# Technology Compatibility Kit User's Guide for Technology Licensees

# Table of Contents

# Eclipse Foundation

Technology Compatibility Kit User's Guide for Java API for XML-based RPC

Release 1.1 for Jakarta EE

September 2019

Technology Compatibility Kit User's Guide for Java API for XML-based RPC, Release 1.1 for Jakarta EE

# Preface

This guide describes how to install, configure, and run the Technology Compatibility Kit (TCK) that is used to test the Java API for XML-based RPC (JAX-RPC 1.1) (JSR 101) technology.

The JAX-RPC TCK is a portable, configurable automated test suite for verifying the compatibility of a licensee's implementation of the JAX-RPC 1.1 Specification (hereafter referred to as the licensee implementation). The JAX-RPC TCK uses the JavaTest harness version 5.0 to run the test suite

> ℹ️ 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.

Refer to the Java Licensee Engineering (https://javapartner.oracle.com) Web site for answers to frequently asked questions and send questions you may have to your Java Licensee Engineering contact.

## Who Should Use This Book

This guide is for licensees of the JAX-RPC 1.1 technology to assist them in running the test suite that verifies compatibility of their implementation of the JAX-RPC 1.1 Specification.

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

## Before You Read This Book

You should be familiar with the JAX-RPC 1.1 Specification, which can be found at http://jcp.org/en/jsr/detail?id=101.

Before running the tests in the JAX-RPC 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**.<br><br>A **cache** is a copy that is stored locally.<br><br>`machine_name% *su*`<br>`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.<br><br>Use `ls -a` to list all files.<br><br>`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*.<br><br>Do *not* save the file.<br><br>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 Java API for XML-based RPC TCK (JAX-RPC TCK 1.1) (JSR 101). It also includes a high level listing of what is needed to get up and running with the JAX-RPC 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 reference implementation for that feature. Compatibility testing is not primarily concerned with robustness, performance, or ease of use.

### 1.1.1 Why Compatibility Testing is Important

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

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

Java platform ports.

## 1.1.2 TCK Compatibility Rules

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

## 1.1.3 TCK Overview

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

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

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

## 1.1.4 Java Community Process (JCP) Program and Compatibility Testing

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

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

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

# 1.2 About the TCK

The JAX-RPC TCK 1.1 is designed as a portable, configurable, automated test suite for verifying the compatibility of a licensee's implementation of the JAX-RPC 1.1 Specification.

## 1.2.1 TCK Specifications and Requirements

This section lists the applicable requirements and specifications.

- Specification Requirements: Software requirements for a JAX-RPC implementation are described in detail in the JAX-RPC 1.1 Specification. Links to the JAX-RPC specification and other product information can be found at http://jcp.org/en/jsr/detail?id=101.

- JAX-RPC Version: The JAX-RPC TCK 1.1 is based on the JAX-RPC Specification, Version 1.1.

- Reference Implementation: The JAX-RPC 1.1 Reference Implementation (Eclipse Glassfish 5.0) is available from Java Licensee Engineering (https://javapartner.oracle.com). See the RI documentation page at http://javaee.github.io/glassfish for more information.

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

## 1.2.2 TCK Components

The JAX-RPC TCK 1.1 includes the following components:

- JavaTest harness version 5.0 and related documentation. See the `ReleaseNotes-jtharness.html` file and the JT Harness web site for additional information.

- JAX-RPC TCK signature tests; check that all public APIs are supported and/or defined as specified in the JAX-RPC Version 1.1 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 JAX-RPC API in all related packages:
    - `javax.xml.rpc`
    - `javax.xml.rpc.encoding`
    - `javax.xml.rpc.handler`
    - `javax.xml.rpc.handler.soap`
    - `javax.xml.rpc.holders`
    - `javax.xml.rpc.namespace`
    - `javax.xml.rpc.server`

    ○ `javax.xml.rpc.soap`

The JAX-RPC TCK tests run on the following platforms:

- Windows 10

- Oracle Linux 7.1

### 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. 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 JAX-RPC TCK 1.1 to test a JAX-RPC 1.1 implementation. The tests are designed to verify that a licensee'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 licensee 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 JAX-RPC TCK.

> From time to time, updates to the Exclude List are made available on the Java Licensee Engineering (https://javapartner.oracle.com) Web site. You should always make sure you are using an up-to-date copy of the Exclude List before running the JAX-RPC 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 reference implementations. Any tests that fail when run on a reference Java 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.

