server_policy.append` to the Java policy file or files on your Jakarta EE 10 server.
    This file contains the grant statements used by the test harness, signature tests, and API tests.

    e. Append the file `<TS_HOME>/bin/client_policy.append` to the application client's Java policy file, which is referenced in the `TestExecuteAppClient` section of the `ts.jte` file.

    f. Make the appropriate transaction interoperability setting on the Jakarta EE 10 server and the server that is running the Jakarta EE 10 CI.

    g. If necessary, refer to the sections later in this chapter for additional configuration information you may require for your particular test goals.

    h. Restart your Jakarta EE 10 server.

    i. Build the special Web services clients.
    This step may be bypassed at this time if you are not going to immediately run the tests under `<TS_HOME>/src/com/sun/ts/tests/webservices12`. However, you must return to this configuration section and complete it in order to run these tests.
    The special Web services tests under the `webservices12/specialcases` directory have prebuilt endpoints, but the clients are not prebuilt. The clients will be built after the endpoints are first

predeployed to the application server under test.

During the build the clients import the WSDLs (by means of the Jakarta EE `wsimport` and `wsgen` tools) from the predeployed Web services endpoints. This process verifies that importing a WSDL from a predeployed Web service endpoint works properly.

j. Install the Jakarta EE 10 CI.

k. Set the following properties in your `<TS_HOME>/bin/ts.jte` file.

The current values should be saved since they will be needed later in this step.

- Set the `javaee.home.ri` property to the location where the Jakarta EE 10 CI is installed.

- Set the `wsgen.ant.classname` property to the Jakarta EE 10 application server Ant task that in turn calls the `wsimport` Java-to-WSDL tool. It must be set to:

```
com.sun.tools.ws.ant.WsGen
```

- Set the `wsgen.classpath` property to:

```
${javaee.classes.ri}:${tools.jar}
```

- Set the `wsimport.ant.classname` property to the Jakarta EE 10 application server Ant task that in turn calls the `wsimport` WSDL-to-Java tool.
It must be set to `com.sun.tools.ws.ant.WsImport`

- Set the `wsimport.classpath` property to the following value:

```
${javaee.classes.ri}:${tools.jar}
```

l. Build the special Web services clients by executing the following command:

```
ant build.special.webservices.clients
```

This predeploys all the special Web services endpoints, builds all the special webservices clients, and then undeploys the special webservices endpoints. See Section 11.2.2, "Ant-Based Deployment Interface," for more information about the Ant-based deployment interface, including guidelines for creating your own Ant-based deployment implementation.

m. Once this command completes successfully, the following `ts.jte` properties must be set back to their previous values:

- `wsgen.ant.classname`

- `wsgen.classpath`

- `wsimport.ant.classname`

- `wsimport.classpath`

n. The following `webservices12-url-props.dat` properties must be set back to their original values:

- `specialcases.defaultserviceref.wsdlloc`

- `specialcases.nameattrserviceref.wsdlloc`

- `specialcases.providerserviceref.wsdlloc`

18. Continue on to .

5.4 Modifying Environment Settings for Specific Technology Tests

Before you can run any of the technology-specific Jakarta EE 10 Platform TCK tests, you must supply certain information that JavaTest needs to run the tests in your particular environment. This information exists in the `<TS_HOME>/bin/ts.jte` environment file. This file contains sets of name/value pairs that are used by the tests. You need to assign a valid value for your environment for all of the properties listed in the sections that follow.

> This section only discusses a small subset of the properties you can modify. Refer to the `ts.jte` file for information about the many other properties you may want to modify for your particular test environment.

This section includes the following topics:

- Test Harness Setup

- Windows-Specific Properties

- Test Execution Command Setup

- Jakarta Servlet Test Setup

- Jakarta WebSocket Test Setup

- JDBC Test Setup

- Jakarta Mail Test Setup

- Jakarta XML Registry Test Setup

- Jakarta RESTful Web Services Test Setup

- Jakarta Connector Test Setup

- XA Test Setup

- Jakarta Enterprise Beans 4.0 Test Setup

- Jakarta Enterprise Beans Timer Test Setup

- Entity Bean Container-Managed Persistence Test Setup for Jakarta Enterprise Beans V 1.1

- Jakarta Persistence API Test Setup

- Jakarta Messaging Test Setup

- Jakarta Authentication Test Setup

- Jakarta Authorization Test Setup

- WSDL: Webservice Test and Runtime Notes

- Jakarta Security API Test Setup

- Signature Test Setup

- Backend Database Setup

5.4.1 Test Harness Setup

Verify that the following properties, which are used by the test harness, have been set in the `<TS_HOME>/bin/ts.jte` file:

```
harness.temp.directory=<TS_HOME>/tmp
harness.log.port=2000
harness.log.traceflag=[true | false]
deployment_host.1=<hostname>
deployment_host.2=<hostname>
porting.ts.deploy2.class.1=<vendor-deployment-class>
porting.ts.login.class.1=<vendor-login-class>
porting.ts.url.class.1=<vendor-url-class>
porting.ts.jms.class.1=<vendor-jms-class>
porting.ts.tsHttpsURLConnection.class.1=<vendor-HttpsURLConnection-class>
```

- The `harness.temp.directory` property specifies a temporary directory that the harness creates and to which the TCK harness and tests write temporary files. The default setting should not need to be changed.

- The `harness.log.port` property specifies the port that server components of the tests use to send logging output back to JavaTest. If the default port is not available on the machine running JavaTest, you must edit this property and set it to an available port. The default setting is `2000`.

- The `harness.log.traceflag` property is used to turn on or turn off verbose debugging output for the tests. The value of the property is set to `false` by default. Set the property to `true` to turn debugging on.

- The `deployment_host.1` and `deployment_host.2` properties specify the systems where the vendor's Jakarta Platform, Enterprise Edition server and the Jakarta Platform, Enterprise Edition CI server are running. By default, JavaTest will use the `orb.host` and `orb.host.ri` systems, which are set in the

`ts.jte` file.

- The porting class `.1` and `.2` property sets specify the class names of porting class implementations. By default, both property sets point to the Jakarta Platform, Enterprise Edition CI-specific classes. To run the interoperability tests, do not modify the `.2` set. These properties should always point to the Jakarta Platform, Enterprise Edition CI classes. Modify the `.1` set to point to implementations that work in your specific Jakarta Platform, Enterprise Edition environment.

- The `-Dcts.tmp` option for the `testExecute` and `testExecuteAppClient` commands in the `ts.jte` file have been set. This Java option tells the test suite the location to which the test suite will write temporary files.

5.4.2 Windows-Specific Properties

When configuring the Jakarta EE 10 Platform TCK for the Windows environment, set the following properties in `<TS_HOME>/bin/ts.jte`:

- `pathsep` to semicolon (pathsep=`;`)

- `s1as.applicationRoot` to the drive on which you have installed CTS (for example, `s1as.applicationRoot=C:`)
  When installing in the Windows environment, the Jakarta Platform, Enterprise Edition CI, JDK, and TCK should all be installed on the same drive. If you must install these components on different drives, also change the `ri.applicationRoot` property in addition to the `pathsep` and `s1as.applicationRoot` properties; for example:

```
ri.applicationRoot=C:
```

> ℹ️ When configuring the CI and TCK for the Windows environment, never specify drive letters in any path properties in `ts.jte`.

5.4.3 Test Execution Command Setup

The test execution command properties are used by the test harness. By default, the `ts.jte` file defines a single command line for each of the commands that is used for both UNIX and Windows environments.

- `command.testExecute`

- `command.testExecuteAppClient`

- `command.testExecuteAppClient2`

If these commands do not meet your needs, you can define separate entries for the UNIX and Windows

environments. Edit either the `ts_unix` or `ts_win32` test execution properties in the `ts.jte` file. For UNIX, these properties are:

- `env.ts_unix.command.testExecute`
- `env.ts_unix.command.testExecuteAppClient`
- `env.ts_unix.command.testExecuteAppClient2`

For Windows, these properties are:

- `env.ts_win32.command.testExecute`
- `env.ts_win32.command.testExecuteAppClient`
- `env.ts_win32.command.testExecuteAppClient2`

The `testExecute` property specifies the Java command that is used to execute individual tests from a standalone URL client. Tests in which the client directly invokes a web component (Jakarta Servlet or Jakarta Server Pages), use this command line since there is no application client container involved.

> The default settings are specific to the Jakarta Platform, Enterprise Edition CI. If you are not using the Jakarta Platform, Enterprise Edition CI, adjust these properties accordingly.

5.4.4 Jakarta Servlet Test Setup

Make sure that the following servlet properties have been set in the `ts.jte` file:

```
ServletClientThreads=[2X size of default servlet instance pool]
servlet_waittime=[number_of_milliseconds]
servlet_async_wait=[number_of_seconds]
logical.hostname.servlet=server
s1as.java.endorsed.dirs=${endorsed.dirs}${pathsep}${ts.home}/endorsedlib
```

The `servlet_waittime` property specifies the amount of time, in milliseconds, to wait between the time when the `HttpSession` is set to expire on the server and when the `HttpSession` actually expires on the client. This time is configurable to allow the servlet container enough time to completely invalidate the `HttpSession`. The default value is 10 milliseconds.

The test `serverpush` in Jakarta Servlet 6.0, uses `httpclient`, a new library in JDK9 that depends on `java.util.concurrent.flow` (also new class in JDK9).

The `servlet_async_wait` property sets the duration of time in seconds to wait between sending asynchronous messages. This property is used in place to test non-interrupted IO, where two messages are sent in two different batches and the receiving end will be read in a different read cycle. This

property sets the time to wait in seconds on the sending side. The default is 4 seconds.

The `logical.hostname.servlet` property identifies the configuration name of the logical host on which the `ServletContext` is deployed. This used to identify the name of a logical host that processes Jakarta EE 10 requests. Jakarta EE 10 requests may be directed to a logical host using various physical or virtual host names or addresses, and a message processing runtime may be composed of multiple logical hosts. The `logical.hostname.servlet` property is required to properly identify the Jakarta EE 10 profile's `AppContextId` hostname. This property is used by the Jakarta EE 10 security tests as well as by the `ServletContext.getVirtualServerName()` method. If a `logical.hostname.servlet` does not exist, set this property to the default hostname (for example, `webServerHost`). The default is "server".

5.4.5 Jakarta WebSocket Test Setup

Make sure that the following WebSocket property has been set in the `ts.jte` file:

```
ws_wait=[number_of_seconds]
```

The `ws_wait` property configures the wait time, in seconds, for the socket to send or receive a message. A multiple of 5 of this time is also used to test socket timeouts.

The Jakarta WebSocket tests also use the following properties: `webServerHost` and `webServerPort`. See Section 5.3.2, "Configuring the Jakarta EE 10 CI as the VI," for more information about setting these properties.

> **ⓘ** The SSL related tests under `/ts/javaeetck/src/com/sun/ts/tests/websocket/platform/jakarta/websocket/server/handshakerequest/authenticatedssl/` use self signed certificate bundled with the TCK bundle. These certificates are generated with localhost as the hostname and would work only when orb.host value is set to localhost in ts.jte. If the server's hostname is used instead of the localhost, the tests in this suite might fail with the below exception - jakarta.websocket.DeploymentException: SSL handshake has failed.

5.4.6 JDBC Test Setup

The JDBC tests require you to set the timezone by modifying the `tz` property in the `ts.jte` file. On Solaris systems, you can check the timezone setting by looking in the file `/etc/default/init`. Valid values for the `tz` property are in the directory `/usr/share/lib/zoneinfo`. The default setting is `US/Eastern`. This setting is in `/usr/share/lib/zoneinfo/US`.

The `tz` property is only used for Solaris configurations; it does not apply to Windows XP/2000.

5.4.7 Jakarta Mail Test Setup

Complete the following tasks before you run the Jakarta Mail tests:

1. Set the following properties in the `ts.jte` file:

    ```
    mailuser1=[user@domain]
    mailFrom=[user@domain]
    mailHost=mailserver
    javamail.password=password
    ```

    ◦ Set the `mailuser1` property to a valid mail address. Mail messages generated by the Jakarta Mail tests are sent to the specified address. This user must be created in the IMAP server.

    ◦ Set the `mailFrom` property to a mail address from which mail messages that the Jakarta Mail tests generate will be sent.

    ◦ Set the `mailHost` property to the address of a valid mail server where the mail will be sent.

    ◦ Set the `javamail.password` property to the password for `mailuser1`.

2. Populate your IMAP server with sample messages.
    Change to the `<TS_HOME>/bin` directory and execute the Ant target `populateMailbox` to create the sample messages in your IMAP server.

    ```
    cd <TS_HOME>/bin
    ant populateMailbox
    ```

5.4.8 Jakarta RESTful Web Services Test Setup

This section explains how to set up the test environment to run the Jakarta RESTful Web Services tests using the Jakarta EE 10 Compatible Implementation and/or a Vendor Implementation. This setup also includes steps for packaging/repackaging and publishing the packaged/repackaged WAR files as well.

5.4.8.1 To Configure Your Environment to Run the Jakarta RESTful Web Services Tests Against the Jakarta EE 10 CI

Copy <TS_HOME>/bin/ts.jte.jdk11 as <TS_HOME>/bin/ts.jte if JAVA_HOME is Java SE 11. Edit your `<TS_HOME>/bin/ts.jte` file and set the following environment variables:

1. Set the `jaxrs_impl_lib` property to point to the Jakarta RESTful Web Services CI.
   The default setting for this property is `${javaee.home}/modules/jersey-container-servlet-core.jar` .

2. Set the `servlet_adaptor` property to point to the Servlet adapter class for the Jakarta RESTful Web Services implementation.
   The default setting for this property is `org/glassfish/jersey/servlet/ServletContainer.class`, the servlet adaptor supplied in Jersey.

3. Set the `jaxrs_impl_name` property to the name of the Jakarta RESTful Web Services CI.
   The default setting for this property is `jersey`.
   An Ant script, jersey.xml, in the `<TS_HOME>/bin/xml/impl/glassfish` directory contains packaging instructions.

5.4.8.2 To Package WAR files for Deployment on the Jakarta EE 10 CI

The Jakarta EE 10 Platform TCK test suite does not come with prebuilt test WAR files for deployment on Jakarta EE 10 CI. The test suite includes a command to generate the test WAR files that will be deployed on the Jakarta EE 10 CI. The WAR files are Jersey-specific, with Jersey's servlet class and Eclipse Jersey's servlet defined in the `web.xml` deployment descriptor.

To package the Jakarta RESTful Web Services WAR files for deployment on the Jakarta EE 10 CI, complete the following steps:

1. Change to the `<TS_HOME>/bin` directory.

2. Execute the `update.jaxrs.wars` Ant target.
   In a test WAR files that has the `servlet_adaptor` property defined, this target replaces the `servlet_adaptor` value of the servlet class name property in the `web.xml` file of the WAR files to be deployed on the Jakarta EE 10 CI.

5.4.8.3 To Configure Your Environment to Run the Jakarta RESTful Web Services Tests Against a Vendor Implementation

Complete the following steps to configure your test environment to run the Jakarta RESTful Web Services tests against your vendor implementation. Before you can run the tests, you need to repackage the WAR files that contain the Jakarta RESTful Web Services tests and the VI-specific Servlet class that will be deployed on the vendor's Jakarta EE 10-compliant application server.

Copy <TS_HOME>/bin/ts.jte.jdk11 as <TS_HOME>/bin/ts.jte if JAVA_HOME is Java SE 11. Edit your

`<TS_HOME>/bin/ts.jte` file and set the following properties:

1. Set the `jaxrs_impl_lib` property to point to the JAR file that contains the vendor's Jakarta RESTful Web Services Servlet adapter implementation.
   The default setting for this property is `${javaee.home}/modules/jersey-container-servlet-core.jar` .

2. Set the `servlet_adaptor` property to point to the Servlet adapter class for the vendor's Jakarta RESTful Web Services implementation.
   The class must be located in the JAR file defined by the `jaxrs_impl_lib` property. By default, this property is set to `org/glassfish/jersey/servlet/ServletContainer.class`, the servlet adapter supplied in Jersey.

3. Set the `jaxrs_impl_name` property to the name of the Jakarta RESTful Web Services vendor implementation to be tested.
   The name of the property must be unique. An Ant file bearing this name, <jaxrs_impl_name>.xml, should be created under `<TS_HOME>/bin/xml/impl/${impl.vi}` with packaging and/or deployment instructions as described in Section 5.4.9.4, "To Repackage WAR files for Deployment on the Vendor Implementation."
   The default setting for this property is `jersey`.

5.4.8.4 To Repackage WAR files for Deployment on the Vendor Implementation

To run the Jakarta RESTful Web Services tests against a vendor's implementation in a Jakarta EE 8-compliant application server, the tests need to be repackaged to include the VI-specific servlet, and the VI-specific servlet must be defined in the deployment descriptor.

A vendor must create VI-specific Jakarta EE 10-compliant WAR files so the VI-specific Servlet class will be included instead of the Jakarta EE 10 CI-specific Servlet class.

All resource and application class files are already compiled. The Vendor needs to package these files. Jakarta EE 10 Platform TCK makes this task easier by including template WAR files that contain all of the necessary files except for the VI-specific servlet adaptor class. The Jakarta EE 10 TCK also provides a tool to help with the repackaging task.

Each test that has a Jakarta RESTful Web Services resource class to publish comes with a template deployment descriptor file. For example, the file `<TS_HOME>/src/com/sun/ts/tests/jaxrs/ee/rs/get/web.xml.template` contains the following elements:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" \
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" \
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee \
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>CTSJAX-RSGET</servlet-name>
        <servlet-class>servlet_adaptor</servlet-class>
        <init-param>
            <param-name>jakarta.ws.rs.Application</param-name>
            <param-value>com.sun.ts.tests.jaxrs.ee.rs.get.TSAppConfig</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>CTSJAX-RSGET</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
</web-app>
```

In this example, the `<servlet-class>` element has a value of `servlet_adaptor`, which is a placeholder for the implementation-specific Servlet class. An Eclipse Jersey-specific deployment descriptor also comes with the Jakarta EE 10 CI, Eclipse GlassFish 6.1, and has the values for the `com.sun.jersey.spi.container.servlet.ServletContainer`:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" \
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" \
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee \
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>CTSJAX-RSGET</servlet-name>
        <servlet-class>
        org.glassfish.jersey.servlet.ServletContainer
        </servlet-class>
        <init-param>
            <param-name>jakarta.ws.rs.Application</param-name>
            <param-value>com.sun.ts.tests.jaxrs.ee.rs.get.TSAppConfig</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>CTSJAX-RSGET</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>30</session-timeout>
        </session-config>
</web-app>
```

The Jakarta EE 10 Platform TCK test suite provides a tool, `${ts.home}/bin/xml/impl/glassfish/jersey.xml`, for the Jakarta EE 10 CI that you can use as a model to help you create your own VI-specific Web test application.

The following steps explain how to create a VI-specific deployment descriptor.

1. Create a VI handler file.
   Create a VI-specific handler file `<TS_HOME>/bin/xml/impl/${impl.vi}/${jaxrs_impl_name}.xml` if one does not already exist.
   Ensure that the `jaxrs_impl_name` property is set in the `ts.jte` file and that its name is unique, to prevent another file with the same name from being overwritten.

2. Set the `servlet_adaptor` property in the `ts.jte` file.
   This property will be used to set the value of the `<servlet-class>` element in the deployment descriptor.

3. Create VI Ant tasks.
   Create a `update.jaxrs.wars` target in the VI handler file. Reference this `update.jaxrs.wars` target in the `jersey.xml` file.
   This target will create a `web.xml.${jaxrs_impl_name}` for each test that has a deployment descriptor template. The `web.xml.${jaxrs_impl_name)` will contain the VI-specific Servlet class name. It will also create the test WAR files will be created under the `<TS_HOME>/dist` directory. For example:

```
cd $TS_HOME/dist/com/sun/ts/tests/jaxrs/ee/rs/get/
ls jaxrs_rs_get_web.war.jersey
jaxrs_rs_get_web.war.${jaxrs_impl_name}
```

4. Change to the `<TS_HOME>/bin` directory and execute the `update.jaxrs.wars` Ant target.
   This creates a `web.xml.${jaxrs_impl_name}` file for each test based on the VI's servlet class name and repackages the tests.

5.4.9 Jakarta Connector Test Setup

The Jakarta Connector tests verify that a Jakarta EE 10 server correctly implements the Jakarta Connector V1.7 specification. The Connector compatibility tests ensure that your Jakarta EE 10 server still supports the Connector V1.0 functionality.

The `config.vi` target is run to configure the Jakarta EE 10 server for running Connector tests. The `config.vi` target calls the `config.connecto`r target, which is defined in `TS_HOME/bin/xml/impl/glassfish/s1as.xml`, to deploy the RAR files listed in Section 5.4.10.1, "Extension Libraries." and create the required connection resources and connection pools used for the Connector tests. The `config.vi` target also performs several other tasks, such as creating required users and security mappings, setting appropriate JVM options, etc. that also are needed to run the Connector tests.

5.4.9.1 Extension Libraries

The following Connector files are deployed as part of the `config.vi` Ant target:

- `whitebox-mixedmode.rar`
- `whitebox-tx-param.rar`
- `whitebox-multianno.rar`
- `whitebox-tx.rar`
- `whitebox-anno_no_md.rar`
- `whitebox-notx-param.rar`
- `whitebox-xa-param.rar`
- `whitebox-mdcomplete.rar`
- `whitebox-notx.rar`
- `whitebox-xa.rar`
- `old-dd-whitebox-notx-param.rar`

- `old-dd-whitebox-xa-param.rar`

- `old-dd-whitebox-tx.rar`

- `old-dd-whitebox-notx.rar`

- `old-dd-whitebox-xa.rar`

- `old-dd-whitebox-tx-param.rar`

> RAR files with an `old` prefix are used to test the support of RAs that are bundled with an older version of the `ra.xml` files.

