TCK User’s Guide for Technology Implementors

Table of Contents

Eclipse Foundation

Technology Compatibility Kit User's Guide for Jakarta Messaging

Release 3.1 for Jakarta EE

March 2022

Technology Compatibility Kit User's Guide for Jakarta Messaging, Release 3.1 for Jakarta EE

References in this document to JMS refer to the Jakarta Messaging unless otherwise noted.

Preface

This guide describes how to install, configure, and run the Technology Compatibility Kit (TCK) that is used to test the Jakarta Messaging (Messaging 3.1) technology.

The Messaging TCK is a portable, configurable automated test suite for verifying the compatibility of a vendor's implementation of the Messaging 3.1 Specification (hereafter referred to as the vendor implementation or VI). The Messaging TCK uses the JavaTest harness version 5.0 to run the test suite

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

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

Who Should Use This Book

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

Before You Read This Book

You should be familiar with the Messaging 3.1, version 3.1 Specification, which can be found at https://jakarta.ee/specifications/messaging/3.1/.

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

Typographic Conventions

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

Convention	Meaning	Example

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

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

Shell Prompts in Command Examples

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

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

1 Introduction

This chapter provides an overview of the principles that apply generally to all Technology Compatibility Kits (TCKs) and describes the Jakarta Messaging TCK (Messaging 3.1 TCK). It also includes a high level listing of what is needed to get up and running with the Messaging TCK.

This chapter includes the following topics:

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

1.1 Compatibility Testing

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

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

Compatibility test development for a given feature relies on a complete specification and compatible implementation (CI) for that feature. Compatibility testing is not primarily concerned with robustness, performance, nor ease of use.

1.1.1 Why Compatibility Testing is Important

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

- Compatibility testing ensures that the Jakarta platform does not become fragmented as it is ported to different operating systems and hardware environments.
- Compatibility testing benefits developers working in the Jakarta programming language, allowing them to write applications once and then to deploy them across heterogeneous computing environments without porting.
- Compatibility testing allows application users to obtain applications from disparate sources and deploy them with confidence.
- Conformance testing benefits Jakarta platform implementors by ensuring a level playing field for

all Jakarta platform ports.

1.1.2 TCK Compatibility Rules

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

1.1.3 TCK Overview

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

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

The definition of required tests will change over time. Before your final certification test pass, be sure to download the latest version of this TCK.

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

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

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

For further information about the JESP program, go to Jakarta EE Specification Process community page https://jakarta.ee/specifications.

1.2 About the TCK

The Messaging TCK 3.1 is designed as a portable, configurable, automated test suite for verifying the compatibility of a vendor's implementation of the Messaging 3.1 Specification.

1.2.1 TCK Specifications and Requirements

This section lists the applicable requirements and specifications.

- Specification Requirements: Software requirements for a Messaging implementation are described in detail in the Messaging 3.1 Specification. Links to the Messaging specification and other product information can be found at https://jakarta.ee/specifications/messaging/3.1/.
- Messaging Version: The Messaging 3.1 TCK is based on the Messaging Specification, Version 3.1.
- Compatible Implementation: One Messaging 3.1 Compatible Implementation, Eclipse OpenMQ 6.3 is available from the Eclipse EE4J project (https://projects.eclipse.org/projects/ee4j). See the CI documentation page at https://projects.eclipse.org/projects/ee4j.openmq for more information.

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

1.2.2 TCK Components

The Messaging TCK 3.1 includes the following components:

- JavaTest harness version 5.0 and related documentation. See JT Harness web site for additional information.
- Messaging TCK signature tests; check that all public APIs are supported and/or defined as specified in the Messaging Version 3.1 implementation under test.
- If applicable, an exclude list, which provides a list of tests that your implementation is not required to pass.
- API tests for all of the Messaging API in all related packages:
  - `jakarta.jms`

The Messaging TCK tests run on the following platforms:

- CentOS Linux 7

1.2.3 JavaTest Harness

The JavaTest harness version 5.0 is a set of tools designed to run and manage test suites on different Java platforms. To JavaTest, Jakarta EE can be considered another platform. 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 4, "Setup and Configuration."

1.2.4 TCK Compatibility Test Suite

The test suite is the collection of tests used by the JavaTest harness to test a particular technology implementation. In this case, it is the collection of tests used by the Messaging TCK 3.1 to test a Messaging 3.1 implementation. The tests are designed to verify that a vendor's runtime implementation of the technology complies with the appropriate specification. The individual tests correspond to assertions of the specification.

The tests that make up the TCK compatibility test suite are precompiled and indexed within the TCK test directory structure. 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

Each version of a 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 vendor's compatible implementation 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 Messaging TCK.

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

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

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

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

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

1.2.6 TCK Configuration

You need to set several variables in your test environment, modify properties in the `<TS_HOME>/bin/ts.jte` file, and then use the JavaTest harness to configure and run the Messaging tests, as described in Chapter 4, "Setup and Configuration."

The Jakarta EE Specification Process support multiple compatible implementations. These instructions explain how to get started with the Eclipse OpenMQ 6.3 CI. If you are using another compatible implementation, refer to material provided by that implementation for specific instructions and procedures.

1.3 Getting Started With the TCK

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

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

- Java SE 11
- Apache Ant 1.10.0+
- A CI for Messaging 3.1. One example is Eclipse OpenMQ 6.3.
- Messaging TCK version 3.1, which includes:
  - JDOM 1.1.3
  - Apache Commons HTTP Client 3.1
  - Apache Commons Logging 1.1.3
  - Apache Commons Codec 1.9
- The Messaging 3.1 Vendor Implementation (VI)
  See the documentation for each of these software applications for installation instructions. See Chapter 3, "Installation," for instructions on installing the Messaging TCK.