> Licensees 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 JAX-RPC tests, as described in Chapter 4, "Setup and Configuration."

# 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 JAX-RPC 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 EE 8 RI or, at a minimum, a Web server with a Servlet container

- Java SE 8

- JAX-RPC 1.1 RI, which is Eclipse Glassfish 5.0

- JAX-RPC TCK version 1.1, which includes:

  - JDOM 1.0

  - Apache Commons HTTP Client 3.1

  - Apache Commons Logging 1.1.1

  - Apache Commons Codec 1.3

  - Apache Ant 1.8.4

- The JAX-RPC 1.1 Vendor Implementation (VI)
  See the documentation for each of these software applications for installation instructions. See Chapter 3, "Installation," for instructions on installing the JAX-RPC TCK.

  1. Set up the JAX-RPC 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 JAX-RPC 1.1 implementation.
     Test the JAX-RPC 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 Java API for XML-based RPC. This chapter contains the following sections:

- Certification Overview

- Compatibility Requirements

- Test Appeals Process

- Specifications for Java API for XML-based RPC

- Libraries for Java API for XML-based RPC

## 2.1 Certification Overview

The certification process for JAX-RPC 1.1 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 Java Partner organization that you have finished testing and that you meet all of the compatibility requirements.

## 2.2 Compatibility Requirements

The compatibility requirements for JAX-RPC 1.1 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 distributed by the Maintenance Lead, excluding those tests on the Exclude List for the Technology Under Test. |
| Documented | Made technically accessible and made known to users, typically by means such as marketing materials, product documentation, usage messages, or developer support programs. |
| Exclude List | The most current list of tests, distributed by the Maintenance Lead, that are not required to be passed to certify conformance. The Maintenance Lead may add to the Exclude List for that Test Suite as needed at any time, in which case the updated Exclude List supplants any previous Exclude Lists for that Test Suite. |
| Libraries | The class libraries, as specified through the Java Community Process (JCP), for the Technology Under Test.

The Libraries for Java API for XML-based RPC are listed at the end of this chapter. |

| Term | Definition |
|---|---|
| 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 Java Community Process member responsible for maintaining the Specification, reference implementation, and TCK for the Technology. Oracle is the Maintenance Lead for Java API for XML-based RPC. |
| Operating Mode | Any Documented option of a Product that can be changed by a user in order to modify the behavior of the Product.

For example, an Operating Mode can be binary (enable/disable optimization), an enumeration (select from a list of protocols), or a range (set the maximum number of active threads).

Note that an Operating Mode may be selected by a command line switch, an environment variable, a GUI user interface element, a configuration or control file, etc. |
| Product | A licensee product in which the Technology Under Test is implemented or incorporated, and that is subject to compatibility testing. |
| Product Configuration | A specific setting or instantiation of an Operating Mode.

For example, a Product supporting an Operating Mode that permits user selection of an external encryption package may have a Product Configuration that links the Product to that encryption package. |
| 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 the Java EE 8 RI verifies that the mechanism generates compatible artifacts. |
| Resource | A Computational Resource, a Location Resource, or a Security Resource. |
| Rules | These definitions and rules in this Compatibility Requirements section of this User's Guide. |
| Security Resource | A security privilege or policy necessary for the proper execution of the Test Suite.

For example, the user executing the Test Suite will need the privilege to access the files and network resources necessary for use of the Product. |

| Term | Definition |
|------|-----------|
| Specifications | The documents produced through the Java Community Process that define a particular Version of a Technology.<br><br>The Specifications for the Technology Under Test are referenced later in this chapter. |
| Technology | Specifications and a reference implementation produced through the Java Community Process. |
| Technology Under Test | Specifications and the reference implementation for Java API for XML-based RPC Version 1.1. |
| 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 Java Community Process. |

## 2.2.2 Rules for Java API for XML-based RPC Products

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

**JAX-RPC1** 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.

**JAX-RPC1.1** 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.

**JAX-RPC1.2** 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.

**JAX-RPC1.3** An API Definition Product is exempt from all functional testing requirements defined here,

except the signature tests.

**JAX-RPC2** 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 would be posted to the Java Licensee Engineering web site and apply to all licensees.

**JAX-RPC3** The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

**JAX-RPC4** The Exclude List associated with the Test Suite cannot be modified.

**JAX-RPC5** The Maintenance Lead can define exceptions to these Rules. Such exceptions would be made available to and apply to all licensees.