The manifest file in each RAR file includes a reference to the whitebox extension library. The `whitebox.jar` file is a Shared Library that must be deployed as a separate entity that all the Jakarta Connector RAR files access. This extension library is needed to address classloading issues.

The RAR files that are used with Jakarta EE 10 Platform TCK test suite differ from those that were used in earlier test suites. Jakarta EE 10 Platform TCK no longer bundles the same common classes into every RAR file. Duplicate common classes have been removed and now exist in the `whitebox.jar` file, an Installed Library that is deployed and is made available before any other RAR files are deployed.

This was done to address the following compatibility issues:

- Portable use of Installed Libraries for specifying a resource adapter's shared libraries
  See section EE.8.2.2 of the Jakarta EE 10 platform specification and section 20.2.0.1 in the Jakarta Connectors (formerly JCA) 1.7 specification, which explicitly state that the resource adapter server may employ the library mechanisms in Jakarta EE 10.

- Support application-based standalone connector accessibility
  Section 20.2.0.4 of the Jakarta Connectors (formerly JCA) 1.7 Specification uses the classloading requirements that are listed in section 20.3 in the specification.

5.4.9.2 Connector Resource Adapters and Classloading

Jakarta EE 10 Platform TCK has scenarios in which multiple standalone RAR files that use the same shared library (for example, `whitebox.jar`) are referenced from an application component.

Each standalone RAR file gets loaded in its own classloader. Since the application component refers to more than one standalone RAR file, all of the referenced standalone RAR files need to be made available in the classpath of the application component. In versions of the TCK prior to Java EE 5, since each standalone RAR file contained a copy of the `whitebox.jar` file, every time there was a reference to a class in the `whitebox.jar` file from a standalone RAR, the reference was resolved by using the private version of `whitebox.jar` (the `whitebox.jar` file was bundled in each standalone RAR file). This approach can lead to class type inconsistency issues.

5.4.9.3 Use Case Problem Scenario

Assume that RAR1 and RAR2 are standalone RAR files that are referred to by an application, where:

- RAR1's classloader has access to RAR1's classes and its copy of `whitebox.jar`. (RAR1's classloader contains RAR1's classes and `whitebox.jar`)

- RAR2's classloader has access to RAR2's classes and its copy of `whitebox.jar`. (RAR2's classloader contains RAR2's classes and `whitebox.jar`)

When the application refers to both of these RAR files, a classloader that encompasses both of these classloaders (thereby creating a classloader search order) is provided to the application. The classloader search order could have the following sequence: ([RAR1's Classloader: RAR1's classes and whitebox.jar], [RAR2's Classloader: RAR2's classes and whitebox.jar]).

In this scenario, when an application loads a class (for example, class `Foo`) in `whitebox.jar`, the application gets class `Foo` from RAR1's classloader because that is first in the classloader search order. However, when this is cast to a class (for example, `Foo` or a subclass of `Foo` or even a class that references `Foo`) that is obtained from RAR2's classloader (a sequence that is typically realized in a `ConnectionFactory` lookup), this would result in a class-cast exception.

The portable way of solving the issues raised by this use case problem scenario is to use installed libraries, as described in section EE.8.2.2 in the Jakarta EE 10 platform specification. If both RAR files (RAR1 and RAR2) reference `whitebox.jar` as an installed library and the application server can use a single classloader to load this common dependency, there will be no type-related issues.

In the CI Eclipse GlassFish 6.1, `domain-dir/lib/applibs` is used as the Installed Library directory and is the location to which the `whitebox.jar` file gets copied.

5.4.9.4 Required Porting Package

The Jakarta EE 10 Platform TCK test suite treats the `whitebox.jar` dependency as an Installed Library dependency instead of bundling the dependency (or dependencies) with every RAR file. Each RAR file now contains a reference to the `whitebox.jar` file through its Manifest files Extension-List attribute.

It is necessary to identify the `whitebox.jar` to the connector server as an installed library. The mechanism used to identify the `whitebox.jar` file to the connector server as an Installed Library must allow the Installed Libraries to have dependencies on Jakarta EE APIs. In other words, because the `whitebox.jar` file depends on Jakarta EE APIs, one cannot simply put the `whitebox.jar` file into a `java.ext.dir` directory , which gets loaded by the VM extension classloader, because that mechanism does not allow the `whitebox.jar` file to support its dependencies on the Jakarta EE APIs. For this reason, the Installed Library must support access to the Jakarta EE APIs.

See section EE.8.2.2 in the Jakarta EE 10 platform specification for information about the compatible implementation's support for Installed libraries. However, note that this section does not recommend a mechanism that a deployer can use to provide Installed Libraries in a portable manner.

5.4.9.5 Creating Security Mappings for the Connector RAR Files

The Ant target `create.security.eis.mappings` in the `<TS_HOME>/bin/xml/impl/glassfish/connector.xml` file maps Resource Adapter user information to existing user information in the CI.

For the Eclipse GlassFish 6.1 CI, these mappings add a line to the `domain.xml` file, similar to the one shown below, and should include 6 of these mappings:

```
<jvm-options>-Dwhitebox-tx-map=cts1=j2ee</jvm-options>
<jvm-options>-Dwhitebox-tx-param-map=cts1=j2ee</jvm-options>
<jvm-options>-Dwhitebox-notx-map=cts1=j2ee</jvm-options>
<jvm-options>-Dwhitebox-notx-param-map=cts1=j2ee</jvm-options>
<jvm-options>-Dwhitebox-xa-map=cts1=j2ee</jvm-options>
<jvm-options>-Dwhitebox-xa-param-map=cts1=j2ee</jvm-options>
```

If the `rauser1` property has been set to `cts1` and the `user` property has been set to `j2ee` in the `ts.jte` file, the following mappings would be required in the connector runtime:

- For RA `whitebox-tx`, map `cts1` to `j2ee`

- For RA `whitebox-tx-param`, map `cts1` to `j2ee`

- For RA `whitebox-notx`, map `cts1` to `j2ee`

- For RA `whitebox-notx-param`, map `cts1` to `j2ee`

- For RA `whitebox-xa`, map `cts1` to `j2ee`

- For RA `whitebox-xa-param`, map `cts1` to `j2ee`

5.4.9.6 Creating Required Server-Side JVM Options

Create the required JVM options that enable user information to be set and/or passed from the `ts.jte` file to the server. The RAR files use some of the property settings in the `ts.jte` file.

To see some of the required JVM options for the server under test, see the `s1as.jvm.options` property in the `ts.jte` file. The connector tests require that the following subset of JVM options be set in the server under test:

```
-Dj2eelogin.name=j2ee
-Dj2eelogin.password=j2ee
-Deislogin.name=cts1
-Deislogin.password=cts1
```

5.4.10 XA Test Setup

The XA Test setup requires that the `ejb_Tsr.ear` file be deployed as part of the `config.vi` Ant target. The `ejb_Tsr.ear` file contains an embedded RAR file, which requires the creation of a connection-pool and a connector resource.

For more details about the deployment of `ejb_Tsr.ear` and its corresponding connection pool and connector resource values, see the `setup.tsr.embedded.rar` Ant target in the `<TS_HOME>/bin/xml/impl/glassfish/s1as.xml` file.

The XA tests reference some `JDBCWhitebox` name bindings that are created as part of the `config.vi` target but those name bindings are not tied to any JDBC RAR files. Instead, the following XA-specific connection pool ids are referenced by the XA tests:

- `eis/JDBCwhitebox-xa`

- `eis/JDBCwhitebox-tx`

- `eis/JDBCwhitebox-notx`

For more details on these JDBC resources, examine the `add.jdbc.resources` target in the same file to see the required JDBC resources that are created. Both targets are called as part of the `config.vi` target.

Complete the following steps (create JDBC connection pools and JDBC resource elements, deploy the RAR files) to set up your environment to run the XA tests:

1. Create a JDBC connection pool with the following attributes:

   - Set the resource type to `javax.sql.XADataSource`

   - Set the datasourceclassname to `org.apache.derby.jdbc.EmbeddedXADataSource`

   - Set the property to `DatabaseName=<Derby-location>:user=cts1:password=cts1`

   - Set the connection pool name to `cts-derby-XA-pool`
     For example, you could use the `asadmin` command line utility in the Jakarta EE 10 CI, Eclipse GlassFish 6.1 to create this connection pool:

   ```
   asadmin create-jdbc-connection-pool --restype javax.sql.XADataSource \
   --datasourceclassname org.apache.derby.jdbc.EmbeddedXADataSource  \
   --property 'DatabaseName=/tmp/DerbyDB:user=cts1:password=cts1' \
   cts-derby-XA-pool
   ```

   See the `add.jdbc.pools` Ant target in the `s1as.xml` file for additional information.

2. Create three JDBC connection pool elements (more specifically, the JDBC connection pool elements) with the following JNDI names:

◦ For the first connection pool element, set the connection pool id to `cts-derby-XA-pool` and the JNDI name to `eis/JDBCwhitebox-xa`

◦ For the second connection pool element, set the connection pool id to `cts-derby-XA-pool` and the JNDI name to `eis/JDBCwhitebox-tx`

◦ For the third connection pool element, set the connection pool id to `cts-derby-XA-pool` and the JNDI name to `eis/JDBCwhitebox-notx`

For example, you could use the `asadmin` command line utility in the Jakarta EE 10 CI to create the three connection pool elements:

```
asadmin  asadmin  create-jdbc-resource --connectionpoolid cts-derby-XA-pool \
  eis/JDBCwhitebox-xa
asadmin  create-jdbc-resource --connectionpoolid cts-derby-XA-pool \
  eis/JDBCwhitebox-tx
asadmin  create-jdbc-resource --connectionpoolid cts-derby-XA-pool \
  eis/JDBCwhitebox-notx
```

If two or more JDBC resource elements point to the same connection pool element, they use the same pool connection at runtime. Jakarta EE 10 Platform TCK does reuse the same connection pool ID for testing the Jakarta EE 10 CI Eclipse GlassFish 6.1.

3. Make sure that the following EIS and RAR files have been deployed into your environment before you run the XA tests:

◦ For the EIS resource adapter, deploy the following RAR files. Most of these files are standalone RAR files, but there is also an embedded RAR file that is contained in the `ejb_Tsr.ear` file. With the CI, these RAR files are deployed as part of the config.vi Ant task. The following RAR files are defined in the `ts.jte` file.

```
whitebox-tx=java:comp/env/eis/whitebox-tx
whitebox-notx=java:comp/env/eis/whitebox-notx
whitebox-xa=java:comp/env/eis/whitebox-xa
whitebox-tx-param=java:comp/env/eis/whitebox-tx-param
whitebox-notx-param=java:comp/env/eis/whitebox-notx-param
whitebox-xa-param=java:comp/env/eis/whitebox-xa-param
whitebox-embed-xa=
"__SYSTEM/resource/ejb_Tsr#whitebox-
xa#com.sun.ts.tests.common.connector.whitebox.TSConnectionFactory"
```

◦ The embedded RAR files are located in the `<TS_HOME>/src/com/sun/ts/tests/xa/ee/tsr` directory.

◦ The EIS RAR files are located in the following directory: `<TS_HOME>/src/com/sun/ts/tests/common/connector/whitebox`
RAR files in the `<TS_HOME>/src/com/sun/ts/tests/common/connector` directory must be built before any dependent tests can pass. Deployment can either be done ahead of time or at runtime, as

long as connection pools and resources are established prior to test execution.

The XA tests make use of existing connector RAR files, which typically get deployed when the `config.vi` Ant task is run. Note that there are currently no `JDBCwhitebox` source files and no `JDNCwhitebox` RAR files.

5.4.11 Jakarta Enterprise Beans 4.0 Test Setup

This section explains special configuration that needs to be completed before running the Jakarta Enterprise Beans 4.0 DataSource and Stateful Timeout tests.

The Jakarta Enterprise Beans 4.0 DataSource tests do not test XA capability and XA support in a database product is typically not required for these tests. However, some Jakarta EE products could be implemented in such a way that XA must be supported by the database. For example, when processing the @DataSourceDefinition annotation or `<data-source>` descriptor elements in tests, a Jakarta EE product infers the datasource type from the interface implemented by the driver class. When the driver class implements multiple interfaces, such as `javax.sql.DataSource`, `javax.sql.ConnectionPoolDataSource`, or `javax.sql.XADataSource`, the vendor must choose which datasource type to use. If `javax.sql.XADataSource` is chosen, the target datasource system must be configured to support XA. Consult the documentation for your database system and JDBC driver for information that explains how to enable XA support.

5.4.11.1 To Configure the Test Environment to Run the Jakarta Enterprise Beans 4.0 DataSource Tests

The EJB 3.2 DataSource tests under the following `tests/ejb30` directories require you to run the Ant task in Step 2.

- `com/sun/ts/tests/ejb30/lite/packaging/war/datasource`

- `com/sun/ts/tests/ejb30/misc/datasource`

- `com/sun/ts/tests/ejb30/assembly/appres`

If your database vendor requires you to set any vendor-specific or less common DataSource properties, complete step 1 and then complete step 2, as explained below.

1. Set any vendor-specific or less common datasource properties with the `jdbc.datasource.props` property in the `ts.jte` file.
   The value of the property is a comma-separated array of name-value pairs, in which each property pair uses a `"name=value"` format, including the surrounding double quotes.
   The value of the property must not contain any extra spaces.
   For example:

```
jdbc.datasource.props="driverType=thin","name2=vale2"
```

2. Run the `configure.datasource.tests` Ant target to rebuild the Jakarta Enterprise Beans 4.0 DataSource Definition tests using the new database settings specified in the `ts.jte` file.
This step must be completed for Jakarta EE 10 and Jakarta EE 10 Web Profile testing.

5.4.11.2 To Configure the Test Environment to Run the Jakarta Enterprise Beans 4.0 Stateful Timeout Tests

The Jakarta Enterprise Beans 4.0 Stateful Timeout Tests in the following test directories require special setup:

- `com/sun/ts/tests/ejb30/lite/stateful/timeout`

- `com/sun/ts/tests/ejb30/bb/session/stateful/timeout`

  1. Set the `javatest.timeout.factor` property in the `ts.jte` file to a value such that the JavaTest harness does not time out before the test completes.
  A value of 2.0 or greater should be sufficient.

  2. Set the `test.ejb.stateful.timeout.wait.seconds` property, which specifies the minimum amount of time, in seconds, that the test client waits before verifying the status of the target stateful bean, to a value that is appropriate for your server.
  The value of this property must be an integer number. The default value is 480 seconds. This value can be set to a smaller number (for example, 240 seconds) to speed up testing, depending on the stateful timeout implementation strategy in the target server.

5.4.12 Jakarta Enterprise Beans Timer Test Setup

Set the following properties in the `ts.jte` file to configure the Jakarta Enterprise Beans timer tests:

```
ejb_timeout=[interval_in_milliseconds]
ejb_wait=[interval_in_milliseconds]
```

- The `ejb_timeout` property sets the duration of single-event and interval timers. The default setting and recommended minimum value is `30000` milliseconds.

- The `ejb_wait` property sets the period for the test client to wait for results from the `ejbTimeout()` method. The default setting and recommended minimum value is `60000` milliseconds.

Jakarta EE 10 Platform TCK does not have a property that you can set to configure the date for date timers.

The timer tests use the specific `jndi-name` jdbc`/DBTimer` for the datasource used for container-managed persistence to support the use of an XA datasource in the Jakarta EE 10 timer implementation. For example:

```
<jdbc-resource enabled="true" jndi-name="jdbc/DBTimer"
                object-type="user" pool-name="cts-javadb-XA-pool" />
```

The test directories that use this datasource are:

```
ejb/ee/timer
ejb/ee/bb/entity/bmp/allowedmethostest
ejb/ee/bb/entity/cmp20/allowedmethodstest
```

When testing against the Jakarta Platform, Enterprise Edition CI, Eclipse GlassFish 6.1, you must first start the Derby DB and initialize it in addition to any other database you may be using, as explained in Configuring the Jakarta EE 10 CI as the VI.

5.4.13 Entity Bean Container-Managed Persistence Test Setup for Jakarta Enterprise Beans V 1.1

Your Jakarta Platform, Enterprise Edition implementation should map the following instance variables to a backend datastore. These are needed to run the TCK entity bean container-managed persistence (cmp1.1) tests.

The Jakarta Platform, Enterprise Edition CI creates the table used by container-managed persistence by appending `"Table"` to the bean name. For example, if your bean name is `TestEJB`, the table that will be created will be `TestEJBTable`.

The container-managed fields for most `cmp` tests must have the following names and the following Java types:

Column Name	Java Type
`key_id`	Integer
`brand_name`	String
`price`	Float

These instance variable names correspond to the following database schema:

```
KEY_ID (INTEGER NOT NULL)
BRAND_NAME (VARCHAR(32))
PRICE (FLOAT)
PRIMARY KEY (KEY_ID)
```

These instance variables are used in the transactional entity test bean for the transactional test cases (`tx`) and in the database support utility class for the bean behavior test cases (`bb`). These instance variables, used in the enterprise bean tests, must be accessible at deployment time.

The Jakarta Platform, Enterprise Edition CI, Eclipse GlassFish 6.1 provides the container-managed persistence implementation-specific features as part of its runtime XML file. Your Jakarta Platform, Enterprise Edition platform implementation needs to map the container-managed fields to the appropriate backend datastore. The manner in which you do this is implementation-specific. The `DeploymentInfo` class provides all of the runtime XML information as an object that is passed to the `TSDeploymentInterface` implementation.

For a list of SQL statements used in CMP 1.1 finders, refer to SQL Statements for CMP 1.1 Finders.

5.4.14 Jakarta Persistence API Test Setup

The Jakarta Persistence API tests exercise the requirements as defined in the Jakarta Persistence API Specification. This specification defines a persistence context to be a set of managed entity instances, in which for any persistent identity there is a unique entity instance. Within the persistence context, the entity instances and their life cycles are managed by the entity manager.

Within a Jakarta Platform, Enterprise Edition environment, support for both container-managed and application-managed entity managers is required. Application-managed entity managers can be Jakarta Transactions or resource-local. Refer to Chapter 7 of the Jakarta Persistence API Specification (`https://jakarta.ee/specifications/persistence/3.0`) for additional information regarding entity managers.

5.4.14.1 To Configure the Test Environment to Run the Jakarta Persistence Pluggability Tests

The Jakarta Persistence Pluggability tests under the `src/com/sun/ts/tests/jpa/ee/pluggability` directory ensure that a third-party persistence provider is pluggable, in nature.

After Java EE 7 TCK, the pluggability tests were rewritten to use a stubbed-out legacy JPA 2.1 implementation, which is located in the `src/com/sun/ts/jpa/common/pluggibility/altprovider` directory.

In Java EE 7 TCK, the Persistence API pluggability tests required special setup to run. This is no longer the case, since Jakarta EE 10 Platform TCK now enables the pluggability tests to be executed automatically along with all the other Persistence tests. The Jakarta Persistence tests have a new

directory structure. In Java EE 7 TCK, the tests were in the `src/com/sun/ts/tests/ejb30/persistence` directory. The Jakarta EE 10 tests are now in the `src/com/sun/ts/tests/jpa` directory.

5.4.14.2 Enabling Second Level Caching Support

Jakarta Persistence supports the use of a second-level cache by the persistence provider. The `ts.jte` file provides a property that controls the TCK test suite's use of the second-level cache.

The `persistence.second.level.caching.supported` property is used to determine if the persistence provider supports the use of a second-level cache. The default value is true. If your persistence provider does not support second level caching, set the value to false.

5.4.14.3 Persistence Test Vehicles

The persistence tests are run in a variety of "vehicles" from which the entity manager is obtained and the transaction type is defined for use. There are six vehicles used for these tests:

- `stateless3`: Bean-managed stateless session bean using JNDI to lookup a Jakarta Transactions `EntityManager`; uses `UserTransaction` methods for transaction demarcation

- `stateful3`: Container-managed stateful session bean using `@PersistenceContext` annotation to inject Jakarta Transactions `EntityManager;` uses container-managed transaction demarcation with a transaction attribute (required)

- `appmanaged`: Container-managed stateful session bean using `@PersistenceUnit` annotation to inject an `EntityManagerFactory`; the `EntityManagerFactory` API is used to create an Application-Managed Jakarta Transactions `EntityManager`, and uses the container to demarcate transactions

- `appmanagedNoTx`: Container-managed stateful session bean using `@PersistenceUnit` annotation to inject an `EntityManagerFactory`; the `EntityManagerFactory` API is used to create an Application-Managed Resource Local `EntityManager`, and uses the `EntityTransaction` APIs to control transactions

- `pmservlet`: Servlet that uses the `@PersistenceContext` annotation at the class level and then uses JNDI lookup to obtain the `EntityManager`; alternative to declaring the persistence context dependency via a `persistence-context-ref` in `web.xml` and uses `UserTransaction` methods for transaction demarcation

- `puservlet`: Servlet that injects an `EntityManagerFactory` using the `@PersistenceUnit` annotation to create a to Resource Local `EntityManager`, and uses `EntityTransaction` APIs for transaction demarcation

> For vehicles using a `RESOURCE_LOCAL` transaction type, be sure to configure a non-transactional resource with the logical name `jdbc/DB_no_tx`. Refer to the `ts.jte` file for information about the `jdbc.db` property.

5.4.14.4 GeneratedValue Annotation

The Jakarta Persistence API Specification also defines the requirements for the `GeneratedValue` annotation. The default for this annotation is `GenerationType.AUTO`. Per the specification, `AUTO` indicates that the persistence provider should pick an appropriate strategy for the particular database. The `AUTO` generation strategy may expect a database resource to exist, or it may attempt to create one.