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

     1. Set up your shell environment.
     2. Modify the required properties in the `<TS_HOME>/bin/ts.jte` file.
     3. Configure the JavaTest harness.

  2. Test the Messaging 3.1 implementation.
     Test the Messaging implementation installation by running the test suite. See Chapter 5, "Executing Tests."

2 Procedure for Certification

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

2.1 Certification Overview

The certification process for Messaging 3.1 consists of the following activities:

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

2.2 Compatibility Requirements

The compatibility requirements for Messaging 3.1 consist of meeting the requirements set forth by the rules and associated definitions contained in this section.

2.2.1 Definitions

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

Table 2-1 Definitions

Term	Definition
API Definition Product	A Product for which the only Java class files contained in the product are those corresponding to the application programming interfaces defined by the Specifications, and which is intended only as a means for formally specifying the application programming interfaces defined by the Specifications.
Computational Resource	A piece of hardware or software that may vary in quantity, existence, or version, which may be required to exist in a minimum quantity and/or at a specific or minimum revision level so as to satisfy the requirements of the Test Suite. Examples of computational resources that may vary in quantity are RAM and file descriptors. Examples of computational resources that may vary in existence (that is, may or may not exist) are graphics cards and device drivers. Examples of computational resources that may vary in version are operating systems and device drivers.
Configuration Descriptor	Any file whose format is well defined by a specification and which contains configuration information for a set of Java classes, archive, or other feature defined in the specification.
Conformance Tests	All tests in the Test Suite for an indicated Technology Under Test, as released and distributed by the Eclipse Foundation, excluding those tests on the published Exclude List for the Technology Under Test.
Container	An implementation of the associated Libraries, as specified in the Specifications, and a version of a Java Platform, Standard Edition Runtime Product, as specified in the Specifications, or a later version of a Java Platform, Standard Edition Runtime Product that also meets these compatibility requirements.
Documented	Made technically accessible and made known to users, typically by means such as marketing materials, product documentation, usage messages, or developer support programs.
Exclude List	The most current list of tests, released and distributed by the Eclipse Foundation, that are not required to be passed to certify conformance. The Jakarta EE Specification Committee may add to the Exclude List for that Test Suite as needed at any time, in which case the updated TCK version supplants any previous Exclude Lists for that Test Suite.

Term	Definition
Libraries	The class libraries, as specified through the Jakarta EE Specification Process (JESP), for the Technology Under Test. The Libraries for Jakarta Messaging are listed at the end of this chapter.
Location Resource	A location of classes or native libraries that are components of the test tools or tests, such that these classes or libraries may be required to exist in a certain location in order to satisfy the requirements of the test suite. For example, classes may be required to exist in directories named in a CLASSPATH variable, or native libraries may be required to exist in directories named in a PATH variable.
Maintenance Lead	The corresponding Jakarta EE Specification Project is responsible for maintaining the Specification, and the TCK for the Technology. The Specification Project Team will propose revisions and updates to the Jakarta EE Specification Committee which will approve and release new versions of the specification and TCK.
Operating Mode	Any Documented option of a Product that can be changed by a user in order to modify the behavior of the Product. For example, an Operating Mode can be binary (enable/disable optimization), an enumeration (select from a list of protocols), or a range (set the maximum number of active threads). Note that an Operating Mode may be selected by a command line switch, an environment variable, a GUI user interface element, a configuration or control file, etc.
Product	A vendor’s product in which the Technology Under Test is implemented or incorporated, and that is subject to compatibility testing.
Product Configuration	A specific setting or instantiation of an Operating Mode. For example, a Product supporting an Operating Mode that permits user selection of an external encryption package may have a Product Configuration that links the Product to that encryption package.
Rebuildable Tests	Tests that must be built using an implementation-specific mechanism. This mechanism must produce specification-defined artifacts. Rebuilding and running these tests against a known compatible implementation verifies that the mechanism generates compatible artifacts.
Resource	A Computational Resource, a Location Resource, or a Security Resource.

Term	Definition
Rules	These definitions and rules in this Compatibility Requirements section of this User's Guide.
Runtime	The Containers specified in the Specifications.
Security Resource	A security privilege or policy necessary for the proper execution of the Test Suite. For example, the user executing the Test Suite will need the privilege to access the files and network resources necessary for use of the Product.
Specifications	The documents produced through the Jakarta EE Specification Process (JESP) that define a particular Version of a Technology. The Specifications for the Technology Under Test are referenced later in this chapter.
Technology	Specifications and one or more compatible implementations produced through the Jakarta EE Specification Process (JESP).
Technology Under Test	Specifications and a compatible implementation for Jakarta Messaging Version 3.1.
Test Suite	The requirements, tests, and testing tools distributed by the Maintenance Lead as applicable to a given Version of the Technology.
Version	A release of the Technology, as produced through the Jakarta EE Specification Process (JESP).

2.2.2 Rules for Jakarta Messaging Products

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

Messaging1 The Product must be able to satisfy all applicable compatibility requirements, including passing all Conformance Tests, in every Product Configuration and in every combination of Product Configurations, except only as specifically exempted by these Rules.

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

Messaging1.1 If an Operating Mode controls a Resource necessary for the basic execution of the Test Suite, testing may always use a Product Configuration of that Operating Mode providing that Resource, even if other Product Configurations do not provide that Resource. Notwithstanding such exceptions,

each Product must have at least one set of Product Configurations of such Operating Modes that is able to pass all the Conformance Tests.