**JAX-RPC6** 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.

**JAX-RPC7** 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.

**JAX-RPC7.1** 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.

**JAX-RPC8** 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.

**JAX-RPC9** The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.

# 2.3 Test Appeals Process

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

If a test is determined to be invalid in function or if its basis in the specification is suspect, the test may be challenged by any licensee of the JAX-RPC TCK. Each test validity issue must be covered by a separate test challenge. Test validity or invalidity will be determined based on its technical correctness such as:

- Test has bugs (i.e., program logic errors).

- Specification item covered by the test is ambiguous.

- Test does not match the specification.

- Test assumes unreasonable hardware and/or software requirements.

- Test is biased to a particular implementation.

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

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

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

- Tests that are placed on the Exclude List will be placed on the Exclude List within one business day after the determination of test validity. The new Exclude List will be made available to all JAX-RPC TCK licensees on the Java Licensee Engineering website.

- Oracle, as Maintenance Lead has the option of creating alternative tests to address any challenge. Alternative tests (and criteria for their use) will be made available on the Java Licensee Engineering website.

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

## 2.3.1 TCK Test Appeals Steps

1. Java API for XML-based RPC TCK licensee writes a test challenge to Java Licensee Engineering

contesting the validity of one or a related set of JAX-RPC tests.

A detailed justification for why each test should be invalidated must be included with the challenge as described in Example 2-1, "Test Challenge Form".

2. Java Licensee Engineering evaluates the challenge.

   If the appeal is incomplete or unclear, it is returned to the submitting licensee for correction. If all is in order, Java Licensee Engineering will check with the responsible test developers to review the purpose and validity of the test before writing a response as described in Example 2-2, "Test Challenge Response Form". Java Licensee Engineering will attempt to complete the response within 5 business days. If the challenge is similar to a previously rejected test challenge (i.e., same test and justification), Java Licensee Engineering will send the previous response to the licensee.

3. The challenge and any supporting materials from test developers is sent to the specification engineers for evaluation.

   A decision of test validity or invalidity is normally made within 15 working days of receipt of the challenge. All decisions will be documented with an explanation of why test validity was maintained or rejected.

4. The licensee is informed of the decision and proceeds accordingly.

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

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

## 2.3.2 Test Challenge and Response Forms

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

Example 2-1 Test Challenge Form

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

Example 2-2 Test Challenge Response Form

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

# 2.4 Specifications for Java API for XML-based RPC

The Java API for XML-based RPC specification is available on the JSR 101 Web site at http://jcp.org/en/jsr/detail?id=101 or on the Java Community Process (http://jcp.org/en/home/index) site.

# 2.5 Libraries for Java API for XML-based RPC

The following is a list of the packages comprising the required class libraries for JAX-RPC 1.1:

- javax.xml.rpc
- javax.xml.rpc.encoding
- javax.xml.rpc.handler
- javax.xml.rpc.handler.soap

- `javax.xml.rpc.holders`
- `javax.xml.rpc.namespace`
- `javax.xml.rpc.server`
- `javax.xml.rpc.soap`

For the latest list of packages, also see:

http://jcp.org/en/jsr/detail?id=101

# 3 Installation

This chapter explains how to install the Java API for XML-based RPC 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 the Reference Implementation

You can obtain the JAX-RPC Reference Implementation (RI) (Eclipse Glassfish 5.0) software from Java Licensee Engineering (https://javapartner.oracle.com).

## 3.2 Installing the Software

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

- Java EE 8 RI or, at a minimum, a Web server with a Servlet container
- Java SE 8
- JAX-RPC 1.1 RI, which is Eclipse Glassfish 5.0
- JAX-RPC TCK version 1.1, which includes:
  - JDOM 1.0
  - Apache Commons HTTP Client 3.1
  - Apache Commons Logging 1.1.1
  - Apache Commons Codec 1.3
  - Apache Ant 1.8.4
- The JAX-RPC 1.1 Vendor Implementation (VI)

Follow these steps:

1. Install the Java SE 8 software, if it is not already installed.
   Download and install the Java SE 8 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 JAX-RPC TCK 1.1 software.

   1. Copy or download the JAX-RPC TCK software to your local system.

You can obtain the JAX-RPC TCK software from the Java Licensee Engineering (https://javapartner.oracle.com) Web site. The JAX-RPC TCK software is located in the Web site's Download Center area.