The `db.supports.sequence` property is used to determine if a database supports the use of SEQUENCE. If it does not, this property should be set to false so the test is not run. The default value is true.

If the database under test is not one of the databases defined and supported by TCK, the user will need to create an entry similar to the one listed in Example 5-1.

Example 5-1 GeneratedValue Annotation Test Table

```
DROP TABLE SEQUENCE;
CREATE TABLE SEQUENCE (SEQ_NAME VARCHAR(10), SEQ_COUNT INT, CONSTRAINT SEQUENCE_PK /
PRIMARY KEY (SEQ_NAME) );
INSERT into SEQUENCE(SEQ_NAME, SEQ_COUNT) values ('SEQ_GEN', 0) ;
```

You should add your own table to your chosen database DDL file provided prior to running these tests. The Data Model used to test the Jakarta Persistence Query Language can be found in Appendix D, "EJBQL Schema."

The `persistence.xml` file, which defines a persistence unit, contains the `unitName CTS-EM` for Jakarta Transactions entity managers. This corresponds to `jta-data-source`, `jdbc/DB1`, and to `CTS-EM-NOTX` for `RESOURCE_LOCAL` entity managers, which correspond to a `non-jta-data-source jdbc/DB_no_tx`.

5.4.15 Jakarta Messaging Test Setup

This section explains how to set up and configure the Jakarta EE 10 Platform TCK test suite before running the Jakarta Messaging tests.

> ℹ️ The client-specified values for `JMSDeliveryMode`, `JMSExpiration`, and `JMSPriority` must not be overridden when running the TCK Jakarta Messaging tests.

5.4.15.1 To Configure a Slow Running System

Make sure that the following property has been set in the `ts.jte` file:

```
jms_timeout=10000
```

This property specifies the length of time, in milliseconds, that a synchronous receive operation will wait for a message. The default value of the property should be sufficient for most environments. If, however, your system is running slowly and you are not receiving the messages that you should be, you need to increase the value of this parameter.

5.4.15.2 To Test Your Jakarta Messaging Resource Adapter

If your implementation supports Jakarta Messaging as a Resource Adapter, you must set the name of the `jmsra.name` property in the `ts.jte` file to the name of your Jakarta Messaging Resource Adapter. The default value for the property is the name of the Jakarta Messaging Resource Adapter in the Jakarta EE 10 CI.

If you modify the `jmsra.name` property, you must rebuild the Jakarta Messaging tests that use this property. You rebuild the tests by doing the following:

1. Change to the `TS_HOME/bin` directory.

2. Invoke the following Ant task:

   ```
   ant rebuild.jms.rebuildable.tests
   ```

   This rebuilds the tests under `TS_HOME/src/com/sun/ts/tests/jms/ee20/resourcedefs`.

5.4.15.3 To Create Jakarta Messaging Administered Objects

If you do not have an API to create Jakarta Messaging Administered objects, and you cannot create an Ant target equivalent to `config.vi`, you can use the list that follows and manually create the objects. If you decide to create these objects manually, you need to provide a dummy implementation of the Jakarta Messaging porting interface, `TSJMSAdminInterface`.

The list of objects you need to manually create includes the following factories, queues, and topics.

- Factories:

```
jms/TopicConnectionFactory
jms/DURABLE_SUB_CONNECTION_FACTORY, clientId=cts
jms/MDBTACCESSTEST_FACTORY, clientId=cts1
jms/DURABLE_BMT_CONNECTION_FACTORY, clientId=cts2
jms/DURABLE_CMT_CONNECTION_FACTORY, clientId=cts3
jms/DURABLE_BMT_XCONNECTION_FACTORY, clientId=cts4
jms/DURABLE_CMT_XCONNECTION_FACTORY, clientId=cts5
jms/DURABLE_CMT_TXNS_XCONNECTION_FACTORY, clientId=cts6
jms/QueueConnectionFactory
jms/ConnectionFactory
```

- Queues:

```
MDB_QUEUE
MDB_QUEUE_REPLY
MY_QUEUE
MY_QUEUE2
Q2
QUEUE_BMT
ejb_ee_bb_localaccess_mdbqaccesstest_MDB_QUEUE
ejb_ee_deploy_mdb_ejblink_casesensT_ReplyQueue
ejb_ee_deploy_mdb_ejblink_casesens_ReplyQueue
ejb_ee_deploy_mdb_ejblink_casesens_TestBean
ejb_ee_deploy_mdb_ejblink_scopeT_ReplyQueue
ejb_ee_deploy_mdb_ejblink_scope_ReplyQueue
ejb_ee_deploy_mdb_ejblink_scope_TestBean
ejb_ee_deploy_mdb_ejblink_singleT_ReplyQueue
ejb_ee_deploy_mdb_ejblink_single_ReplyQueue
ejb_ee_deploy_mdb_ejblink_single_TestBean
ejb_ee_deploy_mdb_ejblink_single_TestBeanBMT
ejb_ee_deploy_mdb_ejbref_casesensT_ReplyQueue
ejb_ee_deploy_mdb_ejbref_casesens_ReplyQueue
ejb_ee_deploy_mdb_ejbref_casesens_TestBean
ejb_ee_deploy_mdb_ejbref_scopeT_ReplyQueue
ejb_ee_deploy_mdb_ejbref_scope_Cyrano
ejb_ee_deploy_mdb_ejbref_scope_ReplyQueue
ejb_ee_deploy_mdb_ejbref_scope_Romeo
ejb_ee_deploy_mdb_ejbref_scope_Tristan
ejb_ee_deploy_mdb_ejbref_singleT_ReplyQueue
ejb_ee_deploy_mdb_ejbref_single_ReplyQueue
ejb_ee_deploy_mdb_ejbref_single_TestBean
ejb_ee_deploy_mdb_ejbref_single_TestBeanBMT
ejb_ee_deploy_mdb_enventry_casesensT_ReplyQueue
ejb_ee_deploy_mdb_enventry_casesens_CaseBean
ejb_ee_deploy_mdb_enventry_casesens_CaseBeanBMT
ejb_ee_deploy_mdb_enventry_casesens_ReplyQueue
```

```
ejb_ee_deploy_mdb_enventry_scopeT_ReplyQueue
ejb_ee_deploy_mdb_enventry_scope_Bean1_MultiJar
ejb_ee_deploy_mdb_enventry_scope_Bean1_SameJar
ejb_ee_deploy_mdb_enventry_scope_Bean2_MultiJar
ejb_ee_deploy_mdb_enventry_scope_Bean2_SameJar
ejb_ee_deploy_mdb_enventry_scope_ReplyQueue
ejb_ee_deploy_mdb_enventry_singleT_ReplyQueue
ejb_ee_deploy_mdb_enventry_single_AllBean
ejb_ee_deploy_mdb_enventry_single_AllBeanBMT
ejb_ee_deploy_mdb_enventry_single_BooleanBean
ejb_ee_deploy_mdb_enventry_single_ByteBean
ejb_ee_deploy_mdb_enventry_single_DoubleBean
ejb_ee_deploy_mdb_enventry_single_FloatBean
ejb_ee_deploy_mdb_enventry_single_IntegerBean
ejb_ee_deploy_mdb_enventry_single_LongBean
ejb_ee_deploy_mdb_enventry_single_ReplyQueue
ejb_ee_deploy_mdb_enventry_single_ShortBean
ejb_ee_deploy_mdb_enventry_single_StringBean
ejb_ee_deploy_mdb_resref_singleT_ReplyQueue
ejb_ee_deploy_mdb_resref_single_ReplyQueue
ejb_ee_deploy_mdb_resref_single_TestBean
ejb_ee_sec_stateful_mdb_MDB_QUEUE
ejb_sec_mdb_MDB_QUEUE_BMT
ejb_sec_mdb_MDB_QUEUE_CMT
jms_ee_mdb_mdb_exceptQ_MDB_QUEUETXNS_CMT
jms_ee_mdb_mdb_exceptQ_MDB_QUEUE_BMT
jms_ee_mdb_mdb_exceptQ_MDB_QUEUE_CMT
jms_ee_mdb_mdb_exceptT_MDB_QUEUETXNS_CMT
jms_ee_mdb_mdb_exceptT_MDB_QUEUE_BMT
jms_ee_mdb_mdb_exceptT_MDB_QUEUE_CMT
jms_ee_mdb_mdb_msgHdrQ_MDB_QUEUE
jms_ee_mdb_mdb_msgPropsQ_MDB_QUEUE
jms_ee_mdb_mdb_msgTypesQ1_MDB_QUEUE
jms_ee_mdb_mdb_msgTypesQ2_MDB_QUEUE
jms_ee_mdb_mdb_msgTypesQ3_MDB_QUEUE
jms_ee_mdb_mdb_rec_MDB_QUEUE
jms_ee_mdb_sndQ_MDB_QUEUE
jms_ee_mdb_sndToQueue_MDB_QUEUE
jms_ee_mdb_mdb_synchrec_MDB_QUEUE
jms_ee_mdb_xa_MDB_QUEUE_BMT
jms_ee_mdb_xa_MDB_QUEUE_CMT
testQ0
testQ1
testQ2
testQueue2
fooQ
```

- Topics:

```
MY_TOPIC
MY_TOPIC2
TOPIC_BMT
ejb_ee_bb_localaccess_mdbtaccesstest_MDB_TOPIC
ejb_ee_deploy_mdb_ejblink_casesensT_TestBean
ejb_ee_deploy_mdb_ejblink_scopeT_TestBean
ejb_ee_deploy_mdb_ejblink_singleT_TestBean
ejb_ee_deploy_mdb_ejblink_singleT_TestBeanBMT
ejb_ee_deploy_mdb_ejbref_casesensT_TestBean
ejb_ee_deploy_mdb_ejbref_scopeT_Cyrano
ejb_ee_deploy_mdb_ejbref_scopeT_Romeo
ejb_ee_deploy_mdb_ejbref_scopeT_Tristan
ejb_ee_deploy_mdb_ejbref_singleT_TestBean
ejb_ee_deploy_mdb_ejbref_singleT_TestBeanBMT
ejb_ee_deploy_mdb_enventry_casesensT_CaseBean
ejb_ee_deploy_mdb_enventry_casesensT_CaseBeanBMT
ejb_ee_deploy_mdb_enventry_scopeT_Bean1_MultiJar
ejb_ee_deploy_mdb_enventry_scopeT_Bean1_SameJar
ejb_ee_deploy_mdb_enventry_scopeT_Bean2_MultiJar
ejb_ee_deploy_mdb_enventry_scopeT_Bean2_SameJar
ejb_ee_deploy_mdb_enventry_singleT_AllBean
ejb_ee_deploy_mdb_enventry_singleT_AllBeanBMT
ejb_ee_deploy_mdb_enventry_singleT_BooleanBean
ejb_ee_deploy_mdb_enventry_singleT_ByteBean
ejb_ee_deploy_mdb_enventry_singleT_DoubleBean
ejb_ee_deploy_mdb_enventry_singleT_FloatBean
ejb_ee_deploy_mdb_enventry_singleT_IntegerBean
ejb_ee_deploy_mdb_enventry_singleT_LongBean
ejb_ee_deploy_mdb_enventry_singleT_ShortBean
ejb_ee_deploy_mdb_enventry_singleT_StringBean
ejb_ee_deploy_mdb_resref_singleT_TestBean
jms_ee_mdb_mdb_exceptT_MDB_DURABLETXNS_CMT
jms_ee_mdb_mdb_exceptT_MDB_DURABLE_BMT
jms_ee_mdb_mdb_exceptT_MDB_DURABLE_CMT
jms_ee_mdb_mdb_msgHdrT_MDB_TOPIC
jms_ee_mdb_mdb_msgPropsT_MDB_TOPIC
jms_ee_mdb_mdb_msgTypesT1_MDB_TOPIC
jms_ee_mdb_mdb_msgTypesT2_MDB_TOPIC
jms_ee_mdb_mdb_msgTypesT3_MDB_TOPIC
jms_ee_mdb_mdb_rec_MDB_TOPIC
jms_ee_mdb_mdb_sndToTopic_MDB_TOPIC
jms_ee_mdb_mdb_sndToTopic_MDB_TOPIC_REPLY
jms_ee_mdb_xa_MDB_DURABLE_BMT
jms_ee_mdb_xa_MDB_DURABLE_CMT
testT0
testT1
testT2
```

Implementations of `TSJMSAdminInterface` are called inside the JavaTest VM. The `com.sun.ts.lib.deliverable.cts.CTSPropertyManager` class, which is available to these implementations, provides access to any property in the `ts.jte` file.

5.4.16 Jakarta Authentication Service Test Setup

Jakarta Authentication Service for Containers (Authentication) 1.1 tests are security tests. The Jakarta Authentication Servlet (jaspicservlet) profile is the only required profile for Jakarta EE 10 Platform TCK. There are other optional profile tests, such as SOAP, but you are not required to run these for certification.

The test suite includes the following Ant targets that configure the test environment for the Jakarta Authentication tests

- `config_vi` target in `<TS_HOME>/bin/build.xml`

- `enable.jaspic`, also in `<TS_HOME>/bin/build.xml`

Both targets call `<TS_HOME>/bin/xml/impl/glassfish/javaee_vi.xml`, which then makes calls into `<TS_HOME>/bin/xml/impl/glassfish/s1as.xml`. You may want to examine these targets to see what is done in greater detail.

Complete the following steps before you run the Jakarta Authentication tests:

1. Configure the Jakarta Authentication-required properties in the `ts.jte` file:

   a. Set the `provider.configuration.file` property to the location of your implementation's instance `lib` directory, where it can be loaded when your implementation runtime is started.
   This file typically coexists with the `tssv.jar` file and the `provider-configuration.dtd` file.

   b. Set the `vendor.authconfig.factory` property to specify your `AuthConfigFactory` class.
   This property setting will be used by the Jakarta Authentication tests to register the test suite's provider in your `AuthConfigFactory`.

   c. Set the `logical.hostname.servlet` property to the logical host that will process Servlet requests. Servlet requests may be directed to a logical host using various physical or virtual host names or addresses. A message processing runtime may be composed of multiple logical hosts. This setting is required to properly identify the Servlet profile's application context identifier hostname. If the logical host that will process Servlet requests does not exist, you can set this to the default hostname of your implementation's Web server.

   d. Set the `servlet.is.jsr115.compatible` property based on whether or not you are running the Servlet profile in a Jakarta Authorization 1.5 compatible container.

2. Ensure that the `config.vi` Ant task has been run before running the `enable.jaspic` Ant task.
   These Ant tasks perform the following Jakarta Authentication-required steps:

- Set up users and passwords for your implementation.
  See Step 9b in Configuring Your Application Server as the VI for more information.

- Install the client-side certificate in the `trustStore` in your implementation.
  See Step 17 in Configuring Your Application Server as the VI for more information.

- Append the file `<TS_HOME>/bin/server_policy.append` to the Java policy file or files on your implementation.
  See Step 17 in Configuring Your Application Server as the VIConfiguring Your Application Server as the VI for more information.

- Appends the file `<TS_HOME>/bin/client_policy.append` to the application client's Java policy file, which is referenced in the `TestExecuteAppClient` section of the `ts.jte` file.
  See Step 18 in Configuring Your Application Server as the VIConfiguring Your Application Server as the VI for more information.

- Copies the `<TS_HOME>/lib/tssv.jar` file to your implementation instance library directory.
  The `tssv.jar` file includes the class files necessary to load `TSAuthConfigFactory` and related classes.

- Copies the TSSV configuration files (`ProviderConfiguration.xml`, `configuration.dtd`) to your implementation instance library directory.
  The `provider-configuration.dtd` file is a DTD file that resides in the same directory as the `ProviderConfiguration.xml` file and describes the `ProviderConfiguration.xml` file. This file should not be edited.

- Copies `<TS_HOME>/bin/ts.java.security` to `<JAVAEE_HOME>/domains/domain1/config/ts.java.security`, where `<JAVAEE_HOME>` is the location of your Jakarta EE 10 CI installation.

- Sets the following JVM options:

  - `-Djava.security.properties=<JAVAEE_HOME>/domains/domain1/config/ts.java.security`

  - `-Dlog.file.location=${log.file.location}`

  - `-Dprovider.configuration.file=${provider.configuration.file}`

3. Deploy the Jakarta Authentication log file processor, `<TS_HOME>/dist/com/sun/ts/tests/jaspic/util/jaspic_util_web.war`, to the implementation under test.

   > ℹ️ It may be necessary to restart your implementation after completing this step.

4. Run the tests for the profiles with which you are trying to certify.

5. After running the Jakarta Authentication tests, change back to the `<TS_HOME>/bin` directory and execute the following command:

```
cd <TS_HOME>/bin
ant disable.jaspic
```

This Ant task undoes the changes that were made to your implementation by the `enable.jaspic` target. If these changes are not reversed, your implementation may be left in an uncertain state.

5.4.17 Jakarta Authorization Test Setup

To comply with Jakarta EE 10 requirements, Jakarta Authorization must be supported in both the Web and Jakarta Enterprise Beans environments. The tests for each environment are divided into two directories:

- `src/com/sun/ts/tests/jacc/web`

- `src/com/sun/ts/tests/jacc/ejb`

When deploying the archives that contain Jakarta Authorization tests, don't deploy all the Jakarta Authorization test archives at the same time. While this may work, there have been times when it has caused problems. The recommended course of action is to deploy the test archive for the directory under test. Once done, remove that archive and move onto another directory.

The Jakarta Authorization-TCK provider acts as a delegating security provider sitting between the appserver and vendor provider. The Jakarta Platform, Enterprise Edition appserver comes with a default security provider that is defined by two system properties; for the purposes of this discussion, these are referred to as `A=DefaultProviderFactory` and `B=DefaultPolicyModule`.

TCK moves the values from A and B to two new variables: `C=DefaultProviderFactory` and `D=DefaultPolicyModule`, replacing the TCK provider classes to the variables `A` and `B` (`A=TSProviderFactory` and `B=TSPolicyModule`). This modification allows the server to call the TCK provider for all its functions, and the TCK provider in turn uses these new variables to invoke the real provider.

The property names `A`, `B`, `C`, and `D` are used for convenience here. The real property names are as follows:

- `A=jakarta.security.jacc.PolicyConfigurationFactory.provider`

- `B=jakarta.security.jacc.policy.provider`

- `C=vendor.jakarta.security.jacc.PolicyConfigurationFactory.provider`

- `D=vendor.jakarta.security.jacc.policy.provider`

To configure the Jakarta Authorization provider for the Jakarta Platform, Enterprise Edition CI, execute the Jakarta Authorization Ant target from:

```
<TS_HOME>/bin
```

This command does the following:

- Switches the system properties.
- Adds `tsprovider.jar` to Jakarta Platform, Enterprise Edition application server's classpath.
- Adds `log.file.location` system property to the Jakarta Platform, Enterprise Edition application server's system properties. This is used for generating log files, which is used for verifying Jakarta Authorization 1.5 contracts.

> When running Jakarta Authorization tests against the Jakarta EE 10 CI, if you need to restart the CI, be sure to first remove all Jakarta Authorization log files (`jacc_log.*`) from the `JAVAEE_HOME/domains/domain1/logs` directory before running the Jakarta Authorization tests again.

5.4.18 WSDL: Webservice Test and Runtime Notes

In addition to the WSDL elements described later in this section, the Jakarta Platform, Enterprise Edition CI webservice runtime DTDs contain two new optional elements for publishing and lookup of final WSDLs for a deployed webservice endpoint. These new tags are `<wsdl-publish-location>` and `<wsdl-override>`, and are used by the CTS to automate all TCK webservices tests, regardless of the host or port used to run the tests.

These WSDL tags are also used when performing file URL publishing, as required by Jakarta Implementing Web Services 1.4. Jakarta Implementing Web Services 1.4 states that http URL and file URL publishing must be supported on a Jakarta Platform, Enterprise Edition platform. In addition, the `<wsdl-override>` is used as a mechanism for satisfying the partial WSDL requirement in the Jakarta Implementing Web Services 1.4 specification. This mechanism enables the specification of the location of the final full published WSDL of a deployed webservice endpoint within the client EAR when only a partial WSDL is packaged, which enables client access to the full WSDL and correct SOAP address to communicate with the webservice.

The `<wsdl-publish-location>` tag tells the Jakarta Platform, Enterprise Edition CI where to publish the final WSDL for the deployed webservice endpoint. As stated above, the final WSDL can be published to a file URL or http URL, although the tag is really only necessary for file URL publishing, and is ignored if http URL publishing is specified (http is the default publishing used by the Jakarta Platform, Enterprise Edition CI). This tag is included in all TCK tests for consistency and to aid as a mechanism in automation.

By default, the Jakarta Platform, Enterprise Edition CI publishes the final WSDL during deployment to a http URL following a standard URL naming scheme. See below for details about the Jakarta Platform, Enterprise Edition CI runtime. This default can be overriden to explicitly do file URL publishing.

The `<wsdl-override>` tag tells the client application EAR where to lookup the final published WSDL for the deployed webservice endpoint. This will be either a `file` URL or an `http` URL to match what is specified in the `<wsdl-publish-location>` tag.

5.4.18.1 WSDL ts.jte Properties

For file URL publishing, the TCK defines two properties in the `ts.jte` file, named `wsdlRepository1` and `wsdlRepository2`, which specify the file system directory location to use for publishing final WSDLs that use file URL publishing.