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

Messaging1.2 A Product Configuration of an Operating Mode that causes the Product to report only version, usage, or diagnostic information is exempted from these compatibility rules.

Messaging1.3 An API Definition Product is exempt from all functional testing requirements defined here, except the signature tests.

Messaging2 Some Conformance Tests may have properties that may be changed. Properties that can be changed are identified in the configuration interview. Properties that can be changed are identified in the JavaTest Environment (.jte) files in the Test Suite installation. Apart from changing such properties and other allowed modifications described in this User's Guide (if any), no source or binary code for a Conformance Test may be altered in any way without prior written permission. Any such allowed alterations to the Conformance Tests will be provided via the Jakarta EE Specification Project website and apply to all vendor compatible implementations.

Messaging3 The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

Messaging4 The Exclude List associated with the Test Suite cannot be modified.

Messaging5 The Maintenance Lead can define exceptions to these Rules. Such exceptions would be made available as above, and will apply to all vendor implementations.

Messaging6 All hardware and software component additions, deletions, and modifications to a Documented supporting hardware/software platform, that are not part of the Product but required for the Product to satisfy the compatibility requirements, must be Documented and available to users of the Product.

For example, if a patch to a particular version of a supporting operating system is required for the Product to pass the Conformance Tests, that patch must be Documented and available to users of the Product.

Messaging7 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.

Messaging7.1 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.

Messaging8 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.

Messaging9 The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.

2.3 Test Appeals Process

Jakarta has a well established process for managing challenges to its TCKs. Any implementor may submit a challenge to one or more tests in the Messaging 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 Messaging 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 Messaging TCK.

2.3.3 TCK Test Appeals Steps

1. Challenges should be filed via the Jakarta Messaging 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 Messaging

The Jakarta Messaging specification is available from the specification project web-site: https://jakarta.ee/specifications/messaging/3.1/.

2.5 Libraries for Jakarta Messaging

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

- `jakarta.jms`

For the latest list of packages, also see:

https://jakarta.ee/specifications/messaging/3.1/

3 Installation

This chapter explains how to install the Jakarta Messaging TCK software.

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

3.1 Obtaining a Compatible Implementation

Each compatible implementation (CI) will provide instructions for obtaining their implementation. Eclipse OpenMQ 6.3 is a compatible implementation which may be obtained from https://projects.eclipse.org/projects/ee4j.openmq

3.2 Installing the Software

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

- Java SE 11
- Apache Ant 1.10.0+
- A CI for Messaging 3.1, one example is Eclipse OpenMQ 6.3
- Messaging TCK version 3.1, which includes:
  - JDOM 1.1.3
  - Apache Commons HTTP Client 3.1
  - Apache Commons Logging 1.1.3
  - Apache Commons Codec 1.9
- The Messaging 3.1 Vendor Implementation (VI)

Follow these steps:

1. Install the Java SE 11 software, if it is not already installed.
   Download and install the Java SE 11 software from http://www.oracle.com/technetwork/java/javase/downloads/index.html. Refer to the installation instructions that accompany the software for additional information.
2. Install the Apache Ant 1.10.0+ software, if it is not already installed.
   Download and install Apache Ant 1.10.0+ software from Apache Ant Project. 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`.

3. Install the Messaging TCK 3.1 software.
    1. Copy or download the Messaging TCK software to your local system.
       You can obtain the Messaging TCK software from the Jakarta EE site https://jakarta.ee/specifications/messaging/3.1/.
    2. Use the `unzip` command to extract the bundle in the directory of your choice:
       `unzip jakarta-messaging-tck-3.1.0.zip`
       This creates the TCK directory. The TCK is the test suite home, `<TS_HOME>`.
4. Install a Messaging 3.1 Compatible Implementation.
   A Compatible Implementation is used to validate your initial configuration and setup of the Messaging TCK 3.1 tests, which are explained further in Chapter 4, "Setup and Configuration."
   The Compatible Implementations for Messaging are listed on the Jakarta EE Specifications web site: https://jakarta.ee/specifications/messaging/3.1/.
5. Install the Messaging VI to be tested.
   Follow the installation instructions for the particular VI under test.

4 Setup and Configuration

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

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

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

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

After configuring your environment as described in this section, continue with the instructions in Section 4.6, "Using the JavaTest Harness Software."

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. Finally, be sure to use the appropriate separator for your operating system when specifying multiple path entries (`;` on Windows, `:` on UNIX/Linux).

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

4.1.1 To Configure Your Environment for the Messaging TCK

1. Set the following environment variables in your shell environment:
    - `JAVA_HOME` to the directory in which Java SE 11 is installed
    - `TS_HOME` to the directory in which the Messaging 3.1 software is installed
    - `JMS_HOME` to the directory in which the Messaging 3.1 CI has been installed
    - `PATH` to include the following directories: `JAVA_HOME/bin`, `JMS_HOME/bin`, and `ANT_HOME/bin`