2. Use the `unzip` command to extract the bundle in the directory of your choice:
   `unzip jaxrpc-1_1-fr-tck.zip`
   This creates the TCK directory. The TCK is the test suite home, `<TS_HOME>`.

3. Install the Java EE 8 RI software (the servlet Web container used for running the JAX-RPC TCK with the JAX-RPC 1.1 RI), if it is not already installed.
   Download and install the Servlet Web container with the JAX-RPC 1.1 RI used for running the JAX-RPC TCK 1.1, represented by the Java EE 8 RI. This software can be obtained from the Java Licensee Engineering Web site. Refer to any installation instructions that accompany the software for additional information.

4. Install the JAX-RPC 1.1 Reference Implementation.
   The Reference Implementation is used to validate your initial configuration and setup of the JAX-RPC TCK 1.1 tests, which are explained further in Chapter 4, "Setup and Configuration."
   The JAX-RPC RI can be obtained from Java Licensee Engineering (https://javapartner.oracle.com).

5. Install a Web server on which the JAX-RPC TCK test applications can be published for testing the VI.

6. Install the JAX-RPC Vendor Implementation (VI) to be tested.
   Follow the installation instructions for the particular VI under test.

# 4 Setup and Configuration

This chapter describes how to set up the JAX-RPC 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 JAX-RPC TCK.

## 4.1 Configuring Your Environment to Run the TCK Against the Reference Implementation

After configuring your environment as described in this section, continue with the instructions in Section 4.3, "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.

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

   1. `JAVA_HOME` to the directory in which Java SE 8 is installed

   2. `TS_HOME` to the directory in which the JAX-RPC TCK 1.1 software is installed

   3. `JWSDP_HOME` to the directory in which the JAX-RPC 1.1 RI has been installed

   4. `PATH` to include the following directories: `JAVA_HOME/bin`, `TS_HOME/bin`, and `<TS_HOME>/tools/ant/bin`

2. Edit your `<TS_HOME>/bin/build.properties` file and set the following environment variables:

   1. Set the `webserver.home` property to the directory in which the Java Web Services Developers Pack 1.3 software is installed.
      By default this is `${env.JWSDP_HOME}`.

   2. Set the `webserver.host` property to the host on which the Web server is running.
      By default this is `localhost`.

   3. Set the `webserver.port` property to the port on which the web Server is running.

4. Set the `j2ee.home.ri` property to the directory in which the J2SE software is installed, for reference to the packager tool used by the build infrastructure. You only need to set this property if you want to build the JAX-RPC TCK software.

5. Set the `jaxrpc.home` property to the path for Java Web Services Developers Pack, which contains the JAX-RPC mapping tool, is installed.

6. Set the `jaxrpc.tool` property to the path where the JAX-RPC 1.1 `JavaToWSDL` and `WSDLToJava` mapping tool exists.
   By default this is `<JWSDP_HOME>/jaxrpc/bin/wscompile.sh`.

7. Set the local_classes property to point to the JAX-RPC RI classes/jars and the TCK classes/jars (tsharness.jar, javatest.jar, jaxrpctck.jar). For the Java Web Services Developers Pack (JWSDP 1.3) your `local_classes` property setting is:

```
local_classes=${env.JWSDP_HOME}/jaxp/lib/jaxp-api.jar:
${env.JWSDP_HOME}/jaxp/lib/endorsed/xercesImpl.jar:
${env.JWSDP_HOME}/jaxp/lib/endorsed/xsltc.jar:
${env.JWSDP_HOME}/jaxp/lib/endorsed/xalan.jar:
${env.JWSDP_HOME}/jaxp/lib/endorsed/sax.jar:
${env.JWSDP_HOME}/jaxp/lib/endorsed/dom.jar:
${env.JWSDP_HOME}/saaj/lib/saaj-impl.jar:
${env.JWSDP_HOME}/saaj/lib/saaj-api.jar:
${env.JWSDP_HOME}/jaxrpc/lib/jaxrpc-impl.jar:
${env.JWSDP_HOME}/jaxrpc/lib/jaxrpc-api.jar:
${env.JWSDP_HOME}/jaxrpc/lib/jaxrpc-spi.jar:
${env.JWSDP_HOME}/jwsdp-shared/lib/mail.jar:
${env.JWSDP_HOME}/jwsdp-shared/lib/activation.jar:
${env.JWSDP_HOME}/jwsdp-shared/lib/commons-logging.jar:
${env.JWSDP_HOME}/jwsdp-shared/lib/jax-qname.jar:
${env.JWSDP_HOME}/jwsdp-shared/lib/relaxngDatatype.jar:
${env.JWSDP_HOME}/jwsdp-shared/lib/xsdlib.jar:
${env.JWSDP_HOME}/jwsdp-shared/lib/jaas.jar:
${env.TS_HOME}/lib/javatest.jar:${env.TS_HOME}/lib/
tsharness.jar:
${env.TS_HOME}/lib/jaxrpctck.jar:${env.JWSDP_HOME}/common/
lib/servlet-api.jar
```