The `wsdlRepository1` is used for the Vendor Jakarta Platform, Enterprise Edition Implementation. The `wsdlRepository2` is used for the CI Jakarta Platform, Enterprise Edition Implementation, and is only used for CTS webservices interoperability testing. These directories get created by the TCK harness at runtime. The default settings in the `ts.jte` file will create these directories under:

```
wsdlRepository1=<TS_HOME>/tmp/wsdlRepository1
wsdlRepository2=<TS_HOME>/tmp/wsdlRepository2
```

For file URL publishing, the WSDL tag settings could be as follows:

```
$TS_HOME/src/com/sun/ts/tests/webservices/wsdlImport/file/Simple1
Webservice Endpoint
<wsdl-publish-location>
file:wsdlRepository1/Simple1File
</wsdl-publish-location>

Webservice Client Application
<wsdl-override>
file:wsdlRepository1/Simple1File/Simple1FileSvc.wsdl
</wsdl-override>
```

In this case, the TCK harness substitutes `wsdlRepository1` with the setting in the `<TS_HOME>/bin/ts.jte` file.

For `http` URL publishing, the tag settings might be:

```
$TS_HOME/src/com/sun/ts/tests/webservices/wsdlImport/http/Simple1
Webservice Endpoint
<wsdl-publish-location>
http://webServerHost.1:webServerPort.1/Simple1Http/ws4ee?WSDL
</wsdl-publish-location>

Webservice Client Application
<wsdl-override>
http://webServerHost.1:webServerPort.1/Simple1Http/ws4ee?WSDL
</wsdl-override>
```

In this case, the TCK harness substitutes the webServerHost.1:webServerPort.1 with the settings in the `<TS_HOME>/bin/ts.jte` file.

> In the case of interop webservices tests, the TCK harness substitutes the `webServerHost.2:webServerPort.2` with the settings in the `ts.jte` file. This host and port defines the CI Jakarta Platform, Enterprise Edition implementation used as the interop test machine. See `tests/interop/webservices` for these tests.

5.4.18.2 Webservice Endpoint WSDL Elements

The following are the webservice endpoint WSDL elements:

5.4.18.2.1 Setting Endpoint Address

```
element : endpoint-address-uri
```

The endpoint address URI is used to compose the endpoint address URL through which the endpoint can be accessed. It is required for Jakarta Enterprise Beans endpoints and optional for servlet endpoints.

The `endpoint-address-uri` can have an optional leading forward slash (`/`). It must be a fixed pattern (no asterisk (`*`) wildcards).

- Jakarta Enterprise Beans Example:
  For Jakarta Enterprise Beans endpoints, the URI is relative to root of the web server; for example, if the web server is listening at `http://localhost:8000`, an endpoint address URI of `google/GoogleSearch` would result in an endpoint address of:

```
http://localhost:8000/google/GoogleSearch
```

Note that the first portion of the URI (`google`) should not conflict with the context root of any deployed web application.

```
<enterprise-beans>
    <module-name>ejb.jar</module-name>
    <ejb>
      <ejb-name>GoogleEjb</ejb-name>
      <webservice-endpoint>
        <port-component-name>GoogleSearchPort</port-component-name>
        <endpoint-address-uri>google/GoogleSearch</endpoint-address-uri>
      </webservice-endpoint>
    </ejb>
</enterprise-beans>
```

- Servlet Example:

  For servlet endpoints, the `endpoint-address-uri` is only needed if the servlet does not have a servlet-mapping `url-pattern` in its `web.xml`. Its value is relative to the context root of the servlet's web application.

```
<web>
    <module-name>web.war</module-name>
    <context-root>GoogleServletContext</context-root>
    <servlet>
        <servlet-name>MyGoogleServlet</servlet-name>
        <webservice-endpoint>
            <port-component-name>GoogleSearchPort</port-component-name>
            <endpoint-address-uri>/GoogleSearch</endpoint-address-uri>
         </webservice-endpoint>
    </servlet>
</web>
```

In this case, the target endpoint address would be:

```
http://localhost:8000/GoogleServletContext/GoogleSearch
```

5.4.18.2.2 Jakarta Enterprise Beans Endpoint Security

```
element : login-config
```

This only applies to Jakarta Enterprise Beans endpoints and is optional. It is used to specify how authentication is performed for Jakarta Enterprise Beans endpoint invocations. It consists of a single subelement named `auth-method`. `auth-method` is set to `BASIC` or `CLIENT_CERT`. The equivalent security for servlet endpoints is set through the standard web-application security elements. For example:

```
<ejb>
      <ejb-name>GoogleEjb</ejb-name>
      <webservice-endpoint>
        <port-component-name>GoogleSearchPort</port-component-name>
        <endpoint-address-uri>google/GoogleSearch</endpoint-address-uri>

        <login-config>
            <auth-method>BASIC</auth-method>
        </login-config>
      </webservice-endpoint>
</ejb>
```

5.4.18.2.3 Transport Guarantee

```
element : transport-guarantee
```

This is an optional setting on `webservice-endpoint`. The allowable values are `NONE`, `INTEGRAL`, and `CONFIDENTIAL`. If not specified, the behavior is equivalent to `NONE`. The meaning of each option is the same as is defined in the Security chapter of the Jakarta Servlet 6.0 Specification. This setting will determine the scheme and port used to generate the final endpoint address for a web service endpoint. For `NONE`, the scheme will be `HTTP` and port will be the default HTTP port. For `INTEGRAL/CONFIDENTIAL`, the scheme will be `HTTPS` and the port will be the default HTTPS port.

5.4.18.2.4 Publishing Final WSDL During Deployment

- URL publishing: no extra information required.
  The final WSDL document for each webservice endpoint is always published to a URL having the following syntax:

  ◦ Jakarta Enterprise Beans endpoints:

  ```
  <scheme>://<hostname>:<port>/<endpoint_address_uri>?WSDL
  ```

  ◦ Servlet endpoints:

  ```
  <scheme>://<hostname>:<port>/<context-root><url-pattern>?WSDL
  ```

  or

```
<scheme>://<hostname>:<port>/<context-root><endpoint_address_uri>?WSDL
```

Note that the final WSDL document returned from this URL will contain port entries for all ports within the same service.

- File publishing:

```
element : wsdl-publish-location
```

To have a copy of the final WSDL written to a file, set this element to a file URL; for example:

```
<enterprise-beans>
    <module-name>ejb.jar</module-name>
    <webservice-description>
        <webservice-description-name>GoogleSearchService
        </webservice-description-name>
        <wsdl-publish-location>file:/home/user1/GoogleSearch_final.wsdl
        </wsdl-publish-location>
    </webservice-description>
</enterprise-beans>
```

5.4.18.3 Webservice Client WSDL Elements

In the TCK for file publishing, the directory in which to publish the file WSDL is specified in the `<wsdl-publish-location>` tag for the webservice, and the full path of the WSDL file is specified in the `<wsdl-override>` tag in the client; for example:

```
<wsdl-publish-location>file:/files/wsdls/FileNested1</wsdl-publish-location>
<wsdl-override>file:/files/wsdls/FileNested1/nestedimportwsdl.wsdl</wsdl-override>
```

The Jakarta Platform, Enterprise Edition implementation defines the behavior this way because, for `wsdl-publish-location`, the App Server is potentially publishing many documents, not just one. This is because the main WSDL could have dependencies on relative imports. There is no requirement that the initial WSDL be located at the top of the hierarchy, even though that is commonly the case.

For example, in an ejb-jar with a Main.wsdl that imports a relative WSDL at `../../Relative.wsdl`, the packaging would look like:

```
META-INF/wsdl/a/b/Main.wsdl
META-INF/wsdl/Relative.wsdl
```

The `wsdl-publish-location` tells the TCK where to locate the topmost part of the WSDL content hierarchy. So, given a `wsdl-publish-location` of `/home/foo/wsdlpublishdir`, this location would look like:

```
/home/foo/wsdlpublishdir/Relative.wsdl
/home/foo/wsdlpublishdir/a/b/Main.wsdl
```

The `wsdl-override` property still always points to a specific WSDL document, which in this case would be `/home/foo/wsdlpublishdir/a/b/Main.wsdl`.

5.4.18.3.1 Resolving Container-Managed Ports

```
element : wsdl-port
```

Used to resolve the port to which a `service-ref` Service Endpoint Interface is mapped. Only required for each `port-component-ref` in the `service-ref` that does not have a `port-component-link`. For example:

```
<service-ref>
      <service-ref-name>service/GoogleSearchProxy</service-ref-name>
      <port-info>
        <service-endpoint-interface>googleclient.GoogleSearchPort
        </service-endpoint-interface>
        <wsdl-port>
            <namespaceURI>urn:GoogleSearch</namespaceURI>
            <localpart>GoogleSearchPort</localpart>
        </wsdl-port>
    </port-info>
</service-ref>
```

5.4.18.3.2 Overriding WSDL

```
element : wsdl-override
```

The `wsdl-override` element forces the deployment process to use a different WSDL than the one associated with a `service-ref` in the standard deployment module. This element is optional if the

`service-ref` WSDL is full WSDL, and is required if partial WSDL. In all cases, it must point to a valid URL of a full WSDL document. Some examples are shown below.

- To use the final WSDL generated upon deployment of the EJB endpoint shown above:

```
<service-ref>
        <service-ref-name>service/GoogleSearch</service-ref-name>
        <wsdl-override>http://localhost:8000/google/GoogleSearch?WSDL
        </wsdl-override>
</service-ref>
```

- An alternate way to do the same thing by means of a file URL that matches a webservice's `wsdl-publish-location` could be:

```
<service-ref>
        <service-ref-name>service/GoogleSearch</service-ref-name>
        <wsdl-override>file:/home/user1/GoogleSearch_final.wsdl
        </wsdl-override>
</service-ref>
```

5.4.19 Jakarta Security API Test Setup

Complete the following steps before you run the Jakarta Security API tests:

1. Set the following properties in the ts.jte file:

   > An LDAP server is required in Jakarta Security API testing. You could either use an already existing external LDAP server or use TCK script to install an internal LDAP server.

   Choose one of these two options to make an LDAP server ready for testing:

2. Use internal LDAP server - Unbounded (Recommended, and would be installed by default.)

   > 1. Ensure the ldap.server property is unbounded.
   > 2. Ensure the path of ldap.ldif.file is correct.
   > 3. Ensure the port 11389 is not occupied. Kill any related process using port 11389.

Parts of ts.jte:

- `ldap.server=unboundid`
- `ldap.install.server=true`
- `ldap.ldif.file=$\{ts.home}/bin/ldap.ldif`

3. Use external LDAP server.

1. Ensure the port of LDAP server is 11389.

2. Update ldap.install.server property as false since TCK script need not install LDAP server.

3. Import ldap.ldif file into Ldap server. You can get ldap.ldif from `<TS_HOME>/install/jakartaee/bin/ldap.ldif`.

Part of ts.jte - ldap.install.server=false

4. Configure the VI environment using these commands to run the Jakarta Security API test (including Derby, internal Ldap server which are required by Jakarta Security 1.0):

a. cd <TS_HOME>/bin

b. ant config.vi

c. Start your database.

d. ant init.ldap

If you use the external LDAP server, do not run the command ant init.ldap.

5.4.20 Signature Test Setup

The signature test setup includes the following:

5.4.20.1 sigTestClasspath Property

Set the `sigTestClasspath` property in the `<TS_HOME>/bin/ts.jte` file to include a `CLASSPATH` containing the following:

```
sigTestClasspath=jar_to_test:jars_used_by_yours
```

where:

- ``jar_to_test``: The JAR file you are validating when running the signature tests; when running against the Jakarta Platform, Enterprise Edition CI, Eclipse GlassFish 6.1, set to `javaee.jar`

- ``jars_used_by_yours``: The JAR file or files that are used or referenced by your JAR file; must include any classes that might be extended or implemented by the classes in your jar_to_test; include `rt.jar` when running against the Jakarta Platform, Enterprise Edition CI

5.4.20.2 Additional Signature Test Information

The Jakarta EE 10 Platform TCK signature tests perform verifications in two different modes: static and reflection. The test results list which SPEC API signature tests pass or fail, and the mode (static or reflection) for that test.

Any signature test failure means one of two things, either you have not yet corrected the `sigTestClasspath` or the respective SPEC API jar in your Jakarta EE implementation needs a modification to exactly match the Jakarta EE 10 Platform SPEC API. Your implementation SPEC API jars cannot contain additional public methods/fields, nor can it be missing any expected public methods/fields.

As a troubleshooting aid when failures occur, consider the following:

- All static mode tests fail:
  Verify that the `sigTestClasspath` is using correct SPEC API file names. When running on Windows, be sure to use semicolons (`;`) for `CLASSPATH` separators.

- For all other signature test failures:
  Check the report output from the test to determine which tests failed and why.

For example, some failures from an actual `JavaEESigTest_signatureTest_from_servlet.jtr` failure: ` SVR: Status Report *jakarta.servlet.jsp.jstl.core*

SVR: SignatureTest report Base version: 2.0_se11 Tested version: 2.0_se11 Check mode: src [throws normalized] Constant checking: on

Missing Fields :

jakarta.servlet.jsp.jstl.core.Config: field public final static java.lang.String jakarta.servlet.jsp.jstl.core.Config.FMT_FALLBACK_LOCALE = "jakarta.servlet.jsp.jstl.fmt.fallbackLocale" jakarta.servlet.jsp.jstl.core.Config: field public final static java.lang.String jakarta.servlet.jsp.jstl.core.Config.FMT_LOCALE = "jakarta.servlet.jsp.jstl.fmt.locale" jakarta.servlet.jsp.jstl.core.Config: field public final static java.lang.String jakarta.servlet.jsp.jstl.core.Config.FMT_LOCALIZATION_CONTEXT = "jakarta.servlet.jsp.jstl.fmt.localizationContext" jakarta.servlet.jsp.jstl.core.Config: field public final static java.lang.String jakarta.servlet.jsp.jstl.core.Config.FMT_TIME_ZONE =

"jakarta.servlet.jsp.jstl.fmt.timeZone" jakarta.servlet.jsp.jstl.core.Config: field public final static java.lang.String jakarta.servlet.jsp.jstl.core.Config.SQL_DATA_SOURCE = "jakarta.servlet.jsp.jstl.sql.dataSource" jakarta.servlet.jsp.jstl.core.Config: field public final static java.lang.String jakarta.servlet.jsp.jstl.core.Config.SQL_MAX_ROWS = "jakarta.servlet.jsp.jstl.sql.maxRows"

Added Fields :

jakarta.servlet.jsp.jstl.core.Config: field public final static java.lang.String jakarta.servlet.jsp.jstl.core.Config.FMT_FALLBACK_LOCALE = "javax.servlet.jsp.jstl.fmt.fallbackLocale" jakarta.servlet.jsp.jstl.core.Config: field public final static java.lang.String jakarta.servlet.jsp.jstl.core.Config.FMT_LOCALE = "javax.servlet.jsp.jstl.fmt.locale" jakarta.servlet.jsp.jstl.core.Config: field public final static java.lang.String jakarta.servlet.jsp.jstl.core.Config.FMT_LOCALIZATION_CONTEXT = "javax.servlet.jsp.jstl.fmt.localizationContext" jakarta.servlet.jsp.jstl.core.Config: field public final static java.lang.String jakarta.servlet.jsp.jstl.core.Config.FMT_TIME_ZONE = "javax.servlet.jsp.jstl.fmt.timeZone" jakarta.servlet.jsp.jstl.core.Config: field public final static java.lang.String jakarta.servlet.jsp.jstl.core.Config.SQL_DATA_SOURCE = "javax.servlet.jsp.jstl.sql.dataSource" jakarta.servlet.jsp.jstl.core.Config: field public final static java.lang.String jakarta.servlet.jsp.jstl.core.Config.SQL_MAX_ROWS = "javax.servlet.jsp.jstl.sql.maxRows"

SVR: Package *jakarta.servlet.jsp.jstl.core* - FAILED (STATIC MODE) `

The failure above is a little strange, isn't it? Why are there missing fields? Why are there added fields? The failure means that the `jakarta.servlet.jsp.jstl.core.Config` class needs to be updated to assign the correct values to the indicated constant fields. Basically, instead of setting `Config.FMT_FALLBACK_LOCALE = "javax.servlet.jsp.jstl.fmt.fallbackLocale"`, you should set `Config.FMT_FALLBACK_LOCALE = "jakarta.servlet.jsp.jstl.fmt.fallbackLocale"` The same correction is needed for the other identified fields as well.

Another example only with methods is:

` SVR: Status Report *jakarta.el*

SVR: SignatureTest report Base version: 4.0_se11 Tested version: 4.0_se11 Check mode: src [throws normalized] Constant checking: on

Missing Methods :

jakarta.el.ELContext: method public java.lang.Object jakarta.el.ELContext.getContext(java.lang.Class<?>) jakarta.el.ELContext: method public void jakarta.el.ELContext.putContext(java.lang.Class<?>,java.lang.Object) jakarta.el.StandardELContext: method public java.lang.Object jakarta.el.StandardELContext.getContext(java.lang.Class<?>) jakarta.el.StandardELContext: method public void jakarta.el.StandardELContext.putContext(java.lang.Class<?>,java.lang.Object)

Added Methods :

jakarta.el.ELContext: method public java.lang.Object jakarta.el.ELContext.getContext(java.lang.Class) jakarta.el.ELContext: method public void jakarta.el.ELContext.putContext(java.lang.Class,java.lang.Object) jakarta.el.StandardELContext: method public java.lang.Object jakarta.el.StandardELContext.getContext(java.lang.Class) jakarta.el.StandardELContext: method public void jakarta.el.StandardELContext.putContext(java.lang.Class,java.lang.Object) `

The failure above is a little strange, isn't it? Why are there missing methods? Why are there added methods? The failure means that the `java.lang.Object` `jakarta.el.ELContext.getContext(java.lang.Class)` method needs a signature change from `getContext(Class key)` to `getContext(Class<?> key)`. The same correction is needed for the other identified methods as well.

> Refer to Chapter 8, "Debugging Test Problems" for additional debugging information.

5.4.21 Backend Database Setup

The following sections address special backend database setup considerations:

- Setup Considerations for MySQL
- Setup Considerations for MS SQL Server

5.4.21.1 Setup Considerations for MySQL

The Jakarta Persistence API (formerly JPA) tests require delimited identifiers for the native query tests. If you are using delimited identifiers on MySQL, modify the `sql-mode` setting in the `my.cnf` file to set the ANSI_QUOTES option. After setting this option, reboot the MySQL server. Set the option as shown in this example:

```
sql-mode="STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION,ANSI_QUOTES"
```

5.4.21.2 Setup Considerations for MS SQL Server

If your database already exists and if you use a case-sensitive collation on MS SQL Server, execute the following command to modify the database and avert errors caused by case-sensitive collation:

```
ALTER DATABASE ctsdb
COLLATE Latin1_General_CS_AS ;
```

> ℹ️ You need to generate your own deployment plan for each module type, using the deployment tool that comes with your Jakarta Platform, Enterprise Edition server.

5.5 Using the JavaTest Harness Configuration GUI

You can use the JavaTest harness GUI to modify general test settings and to quickly get started with the default TCK test environment. After familiarizing yourself with these basic configuration settings, you will probably want to continue with the instructions in Modifying Environment Settings for Specific Technology Tests.

5.5.1 Basic Configuration Overview

In order for the JavaTest harness to execute the test suite, it requires information about how your computing environment is configured.

The JavaTest harness requires two types of configuration information:

1. Test environment: This is data used by the tests. For example, the path to the Java runtime, how to start the product being tested, network resources, and other information required by the tests in order to run. This information does not change frequently and usually stays constant from test run to test run.

2. Test parameters: This is information used by the JavaTest harness to run the tests. Test parameters are values used by the JavaTest harness that determine which tests in the test suite are run, how the tests should be run, and where the test reports are stored. This information often changes from test run to test run.

When you execute the JavaTest harness software for the first time, the JavaTest harness displays a Welcome dialog box that guides you through the initial startup configuration.

- If it is able to open a test suite, the JavaTest harness displays a Welcome to JavaTest dialog box that guides you through the process of either opening an existing work directory or creating a new work directory as described in the JavaTest online help.

- If the JavaTest harness is unable to open a test suite, it displays a Welcome to JavaTest dialog box that guides you through the process of opening both a test suite and a work directory as described in the JavaTest documentation.

Once the JavaTest harness GUI is displayed, whenever you choose Run Tests and then Start to begin a

test run, the JavaTest harness determines whether all of the required configuration information has been supplied:

- If the test environment and parameters have been completely configured, the test run starts immediately.

- If any required configuration information is missing, the configuration editor displays a series of questions asking you the necessary information. This is called the configuration interview. When you have entered the configuration data, you are asked if you wish to proceed with running the test.

5.5.2 The Configuration Interview

To configure the JavaTest harness to run the Jakarta EE 10 Platform TCK tests, complete the following steps. Note that you only need to complete these steps the first time you start the JavaTest harness. After you complete these steps, you can either run all or a subset of the tests, as described in Chapter 7, "Executing Tests".

1. Change to the `<TS_HOME>/bin` directory and start the JavaTest test harness:

   ```
   cd <TS_HOME>/bin
   ant gui
   ```

   The Welcome screen displays.

2. Click **File**, then click **Create Work Directory** to create a new work directory.
   If you already have a working directory you want to use, click **File**, then click **Open Work Directory** instead.
   At this point, the JavaTest harness is preconfigured to run the basic TCK tests.

3. If you want to run the test suite at this time using your current configuration settings, select **Run Tests** from the main menu, then select **Start**.
   The default tests are executed with the default configuration settings.
   If you do not want to run the test suite at this time, continue with the steps below to modify your test configuration.

4. Select **Configure** from the main menu, then select **Edit Configuration**.
   The **Configuration Welcome** screen displays.

5. Click **Next** (right arrow).
   You are prompted to specify one or more configuration files that contain information about your test environment. By default, this file is `<TS_HOME>/bin/ts.jte`.