2. 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:
   - `jms.home` to the directory in which you installed your Messaging 3.1 implementation
   - `jms.classes` to the JAR file(s) that contain your Messaging 3.1 implementation classes
     For example, if you are using the Eclipse OpenMQ 6.3 CI, the `jms.classes` property would be set to `jms.classes=${ri.jars}`.
   - `impl.vi` to the Messaging 3.1 implementation being tested
     This property signals which configuration script to execute for the `config.vi` and `clean.vi` Ant targets. These targets handle starting and stopping the Messaging implementation under test as well as creating and deleting all of the Messaging administered objects and Messaging users needed by the Messaging TCK.
     The `ts.jte` file supports two Messaging 3.1 Compatible Implementations: Eclipse OpenMQ 6.3 CI and Eclipse GlassFish 6.0 CI.
     For example, if using the Eclipse OpenMQ 6.3 CI , the `impl.vi` property is set to `impl.vi=ri`. With this setting, the config.vi and clean.vi ant targets will invoke the script under `<TS_HOME>/bin/xml/impl/ri/config.vi.xml`.
   - `user` to a valid user name in the Messaging implementation
   - `password` to the password for the user that was specifiedwith the `user` property
   - `jms_timeout` to the length of time used to receive messages
   - `harness.log.port` to the port number that will be used for logging
   - `porting.ts.jmsObjects.class.1` to point to your porting implementation, if you implemented the TSJMSObjectInterface interface
     If you did not implement the interface, use the default setting.
3. If your Messaging 3.1 implementation does not support the `MY_QUEUE`, `MY_TOPIC`, `MyQueueConnectionFactory`, or `MyTopicConnectionFactory` JNDI names, you must provide your own implementation of the `TSJMSObjectInterface` interface. See Implementing the Messaging TCK Porting Package Interface for additional infomation about the TSJMSObjectInterface interface.
4. If you are using the Eclipse OpenMQ 6.3 CI on the Windows platform, edit the `<TS_HOME>/bin/ts.jte` file and add the drive letter to the `admin.pass.file` and `jndi.fs.dir` properties. The property settings for the Windows platform are as follows:

   ```
   admin.pass.file=C:/tmp/ripassword
   jndi.fs.dir=C:/tmp/ri_admin_objects
   ```

   Edit the `<TS_HOME>/bin/xml/impl/ri/jndi.properties` file and add the drive letter to the `java.naming.provider.url` property. The property setting for the Windows platform is as follows:

   ```
   java.naming.provider.url=file:///C:/tmp/ri_admin_objects
   ```

4.1.2 To Configure and Start Your Messaging 3.1 Implementation

1. Set `JMS_HOME` to the location where your Messaging implementation has been installed.
2. Start the JNDI service.

A Vendor Implementation is not required to use a JNDI provider as the object store for Messaging administered objects. For more information, see the `TSJMSObjectsInterface`, an interface that defines a porting layer for looking up the Messaging administered objects. Vendors must either implement this porting package for their specific implementation or use the provided porting package and JNDI file system provider.

3. Start your Messaging implementation.
4. Add the following administered objects:

Name	Type
`MY_QUEUE`	`jakarta.jms.Queue`
`MY_QUEUE2`	`jakarta.jms.Queue`
`testQ0`	`jakarta.jms.Queue`
`testQ1`	`jakarta.jms.Queue`
`testQ2`	`jakarta.jms.Queue`
`testQueue2`	`jakarta.jms.Queue`
`Q2`	`jakarta.jms.Queue`
`MY_TOPIC`	`jakarta.jms.Topic`
`MY_TOPIC2`	`jakarta.jms.Topic`
`testT0`	`jakarta.jms.Topic`
`testT1`	`jakarta.jms.Topic`
`testT2`	`jakarta.jms.Topic`
`MyConnectionFactory`	`jakarta.jms.ConnectionFactory`
`MyQueueConnectionFactory`	`jakarta.jms.QueueConnectionFactory`
`MyTopicConnectionFactory`	`jakarta.jms.TopicConnectionFactory`
`DURABLE_SUB_CONNECTION_FACTORY`	`jakarta.jms.TopicConnectionFactory`

`jms/DURABLE_SUB_CONNECTION_FACTORY` must support durable subscriptions and must be created with `clientID=cts`.

The sections that follow explain how to automatically and manually create these administered objects using the Messaging 3.1 CI.

[automatic-configuration-and-startup-of-the-jms-ri]]

4.1.3 Automatic Configuration and Startup of the Messaging CI

This section is optional.

You do not have to start up and configure the Messaging CI as part of the certification process. You may, however, want to do so to familiarize yourself with the testing process.

The steps in this section explain how to automatically handle the configuration and startup procedures, something you may want to do with your implementation, especially if you have want to test repeatedly. You can examine the provided scripts, then modify them for use with the implementaation under test. Section 4.1.4, "Manual Configuration and Startup of the Messaging CI," provides the manual steps for doing what the provided scripts do automatically.

Complete the following steps to automatically configure and start up the Eclipse OpenMQ 6.3 CI:

1. Set the following environment variables in your shell environment:
   - `JAVA_HOME` to the directory in which the Java SE 11 software has been installed
   - `IMQ_JAVAHOME` to the value of `JAVA_HOME`.
     This is needed for {TTechnologyRI} CI commands and scripts.
   - `TS_HOME` to the directory in which the Messaging TCK software has been installed
   - `JMS_HOME` to the directory in which your Messaging 3.1 implementation has been installed
   - `PATH` to include the following directories: `JAVA_HOME/bin`, `JMS_HOME/bin`, and `/bin`<TS_HOME>/tools/ant`
2. If you are using the CI Eclipse OpenMQ 6.3 on the Windows platform, edit the `<TS_HOME>/bin/ts.jte` file and add the drive letter to the `admin.pass.file` and `jndi.fs.dir` properties. The property settings for the Windows platform are as follows:

```
admin.pass.file=C:/tmp/ripassword
jndi.fs.dir=C:/tmp/ri_admin_objects
```

Edit the `<TS_HOME>/bin/xml/impl/ri/jndi.properties` file and add the drive letter to the `java.naming.provider.url` property. The property setting for the Windows platform is as follows:

```
java.naming.provider.url=file:///C:/tmp/ri_admin_objects
```