3. Provide your own implementation of the porting package interface provided with the JAX-RPC TCK.

   - The porting package interface, `TSURLInterface.java`, obtains URL strings for web resources in an implementation-specific manner.

   - `TSJAXRPCInterface.java` – Obtains RPC service endpoint URLs and WSDL service file location URLs in a vendor-specific manner using standard properties that map to the URL values supplied by the vendor and to obtain Service Implementation Classes in a vendor-specific manner.
     API documentation for the `TSURLInterface.java` porting package interface is available in the

JAX-RPC TCK documentation bundle.

4.  Edit your `<TS_HOME>/bin/ts.jte` file and modify the following environment settings:

    1.  Set the `webServerHost` property to the name of the host on which Java EE 8 RI is running.
        The default setting is `localhost`.

    2.  Set the `webServerPort` property to the port number of the host on which Java EE 8 RI is running.
        The default setting is `8080`.

    3.  Set the `secureWebServerPort` property to the secure SSL HTTP/1.1 port number of the host on which the web server is running.+ The default setting is `8443`.

    4.  Set the `user` and `password` properties to the user and password used for the basic authentication tests.
        The default setting is `j2ee` for both.

    5.  Set the `authuser` and `authpassword` properties to the user and password used for the basic authentication tests.
        The default setting is `javajoe` for both.

    6.  Set the `porting.ts.url.class.1` property to your porting implementation class that is used for obtaining URLs.
        The default setting for the Sun RI porting implementation is:
        `com.sun.ts.lib.implementation.sun.common.SunRIURL`.

    7.  Set the porting.ts.jaxrpc.class.1 property to your porting implementation class that is used for obtaining service implementation classes, port information, and URLs for service endpoints and WSDL locations.+ The default setting for the Sun RI porting implementation is:
        `com.sun.ts.lib.implementation.sun.jaxrpc.SunRIJAXRPC`

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

After configuring your environment as described in this section, continue with the instructions in Section 4.3, "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.

## 4.2.1 Running the JAX-RPC TCK on a Vendor's Implementation

The JAX-RPC 1.1 specification does not specify any normative model for packaging and deployment of service endpoints on a servlet container-based JAX-RPC runtime system. A JAX-RPC 1.1 implementation is allowed to have a vendorspecific deployment and packaging model for servlet-based service endpoints. For this reason, the JAX-RPC TCK provides two distribution directories, /dist and /dist-vi, for the web service endpoint tests.

The /dist directory contains all the WAR files for the JAX-RPC TCK service endpoint tests that have been compiled and packaged for deployment on a Servlet-compliant web container using the standard Web Archive (WAR) format. These WAR files are tailored for use with the JAX-RPC 1.1 Reference Implementation. The web.xml file contains the deployment information that the web container needs to link JAX-RPC service endpoints to its associated servlet class. One generic servlet class, com.sun.xml.rpc.server.http.JAXRPCServlet , is used for all JAX-RPC service endpoints. These WAR files enable you to run, without any additional setup or modification, the JAX-RPC TCK tests and to test the various features and functionality of the JAX-RPC 1.1 Reference Implementation.

The /dist-vi directory contains generic JAR files for all the TCK service endpoint tests that you must use to package and deploy on your JAX-RPC 1.1 implementation in a vendor-specific way. For each of these web applications, you must use your deployment tool to perform the appropriate Java-to-WSDL or WSDL-to-Java mapping, based on whether the service endpoint is defined by a Java Service Endpoint Interface or a WSDL service document. Next, you must package all the generated artifacts needed along with the service implementation classes for the service endpoints and deploy them on your container-based implementation.