6. Accept the default configuration file and click **Next**.
   You are prompted to specify a test environment.

7. Select either `ts_unix` or `ts_win32`, and then click **Next**.
   Choose `ts_unix` if you are running the tests in a Unix or Linux environment; choose `ts_win32` if you are running the tests under Windows.
   After making your selection and clicking **Next**, you are prompted to specify whether you want to run all or a subset of the test suite.

8. Specify whether you want to run all or a subset of the tests, and then click **Next**.
   Select **Yes** to run a subset of the tests; select **No** to run all tests.
   If you select **Yes**, proceed to the next step. If you select **No**, skip to Step 10.

9. Select the tests you want to run from the displayed test tree, and then click **Next**.
   You can select entire branches of the test tree, or use `Ctrl+Click` or `Shift+Click` to select multiple tests or ranges of tests, respectively.

10. Specify whether you want to use an exclude list, and then click **Next**.
    Select Yes to use an exclude list; select No to not use an exclude list.
    If you select **Yes**, proceed to the next step. If you select **No**, skip to Step 13.

11. Specify the exclude list you want to use, and then click **Next**.
    Select `initial` to use the default list; select `custom` to use a custom list.
    If you select `custom`, proceed to the next step. If you select `initial`, skip to Step 13.

12. Specify the custom exclude list file to use, and then click **Next**.

13. Click **Done** to accept and save your configuration settings.
    You are prompted to specify the location in which you want to save your configuration settings.

14. Specify the file in which you want to save your configuration settings, and then click **Save File**.
    You are returned to the JavaTest main window.

15. If you want to run the test suite at this time using your current configuration settings, select Run Tests from the main menu, then select **Start**.
    The default tests are executed with the settings you specified.

6 Setup and Configuration for Testing with the Jakarta EE 10 Web Profile

This chapter describes how to configure the Jakarta EE 10 Platform TCK test suite to work with your Jakarta EE 10 Web Profile test environment. It is recommended that you first set up the testing environment using the Jakarta EE 10 Web Profile CI and then with your Jakarta EE 10 Web Profile server.

6.1 Configuring the Jakarta EE 10 Web Profile Test Environment

The instructions in this section and in Configuring Your Application Server as the VI step you through the configuration process for the Solaris, Microsoft Windows, and Linux platforms.

6.1.1 To Run Tests Against a Jakarta EE 10 Web Profile Implementation

The Jakarta EE 10 Platform TCK is the Technology Compatibility Kit (TCK) for the Jakarta Platform, Enterprise Edition as well as the Jakarta EE 10 Web Profile. Implementations of the full Jakarta Platform, Enterprise Edition must pass all of the tests as defined by Jakarta EE 10 Platform TCK Rules in Chapter 2, "Procedure for Jakarta Platform, Enterprise Edition 10 Certification".

Implementations of the Jakarta EE 10 Web Profile must run the tests that verify requirements defined by the Jakarta EE 10 Web Profile Specification. These tests are defined by the Rules in Chapter 3, "Procedure for Jakarta Platform, Enterprise Edition 10 Web Profile Certification". These requirements are a subset of the tests contained in the Jakarta EE 10 Platform TCK test suite. The test suite provides a mechanism whereby only those tests for the Jakarta EE 10 Web Profile will be run. The following steps explain how to use this mechanism.

1. Set the `javaee.level` property to `web` in the `<TS_HOME>/bin/ts.jte` file.

   ```
   javaee.level=web
   ```

   This setting will only allow WAR files (that is, no EAR files) to be passed to the Deployment Porting Package. This is the minimal set of signature requirements that vendors must support for Web Profile. Specifying a javaee.level of "web" with nothing else implies there are NO additional technologies existing within the vendors implementation. Again, "web" only covers REQUIRED technologies for the Jakarta EE 10 Web Profile.

2. Set the `javaee_web_profile` keyword in one of the following ways:

◦ In batch mode, change to a test directory and execute the following command:

```
ant -Dkeywords=javaee_web_profile runclient
```

Only tests that are required by the Jakarta EE 10 Web Profile will be run.

> If you start a test run in a test directory that contains no Jakarta EE 10 Web Profile tests, the test run will be aborted and the test harness will report that no tests were found.

- In the JavaTest GUI, open the test suite and perform the following steps:

  1. Select View, then select Filters, then select CurrentConfiguration.

  2. Select Configure, then select ChangeConfiguration, then select Keywords.

  3. In the Keywords dialog, select the Select Tests that Match check box, specify the `javaee_web_profile` keyword in the field, then click Done.
     Only those tests that are valid in the Jakarta EE 10 Web Profile will be enabled in the test tree.

7 Executing Tests

The Jakarta EE 10 Platform TCK uses the JavaTest harness to execute the tests in the test suite. For detailed instructions that explain how to run and use JavaTest, see the JavaTest User's Guide and Reference.

This chapter includes the following topics:

- Jakarta EE 10 Platform TCK Operating Assumptions

- Starting JavaTest

- Validating Your Test Configuration

- Running a Subset of the Tests

- Test Reports

> The instructions in this chapter assume that you have installed and configured your test environment as described in Chapter 4, "Installation," and Chapter 5, "Setup and Configuration," respectively.

7.1 Jakarta EE 10 Platform TCK Operating Assumptions

The following are assumed in this chapter:

- Jakarta EE 10 CI is installed and configured as described in this guide.

- Detailed configuration will vary from product to product. In this guide, we provide details for configuring the Jakarta EE CI, Eclipse GlassFish 6.1. If you are using another CI, refer to that product's setup and configuration documentation.

- Java SE 8 software is correctly installed and configured on the host machine.

- Jakarta EE 10 Platform TCK is installed and configured as described in this guide.

- Implementations of the technologies to be tested are properly installed and configured.

7.2 Starting JavaTest

There are two general ways to run Jakarta EE 10 Platform TCK using the JavaTest harness software:

- Through the JavaTest GUI

- From the command line in your shell environment

Running the JavaTest harness from JavaTest GUI is recommended for initial configuration procedures, for validating your configuration, for selecting tests to run, and for general ease-of-use when running tests and viewing test reports.

Running the JavaTest harness from the command line is useful in headless server configurations, and for running tests in batch mode.

> ℹ️ The `build.xml` file in `<TS_HOME>/bin` contains the various Ant targets for the Jakarta EE 10 Platform TCK test suite

7.2.1 To Start JavaTest in GUI Mode

1. Set the `TS_HOME` environment variable to the directory in which the Jakarta EE 10 Platform TCK is installed.

2. Change to the `<TS_HOME>/bin` directory.

3. Copy <TS_HOME>/bin/ts.jte.jdk11 as <TS_HOME>/bin/ts.jte for JakartaEE 10. Ensure that the `ts.jte` file contains information relevant to your setup.
   Refer to Chapter 5, "Setup and Configuration," for detailed configuration instructions.

4. Execute the `ant gui` target to start the JavaTest GUI:

```
ant gui
```

Using the JavaTest GUI to run TCK tests is described later in this guide. For detailed information about using the JavaTest interface, see the JavaTest User's Guide.

The forward and reverse keywords are available to filter the interop and/or rebuildable tests during a selected test run when running tests in the following directory only:

```
<TS_HOME>/src/com/sun/ts/tests/interop
```

Forward tests are interop tests that run from the Vendor Implementation to the Compatible Implementation, as well as rebuildable tests that run only against the Vendor Implementation. Reverse tests (with test names ending in `_reverse`) are interop tests that run from the Compatible Implementation to the Vendor Implementation, as well as rebuildable tests that run only against the Compatible Implementation.

To set one of these keywords in the Javatest GUI, select the Configure menu item, then select Change Configuration, then select Keywords, and set the appropriate keyword.

When one of these keywords has been set, executing tests in the directories above causes only those tests that match the keyword to be run. This can be useful when trying to debug failures with a particular test configuration. Note, however, for certification all tests in both directions must pass.

7.2.2 To Start JavaTest in Command-Line Mode

1. Set the `TS_HOME` environment variable to the directory in which Jakarta EE 10 Platform TCK was installed.

2. Change to any subdirectory under `<TS_HOME>/src/com/sun/ts/tests`.

3. Ensure that the `ts.jte` file contains information relevant to your setup.
   Refer to Chapter 5, "Setup and Configuration," for detailed configuration instructions.

4. Execute the `runclient` Ant target to start the JavaTest:

```
ant runclient
```

This runs all tests in the current directory and any subdirectories.

Example 7-1 Running the Jakarta EE 10 Platform TCK Signature Tests

To run the Jakarta EE 10 Platform TCK signature tests, enter the following commands:

```
cd <TS_HOME>/src/com/sun/ts/tests/signaturetest/javaee
ant runclient
```

Example 7-2 Running a Single Test Directory

To run a single test directory in the `forward` direction, enter the following commands:

```
cd <TS_HOME>/src/com/sun/ts/tests/jaxws/api/jakarta_xml_ws/Dispatch
ant -Dkeywords=forward runclient
```

Example 7-3 Running a Subset of Test Directories

To run a subset of test directories in the `reverse` direction, enter the following commands:

```
cd <TS_HOME>/src/com/sun/ts/tests/jaxws/api
ant -Dkeywords=reverse runclient
```

7.3 Validating Your Test Configuration

7.3.1 To Validate Your Configuration in GUI Mode

1. Start the JavaTest GUI and step through the basic configuration steps, if required, as described in Section 5.5.2, "The Configuration Interview."

2. In the JavaTest GUI tree view, expand the following directories: `com`, `sun`, `ts`, `tests`, `samples`.

3. Highlight the `samples` directory, right-click, and choose **Execute These Tests**.
   If a work directory has not been specified, you are prompted to specify or create a new one.

4. From the **JavaTest** main menu, select **File**, then select **Create Work Directory**. The **Create Work Directory** dialog is displayed.

5. Locate or enter the name of the directory to which the test harness will write temporary files (for example, `/tmp/JTWork`), and click **Create**.

6. From the JavaTest main menu, select **Run Tests**, then select **Start** to run the default tests.
   If your configuration information is incomplete, you are prompted to supply the missing parameters.
   The JavaTest status bar grows while JavaTest tracks statistics relative to the files done, tests found, and tests done.

7. Check the results.

Test progress and results are displayed by the JavaTest harness.

7.3.2 To Validate Your Configuration in Command-Line Mode

1. Go to the `<TS_HOME>/src/com/sun/ts/tests/samples` directory.

2. Start the the test run by executing the following command:

```
ant runclient
```

All sample tests will be run, and should pass.

3. Generate test reports by executing the following commands:

   a. Change to the `<TS_HOME>/bin` directory:

   ```
   cd <TS_HOME>/bin
   ```

   b. Run the `report` Ant target:

   ```
   ant report
   ```

   Reports are written to the report directory you specified in `<TS_HOME>/bin/ts.jte`. If no report directory is specified, reports are written to the `/tmp/JTreport` directory (Solaris/Linux) or `C:\temp\JTreport` (Windows).

7.4 Running a Subset of the Tests

7.4.1 To Run a Subset of Tests in GUI Mode

1. From the JavaTest main menu, select **Configure**, then select **Edit Configuration**.

2. In the Configuration Editor, select **Specify Tests to Run?** from the option list on the left.
   You are asked whether you want to run all or a subset of the test suite.

3. Click **Yes**, and then **Next** to run a subset of tests.

4. Select the tests you want to run from the displayed test tree, and then click **Done**.
   You can select entire branches of the test tree, or use `Ctrl+Click` or `Shift+Click` to select multiple

tests or ranges of tests, respectively.
After clicking **Done**, you are returned to the JavaTest main window.

5. Select **Run Tests**, then select **Start** to run the tests you selected.

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

1. Change to the directory containing the tests you want to run.
   For example, `<TS_HOME>/src/com/sun/ts/tests/samples`.

2. Start the test run by executing the following command:

```
ant runclient
```

The tests in `<TS_HOME>/src/com/sun/ts/tests/samples` and its subdirectories are run.

7.4.3 To Run a Subset of Tests in Batch Mode Based on Prior Result Status ^^^^^^^^^^^^^^^^^^^^^^^^^^^

You can run certain tests in batch mode based on the test's prior run status by specifying the `priorStatus` system property when invoking Ant.

Invoke `ant` with the `priorStatus` property.

The accepted values for the `priorStatus` property are any combination of the following:

- `fail`
- `pass`
- `error`
- `notRun`

For example, you could run all Jakarta EE 10 tests with a status of failed and error by invoking the following commands:

```
cd <TS_HOME>/src/com/sun/ts/tests/ejb
ant -DpriorStatus="fail,error" runclient
```

Note that multiple `priorStatus` values must be separated by commas.

7.5 Using Keywords to Test Required and Optional Technologies

The Jakarta EE TCK includes some tests that may be optional depending on your implementation. For example, certain technologies are now optional for implementations of the full Jakarta EE Platform. There are other technologies which are optional for Web Profile implementations, but may be implemented. If implemented, optional tests must be run and pass. There are two mechanisms in place in the TCK which control whether or not a given set of tests is run - the `javaee.level` property in the `ts.jte` file (see Section 7.5.1, "Setting the javaee.level Property") and keywords (see Section 7.5.2, "Using Keywords to Create Groups and Subsets of Tests").

7.5.1 Setting the javaee.level Property

The `ts.jte` file includes the `javaee.level` property. This property serves two purposes. First, it is used to determine whether the implementation under test is a Jakarta EE Full profile (full) or Jakarta EE Web profile (web). Either "full" or "web" must be specified in the list values. A setting of "full" instructs the test harness to deploy EAR files. A setting of "web" instructs the test harness to deploy WAR files. The `javaee.level` property is also used to help determine which APIs in the signature tests are to be tested. The comments that precede the property setting in the `ts.jte` file provide additional information about setting this property.

The default setting is as follows:

```
javaee.level=full
```

7.5.2 Using Keywords to Create Groups and Subsets of Tests

Each test in TCK has keywords associated with it. The keywords are used to create groups and subsets of tests. At test execution time, a user can tell the test harness to only run tests with or without certain keywords. This mechanism is used to select or omit testing on selected optional technologies. The "keywords" property can be set to a set of available keywords joined by "&" and/or "|".

To set the keywords system property at runtime, you must either pass it on the command line via `-Dkeywords=""` or in the JavaTest GUI, by opening the test suite and performing the following steps:

1. Select **View**, then select **Filters**, then select **CurrentConfiguration**.

2. Select **Configure**, then select **ChangeConfiguration**, then select **Keywords**.

3. In the Keywords dialog, select the Select **Tests that Match** check box, specify the desired keyword in the field, then click **Done**.

Only tests that have been tagged with that keyword will be enabled in the test tree.

The examples in the sections that follow show how to use keywords to run required technologies in both the Full and Web profile, run/omit running optional sets of tests in TCK, and run the Interoperability and Rebuildable tests in forward and reverse directions.

7.5.2.1 To Use Keywords to Run Required Technologies

Example 7-4 Running Tests for Required Technologies in the Full Profile

```
cd <TS_HOME>/src/com/sun/ts/tests
ant -Dkeywords=javaee runclient
```

Only tests that are required by the Full Profile will be run.

Example 7-5 Running Tests for All Required Technologies in the Web Profile

```
cd <TS_HOME>/src/com/sun/ts/tests
ant -Dkeywords=javaee_web_profile runclient
```

Only tests that are required by the Web Profile will be run.

Example 7-6 Running All Required Tests Except Connector Tests in the Full Profile

```
cd <TS_HOME>/src/com/sun/ts/tests
ant -Dkeywords="javaee & !connector" runclient
```

Example 7-7 Running All EJB Tests in the Full Profile

```
cd <TS_HOME>/src/com/sun/ts/tests
ant -Dkeywords=ejb runclient
```

Example 7-8 Running All EJB 3.2 Tests in the Full Profile

```
cd <TS_HOME>/src/com/sun/ts/tests
ant -Dkeywords=ejb32 runclient
```

Example 7-9 Running All EJB Tests in the Web Profile

```
cd <TS_HOME>/src/com/sun/ts/tests
ant -Dkeywords=ejb_web_profile runclient
```

7.5.2.2 To Use Keywords to Run Optional Technologies With the Full Profile

Keywords can be used to run subsets of tests from areas that are not required by the Jakarta EE 10 platform specification. Table 7-1 lists optional subsets of tests that can be run for the Full Profile and provides the technology-to-keyword mappings for each of the optional areas.

Table 7-1 Keyword to Technology Mappings for Full Profile Optional

Subsets

Technology	Keyword
EJB 1.x, CMP, BMP, entity beans	`ejb_1x_optional` or `javaee_optional`
EJB 2.x, CMP, BMP, entity beans	`ejb_2x_optional` or `javaee_optional`
EJBQL	`javaee_optional`
JAXR	`javaee_optional`

Example 7-10 Running Tests for All Optional Technologies in the Full Profile

```
cd <TS_HOME>/src/com/sun/ts/tests
ant -Dkeywords=javaee_optional runclient
```

Example 7-11 Running Jakarta Registries test stage is no longer supported

7.5.2.3 To Use Keywords to Run Optional Subsets of Tests With the Web Profile

Keywords can be used to run subsets of tests from additional areas that are not required by the Jakarta EE 10 Web Profile specification. For example, if your server implements the Jakarta EE 10 Web Profile and the Jakarta Connector Architecture 2.0 technology, set the keywords to `javaee_web_profile|connector_web_profile` to enable running tests for both areas. The command below shows how to specify these keywords to run the tests in both areas.

```
ant -Dkeywords="(javaee_web_profile|connector_web_profile) runclient
```

Table 7-2 lists optional subsets of tests that can be run for the Web Profile and provides the technology-to-keyword mappings for each of the optional areas.

Table 7-2 Keyword to Technology Mappings for Web Profile Optional

Subsets

Technology	Keyword
Jakarta Connectors	connector_web_profile
Jakarta Authorization (formerly JACC)	jacc_web_profile
Jakarta Authentication (formerly JASPIC)	jaspic_web_profile
Jakarta Mail (formerly JavaMail)	javamail_web_profile
Jakarta Registries (formerly JAXR)	jaxr_web_profile
Jakarta Messaging(formerly JMS)	jms_web_profile
XA	xa_web_profile

To add tests for other technologies, select the appropriate keyword from Table 7-2. This table provides a mapping of keywords to optional technologies (test directories) in the test suite and indicates optional test areas for the Jakarta EE 10 Web Profile.

Example 7-12 Running Tests for All Optional Technologies in the Web Profile

```
cd <TS_HOME>/src/com/sun/ts/tests
ant -Dkeywords=javaee_web_profile_optional runclient
```

Example 7-13 Running the Optional Jakarta Authorization and Authentication Tests With All Required Web Profile Tests

```
cd <TS_HOME>/src/com/sun/ts/tests
ant -Dkeywords="javaee_web_profile | jacc_web_profile | jaspic_web_profile" runclient
```

7.5.2.4 To Use Keywords to Run Optional Subsets for Jakarta Enterprise Beans Lite

Table 1-1 shows the TCK keywords you can use to test optional Jakarta Enterprise Beans (formerly EJB)

Lite components. Components denoted with an asterisk (*) are pruned components; components without an asterisk are not required by EJB Lite.

Table 7-3 TCK Keywords for Optional Jakarta Enterprise Beans Lite Components

Component	TCK Keyword
Message-Driven Beans	`ejb_mdb_optional`
1x CMP/BMP Entity Beans *	ejb_1x_optional
2x CMP/BMP Entity Beans, Remote/Home Component, Local/Home Component *	`ejb_2x_optional`
3x Remote	`ejb_3x_remote_optional`
EJB QL *	`ejb_ql_optional`
Persistent Timer Service	`ejb_persistent_timer_optional`
Remote asyncrhonous session bean	`ejb_remote_async_optional`
EJB Embeddable Container	`ejb_embeddable_optional`

Support for the following features has been made optional in this release:

- EJB 2.1 and earlier Entity Bean Component Contract for Container-Managed Persistence and Bean-Managed Persistence
- Client View of an EJB 2.1 and earlier Entity Bean
- EJB QL: Query Language for Container-Managed Persistence Query Methods

7.5.2.5 To Use Keywords to Run Tests in Selected Vehicles

The following vehicle keywords can be used to select or exclude the vehicles in which tests are run:

- connectorservlet_vehicle
- ejblitesecuredjsp_vehicle
- ejbliteservlet_vehicle
- ejbliteservlet2_vehicle
- jaspicservlet_vehicle
- pmservlet_vehicle
- puservlet_vehicle
- wsservlet_vehicle

- servlet_vehicle

- jsp_vehicle

- web_vehicle

- appclient_vehicle

- wsappclient_vehicle

- ejb_vehicle

- wsejb_vehicle

These vehicles are defined in the `<TS_HOME>/src/com/sun/ts/tests/common/vehicle` subdirectory structures.

Example 7-14 Running Tests in the Jakarta Enterprise Beans (EJB) Vehicle Only

```
ant -Dkeywords="ejb_vehicle"  runclient
```

Example 7-15 Running Tests in Vehicles Other Than the Jakarta Enterprise Beans Vehicle

```
ant -Dkeywords="!ejb_vehicle"  runclient
```

7.5.2.6 To Use Keywords to Run Tests in Forward and Reverse Directions

The `forward` and `reverse` keywords can be used to filter the interop and/or rebuildable tests during a selected test run when running tests in one of the following directories only:

```
<TS_HOME>/src/com/sun/ts/tests/jaxws
<TS_HOME>/src/com/sun/ts/tests/jws
<TS_HOME>/src/com/sun/ts/tests/interop
```

`Forward` tests are interop tests that run from the Vendor Implementation to the Compatible Implementation, as well as rebuildable tests that run only against the Vendor Implementation. Reverse tests (with test names ending in _reverse) are interop tests that run from the Compatible Implementation to the Vendor Implementation, as well as rebuildable tests that run only against the Compatible Implementation.