1. Invoke `config.vi`, the Ant configuration script, to start and configure the Messaging CI.
   The Messaging TCK comes with a configuration XML file, which automates starting and stopping the Messaging service as well as creating and deleting all of the Messaging administered objects and Messaging users needed by the Messaging TCK. Use the following command to invoke the configuration XML file to start and configure the Messaging TCK.

```
cd $TS_HOME/bin
ant config.vi
```

When used with the Messaging 3.1 CI or the Eclipse OpenMQ 6.3 implementation, this target invokes the <TS_HOME>/bin/xml/impl/ri/config.vi.xml script, which starts the Messaging service and creates the Messaging administered objects and Messaging users needed by the Messaging TCK. In the ts.jte file, the Messaging 3.1 implementation property, impl.vi, needs to be set to ri and the jms.classes property needs to be set to ${ri.jars} for the Messaging CI or Eclipse OpenMQ 6.3.
To automate the process of creating the Messaging administered obects and Messaging users for your Messaging implementation, provide your own Ant-based configuration file, name it <TS_HOME>/bin/xml/impl/vi/config.vi.xml, and set the impl.vi property to vi in your ts.jte file. When you execute the ant config.vi target, the script invokes and executes your script <TS_HOME>/bin/xml/impl/vi/config.vi.xml. 4. Invoke clean.vi, an Ant configuration script, to stop and unconfigure the Messaging CI.
The Messaging TCK comes with an XML configuration file that automates starting and stopping the Messaging service as well as creating and deleting all of the Messaging administered objects and Messaging users needed by the Messaging TCK. Use the following command to invoke the configuration XML file to stop and unconfigure the Messaging CI.

```
cd $TS_HOME/bin
ant clean.vi
```

If you are using the CI Eclipse GlassFisn 6 or Eclipse OpenMQ 6.3, this target invokes the <TS_HOME>/bin/xml/impl/ri/config.vi.xml script which stops the Messaging service and deletes all the Messaging administered objects and Messaging users needed by the Messaging TCK. In the ts.jte file, set the impl.vi property to ri and the jms.classes property to ${ri.jars} for the Messaging CI or Eclipse OpenMQ 6.3.

4.1.4 Manual Configuration and Startup of the Messaging CI

This section is optional. You do not have to start up and configure the Messaging CI as part of the certification process. You may, however, want to do so to familiarize yourself with the testing process.

The steps in this section explain how to manually configure and start up the Messaging CI, something you will do with your implementation. If you have to test repeatedly, you may want to examine the provided scripts that automate the configuration and startup procedures, then modify them for use with the implementaation under test. Section 4.1.3, "Automatic Configuration and Startup of the Messaging CI," provides the steps for using the provided scripts do perform these steps automatically.

Complete the following steps to manually configure and start up the Messaging CI:

1. Set the following environment variables in your shell environment:

- JAVA_HOME to the directory in which the Java SE 11 software has been installed
- IMQ_JAVAHOME to the value of JAVA_HOME.
  This is needed for Messaging 3.1 CI commands and scripts.
- TS_HOME to the directory in which the Messaging TCK Version 3.1 software has been installed
- JMS_HOME to the directory in which your Messaging 3.1 implementation has been installed
- PATH to include the following directories: JAVA_HOME/bin, JMS_HOME/bin, and ANT_HOME/bin

2. Use the following command to start the Eclipse OpenMQ 6.3 Broker Service:

```
(UNIX) $JMS_HOME/bin/imqbrokerd
```

3. Use the following command to create the j2ee TCK user in the Eclipse OpenMQ 6.3 user repository:

```
(UNIX) $JMS_HOME/bin/imqusermgr add -f -u j2ee -p j2ee -g admin
```

4. Use the following command to create the file system directory as the object store for the Eclipse OpenMQ 6.3 Administered Objects:

```
(UNIX) mkdir /tmp/ri_admin_objects
```

   This directory must match the jndi.fs.dir. property setting in the ts.jte file and the java.naming.provider.url property in the <TS_HOME>/bin/xml/impl/ri/jndi.properties file.

5. Create the following Messaging administered objects:

Name	Type
MY_QUEUE	jakarta.jms.Queue
MY_QUEUE2	jakarta.jms.Queue
testQ0	jakarta.jms.Queue
testQ1	jakarta.jms.Queue
testQ2	jakarta.jms.Queue
testQueue2	jakarta.jms.Queue
Q2	jakarta.jms.Queue
MY_TOPIC	jakarta.jms.Topic
MY_TOPIC2	jakarta.jms.Topic
testT0	jakarta.jms.Topic
testT1	jakarta.jms.Topic
testT2	jakarta.jms.Topic

Name	Type
MyConnectionFactory	jakarta.jms.ConnectionFactory
MyQueueConnectionFactory	jakarta.jms.QueueConnectionFactory
MyTopicConnectionFactory	jakarta.jms.TopicConnectionFactory
DURABLE_SUB_CONNECTION_FACTORY	jakarta.jms.TopicConnectionFactory

Use the following commands to create the MY_QUEUE and MY_TOPIC JMS administered objects.

```
$JMS_HOME/bin/imqobjmgr -f add -l MY_QUEUE -o imqDestinationName=MY_QUEUE -t q \
-j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory \
-j java.naming.provider.url=file:////tmp/ri_admin_objects

$JMS_HOME/bin/imqobjmgr -f add -l MY_TOPIC -o imqDestinationName=MY_TOPIC -t t \
-j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory \
-j java.naming.provider.url=file:////tmp/ri_admin_objects
```