The following README files and TABLE show test breakdown and provide information that should help you with this process

- README.JAVA-TO-WSDL

- README.WSDL-TO-JAVA

- Test Breakdown and Mapping Requirements

These README files contain the manual instructions that you need to perform to implementation. The files also specify the mapping (Java-to-WSDL or WSDL-to- deploy and package the service endpoint tests on your JAX-RPC 1.1 Java) that each test must perform. + For a brief example of how the JAX-RPC 1.1 Reference Implementation does this, look at the J2WMT.war web application in the the /dist directory. This test marshalls/unmarshalls all the JAX-RPC-supported Java data types as outlined in the JAX-RPC 1.1 Specification using the Java-to-WSDL mapping. The web.xml file that is packaged in this web application is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://
java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<display-name>J2WMarshallTestApp</display-name>
<listener>
<listenerclass>
com.sun.ts.tests.jaxrpc.common.JAXRPCContextListener</
listener-class>
</listener>
<servlet>
<servlet-name>XmlServletName</servlet-name>
<servlet-class>com.sun.ts.tests.jaxrpc.common.JAXRPCServlet</
servlet-class>
<load-on-startup>0</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>XmlServletName</servlet-name>
<url-pattern>/jaxrpc/*</url-pattern>
</servlet-mapping>
<session-config>
<session-timeout>54</session-timeout>
</session-config>
</web-app>
```

As you can see, the JAX-RPC service endpoint is linked to a generic servlet in the JAX-RPC RI (com.sun.xml.rpc.server.http.JAXRPCServlet). This servlet is abstracted out in JAX-RPC TCK under common servlet (com.sun.ts.tests.jaxrpc.common.JAXRPCServlet) which simply extends it. This common servlet is used for processing, handling and dispatching to all JAX-RPC service endpoints. The JAX-RPC TCK Service Endpoint WAR files use a consistent naming convention for accessing all Service Endpoint URLs and WSDL URLs: as follows.

- For Service Endpoint URLs, the URL mapping follows this convention: http://webServerHost:webServerPort/ContextRoot/URLAlias/PortName
  The URL Alias used in all JAX-RPC TCK tests is jaxrpc.

- For WSDL URL locations, the URL mapping follows this convention: http://webServerHost:webServerPort/ContextRoot/URLAlias?WSDL

The example that follows is one of the JAX-RPC TCK tests that shows this mapping.

The following implementation-specific runtime configuration file, jaxrpc-riruntime. xml , is supplied as a web archive initialization file to this servlet. This runtime configuration file is automatically generated by the wsdeploy deployment tool, which gives information on all service endpoints containing their interface, implementation and tie classes, their service, port and WSDL document and

their URL pattern for accessing the endpoint.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns='http://java.sun.com/xml/ns/jax-rpc/ri/runtime' version='1.0'>
<endpoint
  name='MarshallTest'
  interface='com.sun.ts.tests.jaxrpc.ee.j2w.marshalltest.Marshall Test'
  implementation='com.sun.ts.tests.jaxrpc.ee.j2w.marshalltest.MarshallTestImpl'
  tie='com.sun.ts.tests.jaxrpc.ee.j2w.marshalltest.MarshallTest_Tie'
  wsdl='/WEB-INF/MarshallTestService.wsdl'
  service='{http://marshalltestservice.org/wsdl}MarshallTestService'
  port='{http://marshalltestservice.org/wsdl}MarshallTestPort'
  urlpattern='/jaxrpc/MarshallTest'/>
</endpoints>
```

The RPC service endpoint has one port defined in its runtime configuration file. The test client code accesses this JAX-RPC service endpoint and port at the following URL location: http://webServerHost:webServerPort/J2WMT/jaxrpc/MarshallTest

where MarshallTest is the service port to access.

The above RPC service endpoint WSDL file describing this service is also published. The test client code accesses this WSDL file based on the above configuration via the following URL location: http://webServerHost:webServerPort/J2WMT/jaxrpc/MarshallTest?WSDL

## 4.2.2 Instructions for Interoperability Test Scenarios

To test for interoperability between your implementation and the JAX-RPC Reference Implementation, we recommend the following two test scenarios:

- J2WMT
- W2JMarshallTest

Complete the following steps to test interoperability between JAX-RPC implementations in both directions:

1. Deploy and run the J2WMT and W2JMarshallTest TCK tests on your own JAX-RPC 1.1 implementation and perform the appropriate mappings (WSDL-to-Java) and (Java-to-WSDL)

2. Deploy the JAX-RPC RI WAR files (J2WMT.war) and (W2JMarshallTest.war) on a remote host system using the JAX-RPC 1.1 Reference Implementation.