To set one of these keywords when running in command-line mode, set the appropriate keyword using the keyword system property.

Example 7-16 Running Tests in the Forward Direction

```
ant -Dkeywords=forward runclient
```

Example 7-17 Running Tests in the Reverse Direction

```
ant -Dkeywords=reverse runclient
```

To set one of these keywords in the Javatest GUI, select the Configure menu item, then select Change Configuration, then select Keywords, and set the appropriate keyword.

When one of these keywords has been set, executing tests in the directories above causes only those tests that match the keyword to be run. This can be useful when trying to debug failures with a particular test configuration. Note, however, for certification all tests in both directions must pass.

7.6 Running Interop or Jakarta XML Web Service Reverse Tests

If you are running Interop or XML Web Service reverse tests, which run against the Jakarta EE 10 CI, you must start the standalone deployment server in a separate shell on the same host as the TCK harness. The default deployment porting implementation goes through a standalone deployment server with a dedicated classpath. To start the standalone deployment server, change to the `<TS_HOME>/bin` directory and execute the `start.auto.deployment.server` Ant task.

7.7 Rebuilding Test Directories

The following directories require rebuilding, which is done by running the `configure.datasource.tests` Ant target:

- `com/sun/ts/tests/ejb30/lite/packaging/war/datasource`

- `com/sun/ts/tests/ejb30/assembly/appres`

- `com/sun/ts/tests/ejb30/misc/datasource`

When the `configure.datasource.tests` Ant target is run from any directory, it rebuilds these directories and any required subdirectories.

The `com/sun/ts/tests/jms/ee20/resourcedefs` directory must also be rebuilt. Run the `build.special.webservices.clients` Ant target to rebuild the tests in this directory.

The database properties in the TCK bundle are set to Derby database. If any other database is used, the `update.metadata.token.values` ant target needs to be executed for metadata-complete tests.

The following directories require rebuilding: `src\com\sun\ts\tests\appclient\deploy\metadatacomplete\testapp.`

This can be done by running the `update.metadata.token.values` Ant target.

7.8 Test Reports

A set of report files is created for every test run. These report files can be found in the report directory you specify. After a test run is completed, the JavaTest harness writes HTML reports for the test run. You can view these files in the JavaTest ReportBrowser when running in GUI mode, or in the web browser of your choice outside the JavaTest interface.

To see all of the HTML report files, enter the URL of the `report.html` file. This file is the root file that links to all of the other HTML reports.

The JavaTest harness also creates a `summary.txt` file in the report directory that you can open in any text editor. The `summary.txt` file contains a list of all tests that were run, their test results, and their status messages.

Although you can run the Ant report target from any test directory, its support is not guaranteed in the lower level directories. It is recommended that you always run the report target from `<TS_HOME>/bin`, from which reports are generated containing information about which tests were or were not run.

7.8.1 Creating Test Reports

7.8.1.1 To Create a Test Report in GUI Mode

1. From the JavaTest main menu, select **Report**, then select **Create Report**.
   You are prompted to specify a directory to use for your test reports.

2. Specify the directory you want to use for your reports, and then click **OK**.
   Use the **Filter** list to specify whether you want to generate reports for the current configuration, all tests, or a custom set of tests.
   You are asked whether you want to view report now.

3. Click **Yes** to display the new report in the JavaTest ReportBrowser.

7.8.1.2 To Create a Test Report in Command-Line Mode

Specify where you want to create the test report.

1. To specify the report directory from the command line at runtime, use:

```
ant report -Dreport.dir="report_dir"
```

   Reports are written for the last test run to the directory you specify.

2. To specify the default report directory, set the `report.dir` property in `<TS_HOME>/bin/ts.jte`.
   For example, `report.dir="/home/josephine/reports"`.

3. To disable reporting, set the `report.dir` property to `"none"`, either on the command line or in `ts.jte`.
   For example:

```
ant -Dreport.dir="none"
```

Troubleshooting

Although you can run the `report` Ant target from any test directory, its support is not guaranteed in the lower level directories. It is recommended that you always run the `report` target from `<TS_HOME)/bin`, from which reports are generated containing information about which tests were or were not run.cc

7.8.2 Viewing an Existing Test Report

7.8.2.1 To View an Existing Report in the JavaTest Report Browser

1. From the JavaTest main menu, select Report, then select Open Report.
   You are prompted to specify the directory containing the report you want to open.

2. Select the report directory you want to open, and then click Open.
   The selected report set is opened in the JavaTest Report Browser.

7.8.2.2 To View an Existing Report in a Web Browser

Use the Web browser of your choice to view the `report.html` file in the report directory you specified from the command line or in `ts.jte`.

The current report directory is displayed when you run the `report` target.

8 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. Note that most of these suggestions are only relevant when running the test harness in GUI mode. This is a dummy change and will be reverted.

This chapter includes the following topics:

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

8.1 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 JavaTest harness. These errors usually occur because the test environment is not properly configured.
- Failures: Tests that fail were executed but had failing results.

The Test Manager GUI provides you with a number of tools for effectively troubleshooting a test run. See the JavaTest User's Guide and JavaTest online help for detailed descriptions of the tools described in this chapter.

8.2 Test Tree

Use the test tree in the JavaTest GUI to identify specific folders and tests that had errors or failing results. Color codes are used to indicate status as follows:

- Green: Passed
- Blue: Test Error
- Red: Failed to pass test

- White: Test not run

- Gray: Test filtered out (not run)

8.3 Folder Information

Click a folder in the test tree in the JavaTest GUI to display its tabs.

Choose the Error and the Failed tabs to view the lists of all tests in and under a folder that were not successfully run. You can double-click a test in the lists to view its test information.

8.4 Test Information

To display information about a test in the JavaTest GUI, click its icon in the test tree or double-click its name in a folder status tab. The tab contains detailed information about the test run and, at the bottom of the window, a brief status message identifying the type of failure or error. This message may be sufficient for you to identify the cause of the error or failure.

If you need more information to identify the cause of the error or failure, use the following tabs listed in order of importance:

- Test Run Messages contains a Message list and a Message section that display the messages produced during the test run.

- Test Run Details contains a two-column table of name/value pairs recorded when the test was run.

- Configuration contains a two-column table of the test environment name/value pairs derived from the configuration data actually used to run the test.

> You can set `harness.log.traceflag=true` in `<TS_HOME>/bin/ts.jte` to get more debugging information. In a terminal window, you can also set an environment variable `HARNESS_DEBUG=true` to display more debugging information.

8.5 Report Files

Report files are another good source of troubleshooting information. You may view the individual test results of a batch run in the JavaTest Summary window, but there are also a wide range of HTML report files that you can view in the JavaTest ReportBrowser or in the external browser or your choice following a test run. See Test Reports for more information.

8.6 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. However, in the case of full-scale launching problems where no tests are actually processed, report files are usually not created (though sometimes a small `harness.trace` file in the report directory is written).

When aborting a test run, consider the following:

- If you abort a test run when running the JavaTest harness in GUI mode, the GUI tools automatically cleans up your environment for the next test run. This cleanup includes undeploying any components or applications that may deployed or registered with the Application Server.

- If you abort a test run in command-line mode (by pressing Ctrl+C), your environment might not be left in a clean state, causing potential failures in subsequent test runs. In such cases, you may need to perform the following procedure to restore your environment to a clean state.

To restore your environment after aborting a test run in command-line mode, perform these steps.

1. Log in to the Eclipse GlassFish 6.1 Application Server with the `asadmin` command.

2. List all registered components with the `asadmin list-components` command.

3. Undeploy any listed components related to your test run with the `asadmin undeploy` listed_component command.

9 Troubleshooting

This chapter explains how to debug test failures that you could encounter as you run the Jakarta Platform, Enterprise Edition Compatibility Test Suite.

10 Common TCK Problems and Resolutions

This section lists common problems that you may encounter as you run the Jakarta Platform, Enterprise Edition Test Compatibility Kit software on the Jakarta Platform, Enterprise Edition CI, Eclipse GlassFish 6.1. It also proposes resolutions for the problems, where applicable.

- Problem:
  The following exception may occur when a Jakarta EE 10 Platform TCK test tries to write a very long tracelog:

```
java.lang.StringIndexOutOfBoundsException: String
index out of range:
-13493
at java.lang.String.substring(String.java:1525)
at java.lang.String.substring(String.java:1492)
at javasoft.sqe.javatest.TestResult$Section
$WritableOutputBuffer.write(TestResult.java:650)
at java.io.Writer.write(Writer.java:153)
at java.io.PrintWriter.write(PrintWriter.java: 213)
at java.io.PrintWriter.write(PrintWriter.java: 229)
at java.io.PrintWriter.print(PrintWriter.java: 360)
at java.io.PrintWriter.println(PrintWriter.java:497)
at javasoft.sqe.javatest.lib.ProcessCommand
$StreamCopier.run(ProcessCommand.java:331)
```

The execution of the test will either fail or hang.
Resolution:
Set the `-Djavatest.maxOutputSize=`nnn system parameter in the `runclient` and/or `gui` targets in the `<TS_HOME>/bin/build.xml` file to a value that is higher than the default setting of `100,000` on the JavaTest VM. * Problem:
When you start the Jakarta Platform, Enterprise Edition CI, Eclipse GlassFish 6.1 on Windows by using the `javaee -verbose` command, the system may not find the specified path and could display one of the following errors:

```
"Verify that JAVA_HOME is set correctly"
"Verify that JAVAEE_HOME is set correctly"
```

Resolution:

Set `JAVA_HOME` to the path where the version of Java being used was installed and set `JAVAEE_HOME` to the location of the Jakarta Platform, Enterprise Edition installation directory. * Problem:

If the `cts.jar` and the `tsharness.jar` files are not loadable by the extension classloader of your Jakarta Platform, Enterprise Edition server, the following exception will be displayed in the window where the server was started when you attempt to run the tests:

```
java.lang.NoClassDefFoundError: com/sun/cts/util
RemoteLoggingInitException
```

Resolution:

Ensure that the `cts.jar` and `tsharness.jar` files can be loaded by the extension class loader of your Jakarta Platform, Enterprise Edition server.

9.2 Support

Jakarta EE is a community sponsored and community supported project. If you need additional assistance, you can reach out to the specific developer community. You will find the list of all Eclipse EE4J projects at `https://projects.eclipse.org/projects/ee4j`. All the sub-projects are listed. Each project page has details regarding how to contact thir developer community.

10 Building and Debugging Tests

For final certification and branding, all tests must be run through the JavaTest test harness. However, you can execute different Ant targets during your build and debug cycle. The following sections describe how to use Ant with the following targets to rebuild, list, and run tests:

- `runclient`

- `clean`

- `build`

- `ld`, `lld`, `lc`, `llc`, `pd`, `pc`

Implementers can only run the version of the tests provided with the CTS for certification, except in the case of rebuildable tests.

This chapter includes the following topics:

- Configuring Your Build Environment

- Building the Tests

- Running the Tests

- Listing the Contents of dist/classes Directories

- Debugging Service Tests

10.1 Configuring Your Build Environment

Complete the following steps to set up your environment to build, deploy, and run the TCK tests using Ant. The following example is for the Solaris platform:

1. Set the following environment variables in your shell environment to use the build infrastructure that comes with the TCK:

   ◦ `TS_HOME` to the directory in which the Jakarta EE 10 Platform TCK software is installed.

   ◦ `TS_HOME/bin` to your `PATH` in your command shell.

   ◦ C Shell:

```
setenv PATH ${TS_HOME}/bin:${PATH}
```

Bourne Shell:

```
PATH=${TS_HOME}/bin:${PATH}
export PATH
```

- `JAVA_HOME` to the directory in which the Java SE 8 or 11 software is installed.

- `JAVAEE_HOME` to the directory in which the Jakarta Platform, Enterprise Edition Compatible Implementation (CI) is installed.

   1. Unset `ANT_HOME`, if it is currently set in your environment.

   2. Change to the `<TS_HOME>/bin` directory and verify that the `ts.jte` file has the following properties set:

- `webserver.home`: the directory in which the CI Web Server is installed

- `webserver.host`: the host on which the CI Web server is running

- `webserver.port`: the port on which the CI Web server is running

- `javaee.home.ri`: the directory in which the Jakarta Platform, Enterprise Edition CI is installed for reference to the packager tool used by the build infrastructure

- `ts.classpath`: required classes needed for building/running the TCK

10.2 Building the Tests

To build the Jakarta EE 10 Platform TCK tests using Ant, complete the following steps:

   1. To build a single test directory, type the following:

```
cd <TS_HOME>/src/com/sun/ts/tests/test_dir
ant clean build
```

This cleans and builds the tests in the directory specified for test_dir. 2. To list the classes directory for this test that was built, type the following:

```
ant lc
```

or

```
ant llc
```

   1. To list the distribution directory of archives for this test that was built, type the following:

```
ant pd
```

or

```
ant pc
```

10.3 Running the Tests

To run the Jakarta EE 10 Platform TCK tests using Ant, use one of the following procedures.

10.3.1 To Run a Single Test Directory

To run a single test directory, type the following:

```
cd <TS_HOME>/src/com/sun/ts/tests/test_dir
ant runclient
```

This runs all tests in test_dir.

10.3.2 To Run a Single Test Within a Test Directory

To run a single test within a test directory, type the following:

```
cd <TS_HOME>/src/com/sun/ts/tests/test_dir
ant runclient -Dtest=test_name
```

This runs only the test_name in the test_dir test directory. To show all the tests that can be run from a particular test directory, change to the directory and execute the `list.tests` Ant task. The actual test name displays to the right of the pound sign (#), which follows the fully qualified name of the client class.

10.4 Listing the Contents of dist/classes Directories

You can use various Ant targets to list the contents of corresponding `dist/classes` directories from the `src` directory without leaving the `src` directory. All listings are sorted by modification time, with the most recent modification listed first. Output is redirected to `more`. The format may vary on Windows and Unix. Ant does not support changing directory into the `dist/classes` directories, but you can copy and paste the first line of the output, which is the target path.

The Ant list targets are as follows:

- `ld`: Lists the contents of the current test's dist directory
- `lld`: Provides a long listing of the contents of the current test's dist directory
- `lc`: Lists the contents of the current test's classes directory
- `llc`: Provides a long listing of the contents of the current test's classes directory
- `pd`: Starts a new shell placed into the current test's dist directory
- `pc`: Starts a new shell placed into the current test's classes directory

If you run these targets in a directory that is not under the `src` directory, they will list the contents of the current directory.

> ℹ️ `pc`, `lc`, and `llc` also support the `-Dbuild.vi` property for listing the rebuildable tests. The rebuildable tests are located under `<TS_HOME>/classes_vi_built` instead of `<TS_HOME>/classes`.

The following listing shows sample output for the Ant `lc` target.

```
cd $TS_HOME/src/com/sun/ts/tests/samples/ejb/ee/simpleHello
ant lc
<TS_HOME>/classes/com/sun/ts/tests/samples/ejb/ee/simpleHello


----------------------------------------------------------------
Hello.class
HelloClient.class
HelloEJB.class
HelloHome.class

ant -Dbuild.vi=true lc
<TS_HOME>/classes_vi_built/com/sun/ts/tests/samples/ejb/ee/simpleHello


----------------------------------------------------------------
Hello.class
HelloClient.class
HelloEJB.class
HelloHome.class
```

10.5 Debugging Service Tests

The Jakarta EE 10 Platform TCK service tests test the compatibility of the Jakarta Platform, Enterprise Edition Service APIs: Jakarta Mail, JDBC, Jakarta Messaging, Jakarta Transactions, Jakarta XML Web Services, Jakarta Web Services Metadata, Jakarta Annotations. The test suite contains sets of tests that the JavaTest harness, in conjunction with the Jakarta EE 10 Platform TCK harness extensions, runs from different Jakarta Platform, Enterprise Edition containers (Jakarta Enterprise Beans, Jakarta Server Pages, Jakarta Servlet, and application client). The test suite wraps each of these tests inside generic components, called vehicles. Each Jakarta EE 10 Platform TCK service test has been set up to run in a default set of vehicles. Each technology's specification determines this set. When run as part of the certification process, all service API tests must pass in their default vehicle set.

Refer to the `<TS_HOME>/src/vehicle.properties` file to for a list the default vehicle sets for the Jakarta EE 10 Platform TCK service API tests.

To help you debug service API tests, the test suite provides a mechanism that allows for fine-grained control over which tests you can run in specific vehicles. When you override the default vehicle set for a particular set of service tests, the new set of vehicles must be a subset of the valid vehicle set for that set of tests. If the new set is not a subset of the default set, the test suite will use the default set.

> You can only use this mechanism for debugging purposes. For certification, you must run using the default set of vehicles.

10.5.1 Examples

Example 10-1 Restricting the JDBC Test Run

To restrict the JDBC test run to the servlet and Jakarta Server Pages vehicles only, set the following system property in the `<TS_HOME>/bin/build.xml` file for the Ant `gui` or `runclient` targets:

```
<sysproperty key="tests_jdbc_ee.service_eetest.vehicles"
  value="servlet jsp"/>
```

Before you run the test or tests, you should temporarily rename the file `<TS_HOME>/src/testsuite.jtd`.

Note that you must remove these properties before you run the Jakarta EE 10 TCK test suite for certification.

10.5.2 Obtaining Additional Debugging Information

When running the JavaTest harness in command-line mode, you can obtain additional debugging information by setting the `HARNESS_DEBUG` environment variable, as follows:

```
setenv HARNESS_DEBUG=true
```

Subsequent runs with the Ant `runclient` command generate additional debugging information.

You can also generate additional test run information by seting the `<TS_HOME>/bin/ts.jte` `harness.log.traceflag` property as follows:

```
harness.log.traceflag=true
```

11 Implementing the Porting Package

Some functionality in the Jakarta Platform, Enterprise Edition platform is not completely specified by an API. To handle this situation, the Jakarta EE test suite defines a set of interfaces in the `com.sun.cts.porting` package, which serve to abstract any implementation-specific code. The CTS also provides implementations of these interfaces to work with the Jakarta Platform, Enterprise Edition CI.

You must create your own implementations of the porting package interfaces to work with your particular Jakarta Platform, Enterprise Edition server environment. You also need to create a deployment plan for each deployable component (EAR, EJB JAR, WAR, and RAR files) in the test suite as defined by the Jakarta Platform, Enterprise Edition platform.

11.1 Overview

The Jakarta Platform, Enterprise Edition CI uses a set of module-name-with-extension`.sun-standard-deployment-desc-component-prefix`.xml` files that are associated with each deployable component. A CTS `DeploymentInfo` object parses the contents of several runtime XML files: `sun-application_1_4-0.xml`, `sun-application-client_1_4-0.xml`, `sun-ejb-jar_2_1-0.xml`, and `sun-web-app_2_4-0.xml`, and makes their content available to create deployment plans by means of the `getDeploymentPlan()` method.

To use specific implementations of these classes, you simply modify the following entries in the `porting class .1` section of the `ts.jte` environment file to identify the fully-qualified class names:

```
porting.ts.deploy2.class.1=[vendor-deployment-class]
porting.ts.login.class.1=[vendor-login-class]
porting.ts.url.class.1=[vendor-url-class]
porting.ts.jms.class.1=[vendor-jms-class]
porting.ts.HttpsURLConnection.class.1=[vendor-httpsURLConnection-class]
```

The `<TS_HOME>/src/com/sun/ts/lib/porting` directory contains the interfaces and `Factory` classes for the porting package.

> You must not modify any of the CTS Release 10 TCK source code. Create your own implementations of these interfaces and point to them in the appropriate section of the `ts.jte` file.

Note the change to the deployment porting property above. It has changed to be `deploy2`. This is because there is a new deployment porting interface because of the standardization of a deployment API in Java Platform, Enterprise Edition. Any functionality that is still not addressed by this API is part of the new interface com.sun.ts.lib.porting.TSDeploymentInterface2.

Make sure your porting class implementations meet the following requirements:

- Implement the following porting interfaces:

  - `TSDeploymentInterface2`

  - `TSLoginContextInterface`

  - `TSURLInterface`

  - `TSJMSAdminInterface`

  - `TSHttpsURLConnectionInterface`

- Include the implementation of the previous interfaces in the classpaths of JavaTest, the test clients, and the test server components:

  - In the `ts.harness.classpath` property in the `<TS_HOME>/bin/ts.jte` file

  - In the `CLASSPATH` variable of the `command.testExecute` and `command.testExecuteAppClient` properties in the `ts.jte` file

  - In the classpath of your Jakarta Platform, Enterprise Edition server

Note that because the JavaTest VM calls certain classes in the CTS porting package directly, porting class implementations are not permitted to exit the VM (for example, by using the `System.exit` call).

11.2 Porting Package APIs

The following sections describe the API in the Jakarta EE 10 CTS porting package. The implementation classes used with the Jakarta Platform, Enterprise Edition CI are located in the `<TS_HOME>/src/com/sun/ts/lib/implementation/sun/javaee` directory. You are encouraged to examine these implementations before you create your own.

Detailed API documentation for the porting package interfaces is available in the `<TS_HOME>/docs/api` directory. The API included in this section are:

- TSDeploymentInterface2 is removed

- Ant-Based Deployment Interface

- TSJMSAdminInterface

- TSLoginContextInterface

- TSURLInterface

- TSHttpsURLConnectionInterface

11.2.1 TSDeploymentInterface2 is removed

TSDeploymentInterface2 + TSDeployment2 are no longer supported since the Jakarta Deployment spec

is removed. Please use the `TSDeploymentInterface` instead. The interface `TSDeploymentInterface`, which uses only the standard Deployment APIs defined by the Jakarta Platform, Enterprise Edition platform. The following properties are still in the `ts.jte` file to reflect this and should not be changed:

11.2.2 Ant-Based Deployment Interface

In addition to the Java-based deployment porting interfaces, Jakarta EE 10 CTS introduces an Ant-based porting interface as well. The Java-based interface is still used for deployment/undeployment during test runs. The Ant-based interface is used when you want to only deploy/undeploy archives associated with a subdirectory of tests. The Ant-based deployment interface is used by the following:

- The `build.special.webservices.clients` target in the `${ts.home}/bin/build.xml` file
  This target deploys archives to your server implementation and then builds the client classes that use those archives. You must run this target before you run the tests under the `${ts.home}/src/com/sun/ts/tests/webservices12/specialcases` directory.