Use the following commands to create the MyConnectionFactory, MyQueueConnectionFactory, and MyTopicConnectionFactory JMS administered objects.

```
$JMS_HOME/bin/imqobjmgr -f add -l MyConnectionFactory -t cf \
-j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory \
-j java.naming.provider.url=file:////tmp/ri_admin_objects

$JMS_HOME/bin/imqobjmgr -f add -l MyQueueConnectionFactory -t qf \
-j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory \
-j java.naming.provider.url=file:////tmp/ri_admin_objects

$JMS_HOME/bin/imqobjmgr -f add -l MyTopicConnectionFactory -t tf \
-j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory \
-j java.naming.provider.url=file:////tmp/ri_admin_objects
```

The DURABLE_SUB_CONNECTION_FACTORY administered object must support durable subscriptions and must be created with clientID=cts. Use the following command to create the DURABLE_SUB_CONNECTION_FACTORY administered object.

```
$JMS_HOME/bin/imqobjmgr -f add -l DURABLE_SUB_CONNECTION_FACTORY \
-o imqConfiguredClientID=cts -t tf \
-j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory \
-j java.naming.provider.url=file:////tmp/ri_admin_objects
```

4.2 Implementing the Messaging TCK Porting Package Interface

You may need to provide your own implementation of the `TSJMSObjectsInterface` porting package interface that is provided with the Messaging TCK.

Note that if you do implement this interface, you will also need to set the `porting.ts.jmsObjects.class.1` property in the `ts.jte` file for accessing your porting implementation class. The default for this property value is set to the implementation used by the Messaging Compatible Implementation. You may need to set this value to point to your implementation of this interface. See Manual Configuration and Startup of the Messaging CI for information about modifying the `ts.jte` file.

The `TSJMSObjectsInterface` interface looks like this:

```
public interface TSJMSObjectsInterface {
public Queue getQueue(java.lang.String name)
        throws java.lang.Exception;
public Topic getTopic(java.lang.String name)
        throws java.lang.Exception;
public TopicConnectionFactory
        getTopicConnectionFactory(java.lang.String name)
throws java.lang.Exception;
        public QueueConnectionFactory
getQueueConnectionFactory(java.lang.String name)
        throws java.lang.Exception;
        public ConnectionFactory
getConnectionFactory(java.lang.String name)
        throws java.lang.Exception;
}
```

where:

- The `getQueue` method, which is used to get a Queue, accepts the name of the queue as an input parameter.
- The `getTopic` method, which is used to get a Topic, accepts the name of the topic as an input parameter.
- The `getTopicConnectFactory` method, which is used to get a `TopicConnectionFactory`, accepts the name of the `TopicConnectionFactory` as an input parameter.
- The `getQueueConnectionFactory` method, which is used to get a `QueueConnectionFactory`, accepts the name of the `QueueConnectionFactory` as an input parameter.
- The `getConnectionFactory` method, which is used to get a `ConnectionFactory`, accepts the name of the `ConnectionFactory` as an input parameter.

Make sure that you set the value of the `porting.ts.jmsObjects.class.1` property in the `ts.jte` file to point to your implementation of the `TSJMSObjectsInterface`. Refer to Manual Configuration and Startup of the Messaging CI for information about the list of administered objects you may need to manually create.

A sample implementation of the `TSJMSObjectsInterface` porting package, `SunRIJMSObjects.java`, is provided with the Messaging TCK and can be found in the `<TS_HOME>/src/com/sun/ts/lib/implementation/sun/jms`. directory.

4.3 Publishing the Test Applications

Not needed for the Messaging TCK.

4.4 Custom Configuration Handlers

Configuration handlers are used to configure and unconfigure a Messaging 3.1 implementation during the certification process. These are similar to deployment handlers but used for configuration. A configuration handler is an Ant build file that contains at least the required targets listed below:

- `config.vi` - to configure the vendor implementation
- `clean.vi` - to unconfigure the vendor implementation

These targets are called from the `<TS_HOME>/bin/build.xml` file and call down into the implementation-specific configuration handlers.

To provide your own configuration handler, create a config.vi.xml file with the necessary configuration steps for your implementation and place the file under the `<TS_HOME>/bin/xml/impl/<your_impl>` directory.

For more information, you may wish to view `<TS_HOME>/bin/xml/impl/glassfish/config.vi.xml`, the configuration file for Jakarta EE 10 Compatible Implementation, Eclipse GlassFish.

4.5 Custom Deployment Handlers

Deployment handlers are used to deploy and undeploy the WAR files that contain the tests to be run during the certification process. A deployment handler is an Ant build file that contains at least the required targets listed in the table below.

The Messaging TCK provides these deployment handlers:

- <TS_HOME>/bin/xml/impl/none/deploy.xml
- <TS_HOME>/bin/xml/impl/glassfish/deploy.xml
- <TS_HOME>/bin/xml/impl/tomcat/deploy.xml

The deploy.xml files in each of these directories are used to control deployment to a specific container (no deployment, deployment to the Eclipse GlassFish Web container, deployment to the Tomcat Web container) denoted by the name of the directory in which each deploy.xml file resides. The primary build.xml file in the <TS_HOME>/bin directory has a target to invoke any of the required targets (-deploy, -undeploy, -deploy.all, -undeploy.all).

4.5.1 To Create a Custom Deployment Handler

To deploy tests to another Messaging implementation, you must create a custom handler.

