TCK User’s Guide for Technology Implementors

Table of Contents

Eclipse Foundation

Technology Compatibility Kit User's Guide for Jakarta XML Web Services

Release 3.0 for Jakarta EE

September 2020

References in this document to JWB refer to the Jakarta XML Web Services 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 XML Web Services (XML Web Services 3.0) technology.

The XML Web Services TCK is a portable, configurable automated test suite for verifying the compatibility of a vendor's implementation of the XML Web Services 3.0 Specification (hereafter referred to as the vendor implementation or VI). The XML Web Services 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 wombat-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 XML Web Services 3.0 technology to assist them in running the test suite that verifies compatibility of their implementation of the XML Web Services 3.0 Specification.

Before You Read This Book

You should be familiar with the XML Web Services 3.0, version 3.0 Specification, which can be found at https://jakarta.ee/specifications/xml-web-services/3.0/.

Before running the tests in the XML Web Services 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 XML Web Services TCK (XML Web Services 3.0 TCK). It also includes a high level listing of what is needed to get up and running with the XML Web Services 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 XML Web Services TCK 3.0 is designed as a portable, configurable, automated test suite for verifying the compatibility of a vendor's implementation of the XML Web Services 3.0 Specification.

1.2.1 TCK Specifications and Requirements

This section lists the applicable requirements and specifications.

- Specification Requirements: Software requirements for a XML Web Services implementation are described in detail in the XML Web Services 3.0 Specification. Links to the XML Web Services specification and other product information can be found at https://jakarta.ee/specifications/xml-web-services/3.0/.
- XML Web Services Version: The XML Web Services 3.0 TCK is based on the XML Web Services Specification, Version 3.0.
- Compatible Implementation: One XML Web Services 3.0 Compatible Implementation, Java EE 8 RI is available from the Eclipse EE4J project (https://projects.eclipse.org/projects/ee4j). See the CI documentation page at https://eclipse-ee4j.github.io/glassfish/ for more information.

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

1.2.2 TCK Components

The XML Web Services TCK 3.0 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.
- XML Web Services TCK signature tests; check that all public APIs are supported and/or defined as specified in the XML Web Services Version 3.0 implementation under test.
- If applicable, an exclude list, which provides a list of tests that your implementation is not required to pass.
- API tests for all of the XML Web Services API in all related packages:
  - `jakarta.xml.ws`
  - `jakarta.xml.ws.handler`

  - `jakarta.xml.ws.handler.soap`
  - `jakarta.xml.ws.http`
  - `jakarta.xml.ws.soap`
  - `jakarta.xml.ws.spi`
  - `jakarta.xml.ws.spi.http`
  - `jakarta.xml.ws.wsaddressing`

The XML Web Services TCK tests run on the following platforms:

- CentOS 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 XML Web Services TCK 3.0 to test a XML Web Services 3.0 implementation. The tests are designed to verify that a vendor's runtime implementation of the technology complies with the appropriate specification. The individual tests correspond to assertions of the specification.

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

1.2.5 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 XML Web Services 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 XML Web Services 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 XML Web Services 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 Java EE 8 RI 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 XML Web Services 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
- A CI for XML Web Services 3.0. One example is Java EE 8 RI.
- XML Web Services TCK version 3.0, which includes:
  - JDOM 1.1.3
  - Apache Commons HTTP Client 3.1
  - Apache Commons Logging 1.1.3
  - Apache Commons Codec 1.9
- The XML Web Services 3.0 Vendor Implementation (VI)
  See the documentation for each of these software applications for installation instructions. See Chapter 3, "Installation," for instructions on installing the XML Web Services TCK.
  1. Set up the XML Web Services 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 XML Web Services 3.0 implementation.
     Test the XML Web Services 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 XML Web Services. This chapter contains the following sections:

- Certification Overview
- Compatibility Requirements
- Test Appeals Process
- Specifications for Jakarta XML Web Services
- Libraries for Jakarta XML Web Services

2.1 Certification Overview

The certification process for XML Web Services 3.0 consists of the following activities:

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

2.2 Compatibility Requirements

The compatibility requirements for XML Web Services 3.0 consist of meeting the requirements set forth by the rules and associated definitions contained in this section.