3. On your local host system that is running your own JAX-RPC implementation, edit the ts.jte file to change the Web server host and port to point to this remote system running the JAX-RPC RI with the J2WMT.war and W2JMarshallTest.war WAR files deployed.

4.  Run the client code for these tests against the remote system to test the marshalling/unmarshalling of data from your JAX-RPC implementaton to the JAX-RPC RI implementation system and back.

5.  Perform the same procedure going in the opposite direction, running the client on the JAX-RPC RI and accessing the Service Endpoint deployed on your JAX-RPC implementation.

# 4.4 Custom Configuration Handlers

Configuration handlers are used to configure and unconfigure a JAX-RPC 1.1 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, see `<TS_HOME>/bin/xml/impl/glassfish/config.vi.xml`, the configuration file for the Java EE 8 RI.

# 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 JAX-RPC TCK provides these deployment handlers: * `<TS_HOME>/bin/xml/impl/none/deploy.xml` * `<TS_HOME>/bin/xml/impl/glassfish/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 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 JAX-RPC implementation, you must create a custom handler. 1. Create a new directory in the <TS_HOME>/bin/impl directory tree. For example, create the <TS_HOME>/bin/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".

1. Copy the deploy.xml file from the <TS_HOME>/bin/xml/impl/none directory to the directory that you created.

2. 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 glassfish directories for guidance as you create the same targets for the Web container in which you will run your implementation of JAX-RPC.

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 JAX-RPC 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 JAX-RPC TCK test environment. This section covers the following topics:

- Configuration GUI Overview

- Starting the Configuration GUI

- To Configure the JavaTest Harness to Run the JAX-RPC 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 8 installed on your system.

The JAX-RPC 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 JAX-RPC 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 JAX-RPC 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 the RI

- 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.

As explained in Appendix B, "Packaging the Test Applications in Servlet-Compliant WAR Files With VI-Specific Information," the JAX-RPC TCK introduces the concept of repackaging the TCK tests.

## 5.1 Starting JavaTest

There are two general ways to run the JAX-RPC TCK using the JavaTest harness software:

- Through the JavaTest GUI

- From the command line in your shell environment

The `<TS_HOME>/tools/ant/bin/ant` command referenced in the following two procedures and elsewhere in this guide is the Ant build tool, which is included in the JAX-RPC TCK bundle. The `build.xml` file in `<TS_HOME>/bin` contains the various Ant targets for the JAX-RPC 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 JAX-RPC TCK Signature Tests