1. Create a new directory in the <TS_HOME>/bin/xml/impl directory tree. For example, create the <TS_HOME>/bin/xml/impl/my_deployment_handler directory. Replace my_deployment_handler with the value of the impl.vi property that you set in Step 5 of the configuration procedure described in Section 4.2, "Configuring Your Environment to Repackage and Run the TCK Against the Vendor Implementation".
2. Copy the deploy.xml file from the <TS_HOME>/bin/xml/impl/none directory to the directory that you created.
3. Modify the required targets in the deploy.xml file. This is what the deploy.xml file for the "none" deployment handler looks like.

```
<project name="No-op Deployment" default="deploy">
    <!-- No-op deployment target -->
    <target name="-deploy">
        <echo message="No deploy target implemented for this deliverable"/>
    </target>
    <target name="-undeploy">
        <echo message="No undeploy target implemented for this deliverable"/>
    </target>
    <target name="-deploy.all">
        <echo message="No deploy target implemented for this deliverable"/>
    </target>
    <target name="-undeploy.all">
        <echo message="No undeploy target implemented for this deliverable"/>
    </target>
</project>
```

Although this example just echoes messages, it does include the four required Ant targets (-deploy, -undeploy, -deploy.all, -undeploy.all) that your custom deploy.xml file must contain. With this as

your starting point, look at the required targets in the `deploy.xml` files in the Tomcat and Eclipse Glassfish directories for guidance as you create the same targets for the Web container in which you will run your implementation of Messaging.

The following Ant targets can be called from anywhere under the `<TS_HOME>/src` directory:

- `deploy`
- `undeploy`
- `deploy.all`
- `undeploy.all`

The `deploy.all` and `undeploy.all` targets can also be called from the `<TS_HOME>/bin` directory.

The targets in the `deploy.xml` file are never called directly. They are called indirectly by the targets listed above.

4.6 Using the JavaTest Harness Software

There are two general ways to run the Messaging TCK test suite using the JavaTest harness software:

- Through the JavaTest GUI; if using this method, please continue on to Section 4.7, "Using the JavaTest Harness Configuration GUI."
- In JavaTest batch mode, from the command line in your shell environment; if using this method, please proceed directly to Chapter 5, "Executing Tests."

4.7 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 Messaging TCK test environment. This section covers the following topics:

- Configuration GUI Overview
- Starting the Configuration GUI
- To Configure the JavaTest Harness to Run the Messaging TCK Tests
- Modifying the Default Test Configuration

It is only necessary to proceed with this section if you want to run the JavaTest harness in GUI mode. If you plan to run the JavaTest harness in command-line mode, skip the remainder of this chapter, and continue with Chapter 5, "Executing Tests."

4.7.1 Configuration GUI 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:

- 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.
- 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.

The first time you run the JavaTest harness software, you are asked to specify the test suite and work directory that you want to use. (These parameters can be changed later from within the JavaTest harness GUI.)

Once the JavaTest harness GUI is displayed, whenever you choose Start, then Run Tests 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.

4.7.2 Starting the Configuration GUI

Before you start the JavaTest harness software, you must have a valid test suite and Java SE 11 installed on your system.

The Messaging TCK includes an Ant script that is used to execute the JavaTest harness from the `<TS_HOME>` directory. Using this Ant script to start the JavaTest harness is part of the procedure described in Section 4.7.3, "To Configure the JavaTest Harness to Run the TCK Tests."

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.

After you specify a work directory, you can use the Test Manager to configure and run tests as described in Section 4.7.3, "To Configure the JavaTest Harness to Run the TCK Tests."

4.7.3 To Configure the JavaTest Harness to Run the TCK Tests

The answers you give to some of the configuration interview questions are specific to your site. For example, the name of the host on which the JavaTest harness is running. Other configuration parameters can be set however you wish. For example, where you want test report files to be stored.

Note that you only need to complete all these steps the first time you start the JavaTest test harness. After you complete these steps, you can either run all of the tests by completing the steps in Section 5.1, "Starting JavaTest," or run a subset of the tests by completing the steps in Section 5.2, "Running a Subset of the Tests."

1. Change to the `<TS_HOME>/bin` directory and start the JavaTest test harness:
   ```
   cd <TS_HOME>/bin
   ant gui
   ```
2. From the File menu, click **Open Quick Start Wizard**.
   The Welcome screen displays.
3. Select **Start a new test run**, and then click **Next**.
   You are prompted to create a new configuration or use a configuration template.
4. Select **Create a new configuration**, and then click **Next**.
   You are prompted to select a test suite.
5. Accept the default suite (`<TS_HOME>/src`), and then click **Next**.
   You are prompted to specify a work directory to use to store your test results.
6. Type a work directory name or use the **Browse** button to select a work directory, and then click **Next**.
   You are prompted to start the configuration editor or start a test run. At this point, the Messaging TCK is configured to run the default test suite.
7. Deselect the **Start the configuration editor** option, and then click **Finish**.
8. Click **Run Tests**, then click **Start**.
   The JavaTest harness starts running the tests.
9. To reconfigure the JavaTest test harness, do one of the following:

- Click **Configuration**, then click **New Configuration**.
- Click **Configuration**, then click **Change Configuration**.

10. Click **Report**, and then click **Create Report**.
11. Specify the directory in which the JavaTest test harness will write the report, and then click **OK**. A report is created, and you are asked whether you want to view it.
12. Click **Yes** to view the report.