2.2.1 Definitions

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

Table 2-1 Definitions

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

Term	Definition
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 XML Web Services Version 3.0.
Test Suite	The requirements, tests, and testing tools distributed by the Maintenance Lead as applicable to a given Version of the Technology.
Version	A release of the Technology, as produced through the Jakarta EE Specification Process (JESP).
Development Kit	A software product that implements or incorporates a Compiler, a Schema Compiler, a Schema Generator, a Java-to-WSDL Tool, a WSDL-to-Java Tool, and/or an RMI Compiler.
Java-to-WSDL Output	Output of a Java-to-WSDL Tool that is required for Web service deployment and invocation.
Java-to-WSDL Tool	A software development tool that implements or incorporates a function that generates web service endpoint descriptions in WSDL and XML schema format from Source Code as specified by the Jakarta XML Web Services Specification.
WSDL-to-Java Output	Output of a WSDL-to-Java tool that is required for Web service deployment and invocation.
WSDL-to-Java Tool	A software development tool that implements or incorporates a function that generates web service interfaces for clients and endpoints from a WSDL description as specified by the Jakarta XML Web Services Specification.

2.2.2 Rules for Jakarta XML Web Services Products

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

XMLWS1 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.

XMLWS1.1 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.

XMLWS1.2 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.

XMLWS1.3 An API Definition Product is exempt from all functional testing requirements defined here, except the signature tests.

XMLWS2 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.

XMLWS3 The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

XMLWS4 The Exclude List associated with the Test Suite cannot be modified.

XMLWS5 The Maintenance Lead can define exceptions to these Rules. Such exceptions would be made available as above, and will apply to all vendor implementations.

XMLWS6 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.

XMLWS7 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.

XMLWS7.1 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.

XMLWS8 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.

XMLWS9 The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.

XMLWS10 Source Code in WSDL-to-Java Output when compiled by a Reference Compiler must execute properly when run on a Reference Runtime.

XMLWS11 Source Code in WSDL-to-Java Output must be in source file format defined by the Java Language Specification (JLS).

XMLWS12 Java-to-WSDL Output must fully meet W3C requirements for the Web Services Description Language (WSDL) 1.1.

XMLWS13 A Java-to-WSDL Tool must not produce Java-to-WSDL Output from source code that does not conform to the Java Language Specification (JLS).

2.3 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 XML Web Services 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 XML Web Services 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 XML Web Services TCK.

2.3.3 TCK Test Appeals Steps

1. Challenges should be filed via the Jakarta XML Web Services 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.committee@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 XML Web Services

The Jakarta XML Web Services specification is available from the specification project web-site: https://jakarta.ee/specifications/xml-web-services/3.0/.

2.5 Libraries for Jakarta XML Web Services

The following is a list of the packages comprising the required class libraries for XML Web Services 3.0:

- `jakarta.xml.ws`
- `jakarta.xml.ws.handler`
- `jakarta.xml.ws.handler.soap`
- `jakarta.xml.ws.http`
- `jakarta.xml.ws.soap`
- `jakarta.xml.ws.spi`
- `jakarta.xml.ws.spi.http`
- `jakarta.xml.ws.wsaddressing`

For the latest list of packages, also see:

https://jakarta.ee/specifications/xml-web-services/3.0/

3 Installation

This chapter explains how to install the Jakarta XML Web Services 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. Java EE 8 RI is a compatible implementation which may be obtained from https://eclipse-ee4j.github.io/glassfish/

3.2 Installing the Software

Before you can run the XML Web Services 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
- A CI for XML Web Services 3.0, one example is Java EE 8 RI
- XML Web Services TCK version 3.0, which includes:
  - JDOM 1.1.3
  - Apache Commons HTTP Client 3.1
  - Apache Commons Logging 1.1.3
  - Apache Commons Codec 1.9
- The XML Web Services 3.0 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 XML Web Services TCK 3.0 software.
   1. Copy or download the XML Web Services TCK software to your local system.
      You can obtain the XML Web Services TCK software from the Jakarta EE site https://jakarta.ee/

specifications/xml-web-services/3.0/.

2. Use the `unzip` command to extract the bundle in the directory of your choice:
   `unzip xml-ws-tck-3.0.0_dd-Mmm-YYYY.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 XML Web Services TCK with the XML Web Services 3.0 RI), if it is not already installed.
   Download and install the Servlet Web container with the XML Web Services 3.0 RI used for running the XML Web Services TCK 3.0, 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 Java EE 8 RI software which contains the JAX-WS 3.0 Reference Implementation or install the Standalone JAX-WS 3.0 RI software.
   The Java EE 8 RI software contains the JAX-WS 3.0 Reference Implementation and is used to validate your initial configuration and setup of the JAX-WS TCK 3.0, as well as to run the reverse tests which are explained further in Chapter 4, "Setup and Configuration" and Appendix B.
   The Standalone JAX-WS 3.0 RI software contains the JAX-WS 3.0 Reference Implementation and can be used with any web container that meets the minimum requirements for a container as defined in the JAX-WS 3.0 Specification such as the Apache Tomcat web container.
   If you use the Standalone JAX-WS 3.0 RI software with the Apache Tomcat web container, you need to copy the JAR files/classes from the standalone JAX-WS 3.0 RI software to the correct location under the Apache Tomcat web container. Assuming the Apache Tomcat web container is installed under `${tomcat.home}`, you would copy the JAR files/classes as follows:

```
cp jaxws-api.jar jaxb-api.jar ${tomcat.home}/common/endorsed

cp commonj.sdo.jar eclipselink.jar FastInfoset.jar gmbal-api-only.jar \
    ha-api.jar jakarta.annotation-api.jar jakarta.persistence.jar \
    javax.xml.soap-api.jar jaxb-core.jar jaxb-impl.jar jaxb-jxc.jar \
    jaxb-xjc.jar jaxws-eclipselink-plugin.jar jaxws-rt.jar jaxwstck.jar \
    jaxws-tools.jar jsr181-api.jar mail.jar management-api.jar mimepull.jar \
    policy.jar resolver.jar saaj-impl.jar sdo-eclipselink-plugin.jar \
    stax2-api.jar stax-ex.jar streambuffer.jar tsharness.jar \
    woodstox-core-asl.jar ${tomcat.home}/shared/lib

cp jaxwstck/lib/tsharness.jar jaxwstck/lib/jaxwstck.jar ${tomcat.home}/shared/lib
```