To run the JAX-RPC TCK signature tests, enter the following commands:

```
cd <TS_HOME>/src/com/sun/ts/tests/signaturetest/jaxrpc
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/jaxrpc/api/client
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/jaxrpc/api
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"

## 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 `<TS_HOME>/tools/ant/bin/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 JAX-RPC 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 the RI

This test scenario is ensures that the configuration and deployment of all the prebuilt JAX-RPC TCK tests against the Reference Implementation are successful, and that the TCK is ready for compatibility testing against the Vendor and Reference Implementations.

1. Verify that you have followed the configuration instructions in Section 4.1, "Configuring Your Environment to Run the TCK Against the Reference 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:

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

## 6.1 Overview

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

- Errors: Tests with errors could not be executed by the 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.

> ℹ️ 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).

## 6.7 Troubleshooting Tips

This section provides some tips for troubleshooting errors that may be encountered.

- Verify that the Ant `config.vi` and `enable.jacc` configuration targets executed correctly.

- If there are several failures during a test run, check the various output for hints about what caused the failures. A common problem is the absence of the JAX-RPC log file. This log file should be created in the directory defined by the `log.file.location` property in the `ts.jte` file. There should be a log file called `JACCLog.txt` in this directory. The `JACCLog.txt` consists of record entries containing permission infomation that will be used to verify the TCK tests for proper compliance.

This log file typically gets populated with JAX-RPC-based security information when test archives are deployed. Then, during test execution, the `JACCLog.txt` file is read and used for validating that JAX-RPC behavior is correct.

- Simultaneously deploying all JAX-RPC TCK test archives may cause false failures. If unexpected failures occur during a TCK run when all JAX-RPC archives were deployed, these failures could be caused by interference from tests and archives that are defined multiple times. If such situational failures do occur, undeploy all archives, remove the `JACCLog.txt` file, recycle your server (if necessary), and rerun only the tests in the directory that showed failures.

- Check that the following JVM variables, which should have been set by invoking the `enable.jacc` Ant target, have been set in the application server :

  ◦ `-Dlog.file.location` (this comes from the `<TS_HOME>/bin/ts.jte` property)

  ◦ `-Djavax.security.jacc.policy.provider=com.sun.ts.tests.jacc.provider.TSPolicy`

  ◦ `-Djavax.security.jacc.PolicyConfigurationFactory.provider=com.sun.ts.tests.jacc.provider.TSPolicyConfigurationFactoryImpl`

  ◦ `-Dvendor.javax.security.jacc.policy.provider=com.sun.enterprise.security.provider.PolicyWrapper`

  ◦ `-Dvendor.javax.security.jacc.PolicyConfigurationFactory.provider=com.sun.enterprise.security.provider.PolicyConfigurationFactoryImpl`

  > ℹ️ The values for the `-Dvendor.javax.security.jacc.policy.provider` and `-Dvendor.javax.security.jacc.PolicyConfigurationFactory.provider` JVM variables will need to be set specific to the application server in use.

# A Frequently Asked Questions

This appendix contains the following questions.

- Where do I start to debug a test failure?

- How do I restart a crashed test run?

- What would cause tests be added to the exclude list?

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

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

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

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

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

The 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 reference 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.

# B Packaging the Test Applications in Servlet-Compliant WAR Files With VI-Specific Information

The JAX-RPC 1.1 specification specifies how JAX-RPC applications are to be published in a Java SE environment, JAX-WS endpoint, or Servlet–compliant Web container.

JAX-RPC TCK test application classes that are to be published in a Java SE environment are located under `$TS_HOME/classes`.

The JAX-RPC TCK comes with prebuilt test WAR files for deployment on Java EE 8 RI , which provides a Servlet–compliant Web container. The WAR files are Eclipse Glassfish 5.0-specific, with Eclipse Glassfish 5.0's servlet class and Eclipse Glassfish 5.0's servlet defined in the `web.xml` deployment descriptor. To run the TCK tests against the VI in a Servlet–compliant Web container, the tests need to be repackaged to include the VI-specific servlet, and the VI-specific servlet must be defined in the deployment descriptor.

The JAX-RPC TCK makes it easier for the vendor by including template WAR files that contain all of the necessary files except for the VI-specific servlet adaptor class. The JAX-RPC TCK provides a tool to help with the repackaging task.

This appendix contains the following sections:

- Overview
- Creating the VI-Specific Servlet–Compliant WAR Files

## B.1 Overview

The set of prebuilt archives and classes that ship with the JAX-RPC TCK were built using the Reference Implementation, and must be deployed on Java EE 8 RI and run against the Eclipse Glassfish 5.0 RI.

The prebuilt Eclipse Glassfish 5.0-specific Servlet–compliant WAR files are located under $TS_HOME`/dist`, and have `jersey` as part of their name; for example:

```
$TS_HOME/dist/com/sun/ts/tests/jaxrs/ee/rs/get/jaxrs_rs_get_web.war.jersey
```

The names are made unique by including `jersey` to minimize the chances that the files will be overwritten or modified.

The Vendor must create VI-specific Servlet–compliant WAR files if the Vendor chooses to publish on a

Servlet–compliant Web container, so that the VI-specific Servlet class will be included instead of the Eclipse Glassfish 5.0-specific Servlet class.

# B.2 Creating the TCK VI-Specific Servlet–Compliant WAR Files

All resource and application class files are already compiled. The Vendor needs to package these files. All tests also come with a `web.xml.template` file to be used for generating deployment descriptor files with a VI-specific Servlet definition.

Each test that has a JAX-RPC 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>javax.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. A Eclipse Glassfish 5.0-specific deployment descriptor also comes with the JAX-RPC TCK, and has the values for the `com.sun.jersey.spi.container.servlet.ServletContainer`:

```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>
            org/glassfish/jersey/servlet/ServletContainer
        </servlet-class>
        <init-param>
            <param-name>javax.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 JAX-RPC TCK provides a tool, `${ts.home}/bin/xml/impl/glassfish/jersey.xml`, for the Java EE 8 RI that you can use as a model to help you create your own VI-specific Web test application.

## B.2.1 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. Make sure the `jaxrs_impl_name` property is set in the `<TS_HOME>/bin/ts.jte`, file and that it has a unique name so no file will be overwritten.

2. Set the VI Servlet class property.
   Set the `servlet_adaptor` property in the `<TS_HOME>/bin/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 under `$TS_HOME/dist`; for example:

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

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