- The `deploy` and `undeploy` targets in each test subdirectory under the `${ts.home}/src/com/sun/ts/tests` directory
  To use these targets, which are useful for debugging, you must provide an Ant-based deployment implementation.

11.2.2.1 Creating Your Own Ant-based Deployment Implementation

The Ant-based deployment implementation for the Jakarta EE 10 CI is under `${ts.home}/bin/xml/impl/glassfish` directory. To create your own implementation, create a `deploy.xml` file under the `${ts.home}/bin/xml/impl/<vendor-name>` directory. Within the file, create and implement the -deploy and -undeploy targets.

See `${ts.home}/bin/xml/impl/glassfish/deploy.xml` to see how these targets are implemented for the Jakarta EE 10 CI.

> There is also a Java-based implementation of TSDeploymentInterface (`com.sun.ts.lib.implementation.sun.javaee.glassfish.AutoDeployment`). This implementation, which leverages the Jakarta EE 10 CI implementation of the Ant-based deployment interface, calls the Ant targets programmatically.

11.2.3 TSJMSAdminInterface

Jakarta Messaging-administered objects are implementation-specific. For this reason, the creation of connection factories and destination objects have been set up as part of the porting package. Each Jakarta Platform, Enterprise Edition implementation must provide an implementation of the

`TSJMSAdminInterface` to support their own connection factory, topic/queue creation/deletion semantics.

The `TSJMSAdmin` class acts as a `Factory` object for creating concrete implementations of `TSJMSAdminInterface`. The concrete implementations are specified by the `porting.ts.jms.class.1` and `porting.ts.jms.class.2` properties in the `ts.jte` file.

If you wish to create the Jakarta Messaging-administered objects prior to executing any tests, you may use the default implementation of `TSJMSAdminInterface`, `SunRIJMSAdmin.java`, which provides a null implementation. In the case of the Jakarta Platform, Enterprise Edition CI Eclipse GlassFish 6.1, the Jakarta Messaging administered objects are created during the execution of the `config.vi` Ant target.

There are two types of Jakarta Messaging-administered objects:

1. A `ConnectionFactory`, which a client uses to create a connection with a JMS provider

2. A `Destination`, which a client uses to specify the destination of messages it sends and the source of messages it receives

11.2.4 TSLoginContextInterface

The `TSLoginContext` class acts as a `Factory` object for creating concrete implementations of `TSLoginContextInterface`. The concrete implementations are specified by the `porting.ts.login.class.1` property in the `ts.jte` file. This class is used to enable a program to login as a specific user, using the semantics of the Jakarta Platform, Enterprise Edition CI. The certificate necessary for certificate-based login is retrieved. The keystore file and keystore password from the properties that are specified in the `ts.jte` file are used.

11.2.5 TSURLInterface

The `TSURL` class acts as a `Factory` object for creating concrete implementations of `TSURLInterface`. The concrete implementations are specified by the `porting.ts.url.class.1` property in the `ts.jte` file. Each Jakarta Platform, Enterprise Edition implementation must provide an implementation of the `TSURLInterface` to support obtaining URL strings that are used to access a selected Web component. This implementation can be replaced if a Jakarta Platform, Enterprise Edition server implementation requires URLs to be created in a different manner. In most Jakarta Platform, Enterprise Edition environments, the default implementation of this class can be used.

11.2.6 TSHttpsURLConnectionInterface

The `TSHttpsURLConnection` class acts as a `Factory` object for creating concrete implementations of `TSHttpsURLConnectionInterface`. The concrete implementations are specified by the

`porting.ts.HttpsURLConnection.class.1` and `.2` properties in the `ts.jte` file.

You must provide an implementation of `TSHttpsURLConnectionInterface` to support the class `HttpsURLConnection`.

> The `SunRIHttpsURLConnection` implementation class uses `HttpsURLConnection` from Java SE 8.

A Common Applications Deployment

TODO: figure out if Common Applications Deployment tests should be removed since they cannot rely on (removed) Jakarta Deployment in EE 9.

Some tests in the test suite require the deployment of additional applications, components, or resource archives that are located in directories other than the test's directory. When the test harness encounters a test that requires these additional applications, components, or resource archives, they are passed to the `TSDeploymentInterface2` implementation to be deployed. Because these applications can be shared by tests in different test directories, they are called common applications.

Table A-1 lists the test directories and the directories that contain the common applications that are required by the test directories.

Table A-1 Required Common Applications

Directory Under `com/sun/ts/tests`	Directory Under `com/sun/ts/tests` **With Associated Common Applications**
`ejb/ee/tx/session`	`ejb/ee/tx/txbean`
`ejb/ee/tx/entity/bmp`	`ejb/ee/tx/txEbean`
`ejb/ee/tx/entity/cmp`	`ejb/ee/tx/txECMPbean`
`ejb/ee/tx/entity/pm`	`ejb/ee/tx/txEPMbean`
`connector/ee/localTx/msginflow`	`common/connector/whitebox`
`connector/ee/mdb`	`connector/ee/localTx`
`common/connector/whitebox`	`connector/ee/noTx`
`common/connector/whitebox`	`connector/ee/xa`
`common/connector/whitebox`	`connector/ee/connManager`
`common/connector/whitebox`	`xa/ee`
`compat13/connector/localTx`	`compat13/connector/whitebox`
`compat13/connector/noTx`	`compat13/connector/whitebox`
`compat13/connector/xa`	`compat13/connector/whitebox`
`interop/tx/session`	`interop/tx/txbean`
`interop/tx/entity`	`interop/tx/txEbean`
`interop/tx/webclient`	`interop/tx/txbean`
`ejb/ee/pm/ejbql`	`ejb/ee/pm/ejbql/schema`
`ejb/ee/tx/session/stateful/bm/TxMDBMS_Direct`	`ejb/ee/tx/session/stateful/bm/TxMDBMSBeans/BeanA`
`ejb/ee/tx/session/stateful/bm/TxMDBMSBeans/BeanB`	`ejb/ee/tx/session/stateful/bm/TxMDBMSBeans/BeanC`
`tests/ejb/ee/tx/session/stateful/bm/TxMDBMS_Indirect`	`ejb/ee/tx/session/stateful/bm/TxMDBMSBeans/BeanA`

Directory Under `com/sun/ts/tests`	**Directory Under** `com/sun/ts/tests` **With Associated Common Applications**
`ejb/ee/tx/session/stateful/bm/TxMDBMSBeans/BeanB`	`ejb/ee/tx/session/stateful/bm/TxMDBMSBeans/BeanC`
`ejb/ee/tx/session/stateful/bm/TxMDBSS_Direct`	`ejb/ee/tx/session/stateful/bm/TxMDBSSBeans/BeanA`
`ejb/ee/tx/session/stateful/bm/TxMDBSSBeans/BeanB`	`tests/ejb/ee/tx/session/stateful/bm/TxMDBSSBeans/BeanC`
`ejb/ee/tx/session/stateful/bm/TxMDBSS_Indirect`	`ejb/ee/tx/session/stateful/bm/TxMDBSSBeans/BeanA`
`ejb/ee/tx/session/stateful/bm/TxMDBSSBeans/BeanB`	`ejb/ee/tx/session/stateful/bm/TxMDBSSBeans/BeanC`

B Jakarta Authentication Technology Notes and Files

The Jakarta Authentication (formerly jaspic) technology tests are used to verify the compatibility of an implementer's implementation of the Jakarta Authentication 2.0 specification.

This appendix provides information about the following topics:

- Jakarta Authentication 2.0 Technology Overview

- Jakarta Authentication TSSV Files

You run the `ant enable.jaspic` command to configure the Jakarta EE 10 CI to run the Jakarta Authentication tests. After running the Jakarta Authentication tests, you run the `ant disable.jaspic` command to restore the Jakarta EE 10 CI to the state it was in before you configured it for running the Jakarta Authentication tests. This is required because Jakarta Authentication replaces some of the Jakarta EE 10 CI's default system security components with TCK security components. If this change is not reverted after the tests have been run, the CI's system security could be left in a partially working state. The TCK security `AuthConfigFactory` and `AuthConfigProvider`(s) are designed for compatibility testing, not the functional completeness that one expects from the Jakarta EE 10 CI.

B.1 Jakarta Authentication 2.0 Technology Overview

The Jakarta Authentication 2.0 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).

Jakarta EE 10 Platform TCK uses a Test Suite SPI Verifier (TSSV) to verify whether the vendor's message processing runtimes invoke the right SPI in the right order.

The TSSV includes test suite implementations of:

- `AuthConfigFactory`

- `AuthConfigProvider`

- `AuthConfig(Client & Server)`

- `AuthContext(client & Server)`

- `AuthenticationModules(Client & Server)`

The TSSV gets loaded into vendor's message processing runtime using one of the following two ways as defined by the Jakarta Authentication 2.0 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 `registerConfigProvider()` method in 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

> **ℹ** For the Jakarta EE 10 Platform TCK, we register more than one provider in 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 the TSSV implementations, and the runtime is analyzed to see whether it invokes the right SPIs at the right time.

The 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 Jakarta 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.

B.2 Jakarta Authentication TSSV Files

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

B.2.1 tssv.jar file

The `tssv.jar` file contains classes necessary for populating your implementation with a TCK AuthConfigFactory (ACF) as well as information used to register TCK 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 your implementation's runtime during startup.

B.2.2 ProviderConfiguration.xml file

The format of the `ProviderConfiguration.xml` file 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 your 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 altered.

The value of the `<app-context-id>` element in the `ProviderConfiguration.xml` file should reflect what your 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 you may need to alter the value of the `<app-context-id>` element as previously described to accommodate the implementation under test. You need to find the correct application context identifier for your implementation.

You 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 and the `HARNESS_DEBUG` environment variable to `true`. If both of these are set, application context identifier information should appear in the debug output.

B.2.3 provider-configuration.dtd file

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

C Configuring Your Backend Database

This appendix explains how to configure a backend database to use with a Jakarta Platform, Enterprise Edition server being tested against the Jakarta EE 10 TCK.

The topics included in this appendix are as follows:

- Overview
- The init.<database> Ant Target
- Database Properties in ts.jte
- Database DDL and DML Files
- CMP Table Creation

C.1 Overview

All Jakarta Platform, Enterprise Edition servers tested against the Jakarta EE 10 TCK must be configured with a database and JDBC 4.1-compliant drivers. Note that the Jakarta Platform, Enterprise Edition CI, Eclipse GlassFish 6.1 includes the Apache Derby database.

To perform interoperability testing, you need to configure two Jakarta Platform, Enterprise Edition servers and two databases, one of which must be the Jakarta Platform, Enterprise Edition RI with the bundled Apache Derby database. See Jakarta Platform, Enterprise Edition Server Configuration Scenarios for more information.

For the purposes of Jakarta EE 10 Platform TCK testing, all database configuration properties required by the TCK are made in the `<TS_HOME>/bin/ts.jte` file. The TCK `init.<`database> Ant target uses the properties you set in `ts.jte` to generate one or more SQL statement files that are in turn used create and populate database tables and configure procedures required by the TCK.

The database configuration process comprises four general steps:

1. Set database-related properties in the `<TS_HOME>/bin/ts.jte` file.
2. Configure your Jakarta Platform, Enterprise Edition server implementation for your database and for TCK.
3. Start your database.
4. Run the `init.`<database> Ant target to initialize your database for TCK.

The procedure for configuring your Jakarta Platform, Enterprise Edition server for your database is described in Configuring a Jakarta EE 10 Server. The final step, initializing your database for TCK by running `init.<`database> target, is explained more in the next section.

C.2 The init.<database> Ant Target

Before your Jakarta Platform, Enterprise Edition server database can be tested against the Jakarta EE 10 Platform TCK, the database must be initialized for TCK by means of the Ant `init.<`database> `target. For example, the `init.javadb` Ant task is used to initialize the Apache Derby database for TCK.

This Ant target references database properties in `ts.jte` file and database-specific DDL and DML files to generate SQL statement files that are read by the Jakarta EE 10 Platform TCK when you start the test suite. The DDL and DML files are described later in this appendix, in Database DDL and DML Files.

The Jakarta EE 10 Platform TCK includes the following database-specific Ant targets:

- `init.cloudscape`
- `init.db2`
- `init.oracle`
- `init.oracleDD`
- `init.oracleInet`
- `init.derby`
- `init.javadb`
- `init.sybase`
- `init.sybaseInet`
- `init.mssqlserver`
- `init.mssqlserverInet`
- `init.mssqlserverDD`

Each Ant target uses a database-specific JDBC driver to configure a backend for a specific database; for example, OracleInet/Oracle Inet driver; OracleDD/Oracle DataDirect driver. These targets are configured in the `<TS_HOME>/xml/initdb.xml` file.

C.3 Database Properties in ts.jte

Listed below are the names and descriptions for the database properties you need to set for TCK testing.

Note that some properties take the form property`.ri`. In all cases, properties with an `.ri` suffix are used for interoperability testing only. In such cases, the property value applies to the Jakarta Platform,

Enterprise Edition VI server (the server you want to test) and the property`.ri` value applies to the Jakarta Platform, Enterprise Edition CI, Eclipse GlassFish 6.1 server. For example:

```
db.dml.file=VI_DML_filename
db.dml.file.ri=RI_DML_filename
```

The property`.ri` properties are only used in two-server configurations; that is, when you are performing interoperability tests.

Table D-1 ts.jte Database Properties

Property	Description
database`.classes`	`CLASSPATH` to JDBC driver classes.
database`.dataSource`	DataSource driver.
database`.dbName`	Database Name.
database`.driver`	DriverManager driver.
database`.password`	User password configured.
database`.poolName`	Name of pool configured in the RI (do not change!).
database`.port`	Database Server port.
database`.properties`	Additional properties required by the defined data source for each driver configuration in `ts.jte`. You should not need to modify this property.
database`.server`	Database Server.
database`.url`	URL for the TCK database; the `dbName`, `server`, and `port` properties are automatically substituted in to build the correct URL. You should never need to modify this property.
database`.user`	User ID configured.
`create.cmp.tables`	When set to `false`, the application server is responsible for creating CMP tables at deployment of the EJB/EAR. When set to `true`, `init.`<datbase> creates the tables used by CMP EJBs. The SQL for the CMP tables are contained in `<TS_HOME>/`database`/sql/`database`.ddl.cmp.sql`` and `<TS_HOME>/`database`/sql/`database`.ddl.interop.sql``.
`db.dml.file`	Tells `init.`database which DML file to use for the VI database; for example, `db.dml.file=${javadb.dml.file}`.

Property	Description
`db.dml.file.ri`	Tells `init.`database which DML file to use for the RI database; for example, `db.dml.file=${javadb.dml.file}`.
`jdbc.lib.class.path`	Used by the database`.classes` properties to point to the location of the JDBC drivers.
`jdbc.poolName`	Configures the connection pool that will be used in the TCK test run; for example, `jdbc.poolName=${javadb.poolName}`. Set this property when running against the RI if using a database other than Apache Derby.
`password1`	Password for the JDBC/DB1 resource; for example, `password1=${javadb.passwd}`.
`password2`	Password for the JDBC/DB2 resource; for example, `password2=${javadb.passwd}`.
`password3`	Password for the JDBC/DBTimer resource; for example, `password3=${javadb.passwd}`.
`user1`	User name for the JDBC/DB1 resource; for example, `user1=${javadb.user}`.
`user2`	User name for the JDBC/DB2 resource; for example, `user2=${javadb.user}`.
`user3`	User name for the JDBC/DBTimer resource; for example, `user3=${javadb.user}`.

C.4 Database DDL and DML Files

For each supported database type, the Jakarta EE 10 Platform TCK includes a set of DDL and DML files in subdirectories off the `<TS_HOME>/sql` directory. The `config.vi` and `config.ri` targets use two `ts.jte` properties, `db.dml.file` and `db.dml.file.ri` (interop only), to determine the database type, and hence which database-specific DML files to copy as `<TS_HOME>/bin/tssql.stmt` and `tssql.stmt.ri` (for interop) files.

The `tssql.stmt` and `tssql.stmt.ri` files contain directives for configuring and populating database tables as required by the TCK tests, and for defining any required primary or foreign key constraints and database-specific conmand line terminators.

In addition to the database-specific DML files, the Jakarta EE 10 Platform TCK includes database-specific DDL files, also in subdirectories off `<TS_HOME>/sql`. These DDL files are used by the `init.`database target to create and drop database tables and procedures required by the TCK.

The SQL statements in the `tssql.stmt` and `tssql.stmt.ri` files are read as requested by individual TCK

tests, which use the statements to locate required DML files.

The DDL and DML files are as follows:

- database`.ddl.sql`: DDL for BMP, Session Beans

- database`.ddl.sprocs.sql`: DDL for creating stored procedures

- database`.ddl.cmp.sql`: DDL for CMP Entity Beans

- database`.ddl.interop.sql`: DDL for interop tests

- database`.dml.sql`: DML used during test runs

Each DDL command in each `<TS_HOME>/sql/`database is terminated with an ending delimiter. The delimiter for each database is defined in the `<TS_HOME>/bin/xml/initdb.xml` file. If your configuration requires the use of a database other than the databases that `initdb.xml` currently supports, you may modify `initdb.xml` to include a target to configure the database that you are using.

An example of the syntax for a database target in `initdb.xml` is shown below:

```
<target name="init.sybase">
  <antcall target="configure.backend">
      <param name="db.driver" value="${sybase.driver}"/>
      <param name="db.url" value="${sybase.url}"/>
      <param name="db.user" value="${sybase.user}"/>
      <param name="db.password" value="${sybase.passwd}"/>
      <param name="db.classpath" value="${sybase.classes}"/>
      <param name="db.delimiter" value="!"/>
      <param name="db.name" value="sybase" />
  </antcall>
</target>
```

The database`.name` property should be added to your `ts.jte` file. The `db.name` property is the name of a subdirectory in `<TS_HOME>/sql`. After updating `initdb.xml`, you invoke the new target with:

```
ant -f <TS_HOME>/bin/xml/initdb.xml init.databasename
```

C.5 CMP Table Creation

If the application server under test does not provide an option to automatically create tables used by CMP Entity EJBs, the needed SQL is provided in `<TS_HOME>/sql/`database/database.cmp.sql`.

Setting the `ts.jte` property `create.cmp.tables=true` instructs the `init.`databasename target to create the tables defined in the `<TS_HOME>/sql/`database/database.cmp.sql` file.

If you set `create.cmp.tables=false` in the `ts.jte` file, it is expected that you will create the necessary CMP tables at deployment time.

D EJBQL Schema

The Jakarta Enterprise Beans, EJB-QL tests perform queries against a CMP 2.0 abstract persistence model that you deploy before you start the test runs.

Section 9.3.5, "EJB QL and SQL," in the EJB 3.1 Specification (`http://jcp.org/en/jsr/detail?id=318`) contains a sample mapping that shows how the Jakarta Platform, Enterprise Edition CI translates EJB QL to SQL, which helps to clarify the EJB QL semantics.

This appendix includes information about the following topics:

- Persistence Schema Relationships
- SQL Statements for CMP 1.1 Finders

D.1 Persistence Schema Relationships

The figure, Figure E-1, below, contains detailed information about the persistence schema relationships.

Figure E-1 Persistence Schema Relationships

Note: EJB in the figure above is a references to Jakarta Enterprise Beans.