4.7.4 Modifying the Default Test Configuration

The JavaTest GUI enables you to configure numerous test options. These options are divided into two general dialog box groups:

- Group 1: Available from the JavaTest **Configure/Change Configuration** submenus, the following options are displayed in a tabbed dialog box:
  - **Tests to Run**
  - **Exclude List**
  - **Keywords**
  - **Prior Status**
  - **Test Environment**
  - **Concurrency**
  - **Timeout Factor**
- Group 2: Available from the JavaTest **Configure/Change Configuration/Other Values** submenu, or by pressing `Ctrl+E`, the following options are displayed in a paged dialog box:
  - **Environment Files**
  - **Test Environment**
  - **Specify Tests to Run**
  - **Specify an Exclude List**

Note that there is some overlap between the functions in these two dialog boxes; for those functions use the dialog box that is most convenient for you. Please refer to the JavaTest Harness documentation or the online help for complete information about these various options.

5 Executing Tests

The Messaging 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 in the documentation bundle.

This chapter includes the following topics:

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

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

5.1 Starting JavaTest

There are two general ways to run the Messaging TCK using the JavaTest harness software:

- Through the JavaTest GUI
- From the command line in your shell environment

The `ant` command referenced in the following two procedures and elsewhere in this guide is the Apache Ant build tool, which will need to be downloaded separately. The `build.xml` file in `<TS_HOME>/bin` contains the various Ant targets for the Messaging TCK test suite.

5.1.1 To Start JavaTest in GUI Mode

Execute the following commands:

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

5.1.2 To Start JavaTest in Command-Line Mode

1. Change to any subdirectory under <TS_HOME>/src/com/sun/ts/tests.
2. Start JavaTest using the following command:

```
ant runclient
```

Example 5-1 Messaging TCK Signature Tests

To run the Messaging TCK signature tests, enter the following commands:

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

Example 5-2 Single Test Directory

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

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

Example 5-3 Subset of Test Directories

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

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

5.2 Running a Subset of the Tests

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

- Section 5.2.1, "To Run a Subset of Tests in GUI Mode"
- Section 5.2.2, "To Run a Subset of Tests in Command-Line Mode"
- Section 5.2.3, "To Run a Subset of Tests in Batch Mode Based on Prior Result Status"

5.2.1 To Run a Subset of Tests in GUI Mode

1. From the JavaTest main menu, click **Configure**, then click **Change Configuration**, and then click **Tests to Run**.
   The tabbed Configuration Editor dialog box is displayed.
2. Click **Specify** from the option list on the left.
3. 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, or select just a single test.
4. Click **Save File**.
5. Click **Run Tests**, and then click **Start** to run the tests you selected.
   Alternatively, you can `right-click` the test you want from the test tree in the left section of the JavaTest main window, and choose **Execute These Tests** from the menu.
6. Click **Report**, and then click **Create Report**.
7. Specify the directory in which the JavaTest test harness will write the report, and then click **OK**
   A report is created, and you are asked whether you want to view it.
8. Click **Yes** to view the report.

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

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

   ```
   ant runclient
   ```

   The tests in the directory and its subdirectories are run.

5.2.3 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 the Messaging tests with a status of failed and error by invoking the following commands:

```
ant -DpriorStatus="fail,error" runclient
```

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

5.3 Running the TCK Against another CI

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

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

5.4 Running the TCK Against a Vendor's Implementation

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

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

5.5 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.

5.5.1 Creating Test Reports

Use the following modes to create test reports:

- Section 5.5.1.1, "To Create a Test Report in GUI Mode"
- Section 5.5.1.2, "To Create a Test Report in Command-Line Mode"

5.5.1.1 To Create a Test Report in GUI Mode

1. From the JavaTest main menu, click **Report**, then click **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.

5.5.1.2 To Create a Test Report in Command-Line Mode

1. Specify where you want to create the test report.
   1. To specify the report directory from the command line at runtime, use:

      ```
      ant -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_HOME>/bin/ts.jte`.
   For example:

```
ant -Dreport.dir="none"
```

5.5.2 Viewing an Existing Test Report

Use the following modes to view an existing test report:

- Section 5.5.2.1, "To View an Existing Report in GUI Mode"
- Section 5.5.2.2, "To View an Existing Report in Command-Line Mode"

5.5.2.1 To View an Existing Report in GUI Mode

1. From the JavaTest main menu, click **Report**, then click **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 ReportBrowser.

5.5.2.2 To View an Existing Report in Command-Line Mode

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_HOME>/bin/ts.jte`.

6 Debugging Test Problems

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

This chapter includes the following topics:

6.1 Overview

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

- Errors: Tests with errors could not be executed by the 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. Ant test execution tasks provide command-line users with immediate test execution feedback to the display. Available JTR report files and log files can also help command-line users troubleshoot test run problems.

For every test run, the JavaTest harness creates a set of report files in the reports directory, which you specified by setting the `report.dir` property in the `<TS_HOME>/bin/ts.jte` file. The report files contain information about the test description, environment, messages, properties used by the test, status of the test, and test result. 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.

The work directory, which you specified by setting the `work.dir` property in the `<TS_HOME>/bin/ts.jte` file, contains several files that were deposited there during test execution: `harness.trace`, `log.txt`, `lastRun.txt`, and `testsuite`. Most of these files provide information about the harness and environment in which the tests were executed.

You can set `harness.log.traceflag=true` in `<TS_HOME>/bin/ts.jte` to get more debugging information.

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

6.2 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)

6.3 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.

6.4 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.

6.5 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 Section 5.5, "Test Reports," for more information.

6.6 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).

A Frequently Asked Questions

This appendix contains the following questions.

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

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

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

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

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

The JavaTest exclude file (`<TS_HOME>/bin/ts.jtx`) contains all tests that are not required to be run. The following is a list of reasons for a test to be included in the Exclude List:

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

Appendix B is not used for the Messaging TCK.