AliasEJB

```
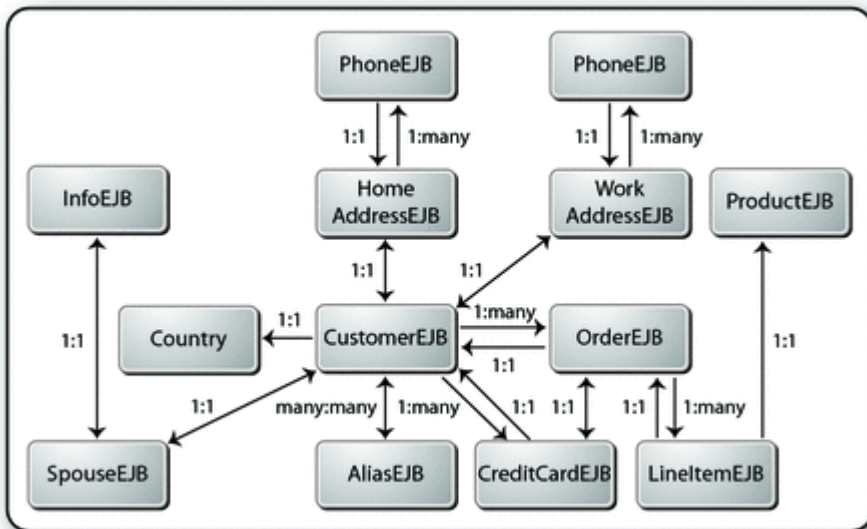String id;(pk)(cmp)
String alias;(cmp)
Customer
customerNoop;(cmr)
Collection
customersNoop;(cmr)
Collection customers;(cmr)
```

Country: DVC

```
String name;(cmp)
String code;(cmp)
```

AddressEJB

```
String id;(pk)(cmp)
String street;(cmp)
String city;(cmp)
String state;(cmp)
String zip;(cmp)
Collection phones;(cmr)
```

ProductEJB

```
String id;(pk)(cmp)
String name;(cmp)
float price;(cmp)
int quantity;(cmp)
long partNumber;(cmp)
```

CustomerEJB

```
String id;(pk)(cmp)
String name;(cmp)
Country country;(cmp)
AddressLocal home;(cmr)
AddressLocal work;(cmr)
Collection
creditCards;(cmr)
Collection orders;(cmr)
Collection aliases;(cmr)
SpouseLocal spouse;(cmr)
```

CreditCardEJB

```
String id;(pk)(cmp)
String type;(cmp)
String expires;(cmp)
boolean approved;(cmp)
String number;(cmp)
OrderLocal order;(cmr)
CustomerLocal
customer;(cmr)
double balance;(cmp)
```

InfoEJB

```
String id;(pk)(cmp)
String street;(cmp)
String city;(cmp)
String state;(cmp)
String zip;(cmp)
SpouseLocal spouse;(cmr)
```

OrderEJB

```
String id;(pk)(cmp)
float totalPrice;(cmp)
CustomerLocal
customer;(cmr)
LineItemLocalsampleLineIt
em;(cmr)
Collection
lineItems;(cmr)
CreditCardLocal
creditCard;(cmr)
```

LineItemEJB

```
String id;(pk)(cmp)
int quantity;(cmp)
OrderLocal order;(cmr)
ProductLocal
product;(cmr)
```

PhoneEJB

```
String id;(pk)(cmp)
String area:(cmp)
String number;(cmp)
AddressLocal
address;(cmr)
```

SpouseEJB

```
String id;(pk)(cmp)
String firstName;(cmp)
String maidenName;(cmp)
String lastName;(cmp)
String
SocialSecurityNumber(cmp)
;
InfoLocal info;(cmr)
CustomerLocal
customer;(cmr)
```

D.2 SQL Statements for CMP 1.1 Finders

Listed below are the SQL statements used for CMP 1.1 finders in:

- ejb/ee/bb/entity/cmp/clientviewtest

- interop/ejb/entity/cmp/clientviewtest

- ejb/ee/bb/entity/cmp/complexpktest

- ejb/ee/tx/txECMPbean

D.2.1 ejb/ee/bb/entity/cmp/clientviewtest, interop/ejb/entity/cmp/clientviewtest

```
<method-name>findWithinPrimaryKeyRange</method-name>
<sql>SELECT "KEY_ID" FROM "TestBeanEJBTable" WHERE "KEY_ID" BETWEEN ?1 AND ?2</sql>
<method-name>findWithinPriceRange</method-name>
<sql>SELECT "KEY_ID" FROM "TestBeanEJBTable" WHERE "PRICE" BETWEEN ?1 AND ?2</sql>
<method-name>findByName</method-name>
<sql>SELECT "KEY_ID" FROM "TestBeanEJBTable" WHERE "BRAND_NAME" = ?1</sql>
<method-name>findAllBeans</method-name>
<sql>SELECT "KEY_ID" FROM "TestBeanEJBTable"</sql>
<method-name>findByPrice</method-name>
<sql>SELECT "KEY_ID" FROM "TestBeanEJBTable" WHERE "PRICE" = ?1</sql>
<method-name>findByNameSingle</method-name>
<sql>SELECT "KEY_ID" FROM "TestBeanEJBTable" WHERE "BRAND_NAME" = ?1</sql>
```

D.2.2 ejb/ee/bb/entity/cmp/complexpktest

```
<method-name>findByPrice</method-name>
<sql>SELECT "BRAND_NAME", "ID" FROM "TestBeanEJBTable" WHERE "PRICE" = ?1</sql>
<method-name>findById</method-name>
<sql>SELECT "BRAND_NAME", "ID" FROM "TestBeanEJBTable" WHERE "ID" = ?1</sql>
<method-name>findByName</method-name>
<sql>SELECT  "BRAND_NAME", "ID" FROM "TestBeanEJBTable" WHERE "BRAND_NAME" = ?1</sql>
```

D.2.3 ejb/ee/tx/txECMPbean

```
<method-name>findByName</method-name>
<sql>SELECT "KEY_ID" FROM "TxECMPBeanEJBTable" WHERE "BRAND_NAME" = ?1</sql>
<method-name>findWithinPrimaryKeyRange</method-name>
<sql>SELECT "KEY_ID" FROM "TxECMPBeanEJBTable" WHERE "PRICE" BETWEEN ?1 AND ?2</sql>
<method-name>findByPrice</method-name>
<sql>SELECT "KEY_ID" FROM "TxECMPBeanEJBTable" WHERE "PRICE" = ?1</sql>
<method-name>findWithinPrimaryKeyRange</method-name>
<sql>SELECT "KEY_ID" FROM "TxECMPBeanEJBTable" WHERE "KEY_ID" BETWEEN ?1 AND ?2</sql>
```

E Context Root Mapping Rules for Web Services Tests

The context root mapping rules that are described in this appendix apply to all of the web services test areas, including Jakarta XML Web Services (`jaxws`), Jakarta Web Services Metadata (`jws`), `webservices`, `webservices12`, and `webservices13`.

This appendix covers the following topics:

- Servlet-Based Web Service Endpoint Context Root Mapping
- Jakarta Enterprise Bean-Based Web Service Endpoint Context Root Mapping

E.1 Servlet-Based Web Service Endpoint Context Root Mapping

This section describes the context root mapping for servlet-based web services endpoints and clients. Since most of the application runtime and Web runtime deployment descriptor files have been removed in Jakarta EE 8, the context root mapping for web archives in a Jakarta EE 10 CI becomes the base name of the Web archive file without the file extension. For example, the context root for the archive `web-client.war` defaults to `web-client`.

This covers the mapping for all servlet-based web services endpoints and clients under the Jakarta EE 10 Platform TCK test trees Jakarta XML Web Services (`jaxws`), Jakarta Web Services Metadata (`jws`), webservices, webservices12, webservices13].

For example, for the Jakarta XML Web Services (`jaxws`) test directory, the context root mapping is as shown in the following table.

Table F-1 Context Root Mapping for Jakarta XML Web Services (`jaxws`) Test Directory

Endpoint	Context Root Mapping
`WSW2JDLHttpTest_web.war`	`WSW2JDLHttpTest_web`
`WSW2JDLHttpTest_wsservlet_vehicle_web.war`	`WSW2JDLHttpTest_wsservlet_vehicle_web`

The directory listing is as follows:

```
% cd $TS_HOME/src/com/sun/ts/tests/jaxws/ee/w2j/document/literal/httptest
% ant ld

[echo] WSW2JDLHttpTest.ear
[echo] WSW2JDLHttpTest_web.war
[echo] WSW2JDLHttpTest_web.war.sun-web.xml
[echo] WSW2JDLHttpTest_wsappclient_vehicle.ear
[echo] WSW2JDLHttpTest_wsappclient_vehicle_client.jar
[echo] WSW2JDLHttpTest_wsappclient_vehicle_client.jar.sun-application-client.xml
[echo] WSW2JDLHttpTest_wsejb_vehicle.ear
[echo] WSW2JDLHttpTest_wsejb_vehicle_client.jar
[echo] WSW2JDLHttpTest_wsejb_vehicle_client.jar.sun-application-client.xml
[echo] WSW2JDLHttpTest_wsejb_vehicle_ejb.jar
[echo] WSW2JDLHttpTest_wsejb_vehicle_ejb.jar.sun-ejb-jar.xml
[echo] WSW2JDLHttpTest_wsservlet_vehicle.ear
[echo] WSW2JDLHttpTest_wsservlet_vehicle_web.war
[echo] WSW2JDLHttpTest_wsservlet_vehicle_web.war.sun-web.xml
```

For Web archives, the context root mapping becomes the base name of the Web archive file minus the extension.

E.2 Jakarta Enterprise Bean-Based Web Service Endpoint Context Root Mapping

This section describes the context root mapping for Jakarta Enterprise Bean-based web services endpoints and clients. The context root mapping for Jakarta Enterprise Bean-based web services and clients is based on the following mapping rules that are used for the Jakarta EE 10 CI.

The following algorithm describes the context root mapping rules that are used by the Jakarta EE 10 Compatible Implementation.

```
if sun-ejb-jar.xml deployment descriptor exists
    if <endpoint-address-uri> tag exists
        context root = value of <endpoint-address-uri>
    else
        if WebService.name annotation is specified on implementation bean
            context root = WSDL Service Name + / + WebService.name
        else
            context root = WSDL Service Name + / + Simple Bean Class Name
        endif
    endif
else
    if WebService.name annotation is specified on implementation bean
        context root = WSDL Service Name + / + WebService.name
    else
        context root = WSDL Service Name + / + Simple Bean Class Name
    endif
endif
```

For example, the following table shows the context root mappings for the webservices12/ejb/annotations directory.

Table F-3 Context Root Mappings

Test Directory	Context Root (<endpoint-address-uri>)
WSEjbMultipleClientInjectionTest1	"WSEjbMultipleClientInjectionTest1/ejb"
WSEjbMultipleClientInjectionTest2	"WSEjbMultipleClientInjectionTest2/ejb"
WSEjbNoWebServiceRefInClientTest	"WSEjbNoWebServiceRefInClientTest/ejb"
WSEjbPortFieldInjectionTest	"WSEjbPortFieldInjectionTest/ejb"
WSEjbPortMethodInjectionTest	"WSEjbPortMethodInjectionTest/ejb"
WSEjbSOAPHandlersTest	"WSEjbSOAPHandlersTest/ejb"
WSEjbSOAPHandlersTest2	"WSEjbSOAPHandlersTest2/ejb"
WSEjbWebServiceProviderTest	"WSEjbWebServiceProviderTest/ejb"
WSEjbWebServiceRefTest2	"WSEjbWebServiceRefTest2/ejb"
WSEjbAsyncTest	"WSEjbAsyncTest/ejb"

```
--------------                   ------------------------------------
Test Directory                   Context Root = <endpoint-address-uri>
--------------                   ------------------------------------
WSEjbMultipleClientInjectionTest1    "WSEjbMultipleClientInjectionTest1/ejb"
WSEjbMultipleClientInjectionTest2    "WSEjbMultipleClientInjectionTest2/ejb"
WSEjbNoWebServiceRefInClientTest     "WSEjbNoWebServiceRefInClientTest/ejb"
WSEjbNoWebServiceRefInClientTest     "WSEjbNoWebServiceRefInClientTest/ejb"
WSEjbPortFieldInjectionTest          "WSEjbPortFieldInjectionTest/ejb"
WSEjbPortMethodInjectionTest         "WSEjbPortMethodInjectionTest/ejb"
WSEjbSOAPHandlersTest                "WSEjbSOAPHandlersTest/ejb"
WSEjbSOAPHandlersTest2               "WSEjbSOAPHandlersTest2"/ejb"
WSEjbWebServiceProviderTest          "WSEjbWebServiceProviderTest/ejb"
WSEjbWebServiceRefTest2              "WSEjbWebServiceRefTest2/ejb"
WSEjbAsyncTest                       "WSEjbAsyncTest/ejb"
```

The following table shows the two test directories under the `webservices12/ejb/annotations` that do not specify the `<endpoint-address-uri>` deployment tag or do not contain a Jakarta Enterprise Bean JAR runtime deployment descriptor file. Because of this, the context root is calculated using the previously described formula. In both cases, the context root is calculated as a concatenation of the WSDL Service Name, a slash (`/`), and the Simple Bean Class Name.

Table F-4 Context Root Mapping for Directories Without Endpoint Address URIs

Test Directory	Context Root (WSDL Service Name/Simple Bean Class Name)
WSEjbWebServiceRefTest1	"WSEjbWebServiceRefTest1HelloService/HelloBean"
WSEjbWebServiceRefWithNoDDs Test	"WSEjbWSRefWithNoDDsTestHelloEJBService/WSEjbWSRefWithNoDDsTestHello EJB"

```
--------------                   --------------------------------------------------------
Test Directory                   Context Root = <WSDL Service Name/Simple Bean Class Name>
--------------                   --------------------------------------------------------
WSEjbWebServiceRefTest1              "WSEjbWebServiceRefTest1HelloService/HelloBean"
WSEjbWebServiceRefWithNoDDsTest "WSEjbWSRefWithNoDDsTestHelloEJBService/\
                                     WSEjbWSRefWithNoDDsTestHelloEJB"
```

The context root mappings for some, but not all, tests also exist in the DAT files under the `$TS_HOME/bin` directory. These include the `jaxws-url-props.dat` (Jakarta XML Web Services), `jws-url-props.dat` (Jakarta Web Services Metadata), and `webservices12-url-props.dat` files.

Implementers can use the previously described information in their porting implementation layer for web services.

F Testing a Standalone Jakarta Messaging Resource Adapter

This appendix explains how to set up and configure a Jakarta EE 10 CI and Jakarta EE 10 Platform TCK so a standalone Jakarta Messaging resource adapter can be tested.

This appendix covers the following topics:

- Setting Up Your Environment

- Configuring Jakarta EE 10 Platform TCK

- Configuring a Jakarta EE 10 CI for the Standalone Jakarta Messaging Resource Adapter

- Modifying the Runtime Deployment Descriptors for the Jakarta Messaging MDB and Resource Adapter Tests

- Running the Jakarta Messaging Tests From the Command Line

- Restoring the Runtime Deployment Descriptors for the Jakarta Messaging MDB and Resource Adapter Tests

- Reconfiguring Jakarta EE 10 CI for Jakarta EE 10 Platform TCK After Testing the Standalone Jakarta Messaging Resource Adapter

F.1 Setting Up Your Environment

Before you can run the Jakarta Messaging TCK tests against a standalone Jakarta Messaging Resource Adapter using a Jakarta EE 10 CI, you must install the following components:

- Java SE 8 or 11 software

- Jakarta EE 10 CI software such as Eclipse GlassFish 6.1

- Jakarta EE 10 Platform TCK software

Complete the following steps to set up Eclipse GlassFish 6.1 in your environment:

1. Set the following environment variables in your shell environment:

   ◦ JAVA_HOME to the directory where the Java SE 8 software has been installed

   ◦ JAVAEE_HOME to the directory where the Jakarta EE 10 CI (Eclipse GlassFish 6.1) software has been installed

   ◦ TS_HOME to the directory where the Jakarta EE 10 Platform TCK software has been installed

2. Update your PATH environment variable to include the following directories: JAVA_HOME/bin, JAVAEE_HOME/bin, TS_HOME/bin, and ANT_HOME/bin.

F.2 Configuring Jakarta EE 10 Platform TCK

The `ts.jte` file includes properties that must be set for testing a standalone Jakarta Messaging Resource Adapter using the Jakarta EE 10 CI. The Jakarta Messaging Resource Adapter documentation in the `ts.jte` file should help you understand what you need to set in this step of the testing process.

1. Set the following properties in the `ts.jte` file:

   - javaee.home to the location where the Jakarta EE 10 CI is installed

   - Use the default value or enter a new host name for the `orb.host` property

   - Use the default value or enter a new port number for the `orb.port` property

   - `test.sa.jmsra` to true

   - `jmsra.rarfile` to the location of the standalone Jakarta Messaging Resource Adapter RAR file

   - `jmsra.jarfile` to the location of the standalone Jakarta Messaging Resource Adapter JAR file

   - `jmsra.name` to the name of the Jakarta Messaging Resource Adapter under test

2. Add `$\{jmsra.jarfile}` to the beginning or at the end of the AppClient classpath:
   `APPCPATH=` list of classes and jars followed by `$\{pathsep}$\{jmsra.jarfile}\`
   The `jmsra.jarfile`, which contains all the Jakarta Messaging Resource Adapter classes, needs to be added to the AppClient classpath in the `ts.jte` file. This JAR file will also be copied to the appropriate directory in your Jakarta EE 10 environment when the `config.vi` Ant task, which is described in the next section, is invoked. For the Jakarta EE 10 CI Eclipse GlassFish 6.1, the file is copied to the `JAVAEE_HOME/lib` directory.

F.3 Configuring a Jakarta EE 10 CI for the Standalone Jakarta Messaging Resource Adapter

Invoke the config.vi Ant task to configure the Jakarta EE 10 CI, Eclipse GlassFish 6.1 for TCK 10 and the standalone Jakarta Messaging Resource Adapter:

1. Change to the `TS_HOME/bin` directory.

2. Execute the `config.vi` Ant task.
   The config.vi Ant task executes scripts, which configure your Jakarta EE 10 environment for TCK 10. One of the ant scripts that gets executed will configure and deploy the standalone Jakarta Messaging Resource Adapter, copy the JAR file containing the classes for the standalone Jakarta Messaging Resource Adapter to the `JAVAEE_HOME/lib` directory, create the Jakarta Messaging connector connect ion pools and resources, and create the necessary Jakarta Messaging administration objects. The following Ant scripts are called by the `config.vi` Ant task:

- ◦ TS_HOME/bin/xml/impl/glassfish/jmsra.xml

- ◦ TS_HOME/bin/xml/impl/glassfish/templates/create.jmsra.template
  The script TS_HOME/bin/xml/impl/glassfish/jmsra.xml calls the template file `TS_HOME/bin/xml/impl/glassfish/templates/create.jmsra.template,` which handles the creation of the Jakarta Messaging connector connection pools, the Jakarta Messaging connector resources and the Jakarta Messaging administration objects.
  These scripts are written for the standalone Generic Jakarta Messaging Resource Adapter (GenericJMSRA) for the Jakarta EE 10 CI. If you are using a different Jakarta EE 10 environment, you will need to rewrite these scripts for that environment. If you are using a different standalone Jakarta Messaging resource adapter, you will need to rewrite these scripts for that Jakarta Messaging resource adapter.

F.4 Modifying the Runtime Deployment Descriptors for the Jakarta Messaging MDB and Resource Adapter Tests

After the standalone Jakarta Messaging Resource Adapter has been configured and deployed and the required Jakarta Messaging connector connection pools, Jakarta Messaging connector resources, and Jakarta Messaging administration objects have been created, the `glassfish-ejb-jar` runtime deployment descriptor XML files must be modified for the Jakarta Messaging MDB and Resource Adapter tests. An Ant task handles the modifications.

1. Change to the `TS_HOME/bin` directory.

2. Execute the following Ant task:
   `ant -f xml/impl/glassfish/jmsra.xml modify-jmsmdbejbxml`

This Ant target modifies the glassfish-ejb-jar runtime deployment descriptor XML files in the distribution directory of the Jakarta Messaging MDB and Resource Adapter test directories that exist under `TS_HOME/src/com/sun/ts/tests/jms/ee/mdb` and `TS_HOME/src/com/sun/ts/tests/jms/ee20/ra.`
The modified `glassfish-ejb-jar` runtime deployment descriptor XML files exist under the `TS_HOME/src/com/sun/ts/tests/jms/commonee/xml/descriptors/genericra` directory. These files are copied into the correct distribution test directory under `TS_HOME/dist/com/sun/ts/tests/jms/ee/mdb` and `TS_HOME/dist/com/sun/ts/tests/jms/ee20/ra`.
The `<mdb-resource-adapter>` information for the standalone Jakarta Messaging Resource Adapter being tested is added to the `glassfish-ejb-jar runtime` deployment descriptor XML files. In the default case, the resource adapter being tested is the Generic Jakarta Messaging Resource Adapter (GenericJMSRA). If you are using a different Jakarta EE 10 environment, your runtime deployment descriptor XML files will need to be vendor specific. In this case, you will need to modify the Ant script to handle your vendor-specific runtime deployment descriptor XML files.

F.5 Running the Jakarta Messaging Tests From the Command Line

Run the Jakarta Messaging tests:

1. Change to the `TS_HOME/src/com/sun/ts/tests/jms` directory.

2. Invoke the `runclient` Ant target:
   `ant runclient`

F.6 Restoring the Runtime Deployment Descriptors for the Jakarta Messaging MDB and Resource Adapter Tests

After you run the Jakarta Messaging tests against your standalone Jakarta Messaging Resource Adapter, you need to restore the Jakarta Messaging MDB and Resource Adapter tests. Jakarta EE 10 Platform TCK provides an Ant task that handles the restoration. Invoke the following Ant task to restore the Jakarta Messaging MDB and Resource Adapter `glassfish-ejb-jar` runtime deployment descriptor XML files to their previous state:

1. Change to the `TS_HOME/bin` directory.

2. Invoke the following Ant target:
   `ant -f xml/impl/glassfish/jmsra.xml restore-jmsmdbejbxml`

If you are using another Jakarta EE 10 environment, these runtime deployment descriptor XML files will be vendor specific. In this case, you will need to modify the Ant script to handle the vendor-specific runtime deployment descriptor XML files appropriate for your environment.

F.7 Reconfiguring Jakarta EE 10 CI for Jakarta EE 10 Platform TCK After Testing the Standalone Jakarta Messaging Resource Adapter

After you finish testing the standalone Jakarta Messaging Resource Adapter, you need to reconfigure the Jakarta EE 10 CI before you can continue testing with Jakarta EE 10 Platform TCK:

1. Change to the `TS_HOME/bin` directory.

2. Invoke the `clean.vi` Ant target:
   `ant clean.vi`

3. Set the following properties in the `ts.jte` file:

- javaee.home to the location where the Jakarta EE 10 CI is installed

- Use the default value for the `orb.host` property or enter a new host name

- Use the default value for the `orb.port` property or enter a new port number

- `test.sa.jmsra` to false

- Unset the `jmsra.rarfile` property

- Unset the `jmsra.jarfile` property

- Reset the `jmsra.name` property to `jmsra` to refer to the Jakarta Messaging Resource Adapter for the Jakarta EE 10 CI

4. From the TS_HOME/bin directory, invoke the config.vi Ant task to reconfigure the Jakarta EE 10 CI for Jakarta EE 10 Platform TCK:
   `ant config.vi`