   If, instead of using Tomcat, you are using the Java EE 8 Reference Implementation, which already includes JAX-WS 3.0, you only need to copy the `jaxwstck/lib/tsharness.jar` and `jaxwstck/lib/jaxwstck.jar` TCK JAR files to the domain's `lib/ext` directory.

5. Install a XML Web Services 3.0 Compatible Implementation.
   A Compatible Implementation is used to validate your initial configuration and setup of the XML Web Services TCK 3.0 tests, which are explained further in Chapter 4, "Setup and Configuration."

The Compatible Implementations for XML Web Services are listed on the Jakarta EE Specifications web site: https://jakarta.ee/specifications/xml-web-services/3.0/.

6. Install a Web server on which the XML Web Services TCK test applications can be published for testing the VI.
7. Install the XML Web Services 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 XML Web Services 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 XML Web Services 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 4.6, "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.

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 XML Web Services TCK 3.0 software is installed
    3. `PATH` to include the following directories: `JAVA_HOME/bin`, `JAXWS_HOME/bin`, and `<TS_HOME>/tools/ant/bin`
2. Edit your `<TS_HOME>/bin/ts.jte` file and set the following environment variables:
    1. Set the `webServerHost` property to the hostname where the web server is running that is configured with the JAX-WS RI.
       The default setting is `localhost`.

2. Set the `webServerPort` property to the port number where the web server is running that is configured with the JAX-WS RI.
   The default setting is `8000`.

3. Set the `jaxws.home` property to the location where the JAX-WS RI is installed.
   The default setting is `${webcontainer.home}`. The value of this property must match the value of the `webcontainer.home` property that is set in step g.

4. Set the `jaxws.classes` property to point to the JAX-WS RI classes/jars.
   Note that this property is already set and should not require any changes.

5. Verify that the `tools.jar` property is set to the location of the `tools.jar` file that is distributed with Java SE.

6. Set the `impl.vi, impl.vi.deploy.dir, impl.vi.host, and impl.vi.port` properties to the supported web container, deploy directory, host and port for the JAX-WS RI.
   The supported web container for the JAX-WS RI is the Java EE 8 RI. So the default settings for these properties are as follows:

   ```
   impl.vi.deploy.dir=${webcontainer.home}/domains/domain1/autodeploy
   impl.vi=glassfish.xml
   impl.vi.host=${webServerHost}
   impl.vi.port=${webServerPort}
   ```

7. Set the `webcontainer.home` property to the location where the web server is installed for the JAX-WS RI. This will be where the Java EE 8 RI is installed.

8. Set the `porting.ts.url.class.1` property to the porting implementation class that is used for obtaining URLs.
   The default setting for the JAX-WS RI porting implementation is:

   ```
   com.sun.ts.lib.implementation.sun.common.SunRIURL
   ```

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

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

11. Set the `http.server.supports.endpoint.publish` property based on whether Endpoint Publish APIs are supported on the container.

12. If using Java SE 8 or above, verify that the property `endorsed.dirs` is set to the location of the VI API jars for those technologies you wish to override. Java SE 8 contains an implementation of JAX-WS 2.2 which will conflict with JAX-WS 3.0, therefore this property must be set so that JAX-WS 3.0 will be used during the building of tests and during test execution.

3. Edit the catalog file `<TS_HOME>/src/com/sun/ts/tests/jaxws/common/xml/catalog /META-INF/jax-ws-catalog.xml`, replacing the host and port settings of `systemId` with the value of your host and port setting where the WSDL is published.

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 4.4, "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.

Before You Begin

Decide against which XML Web Services implementation the tests will be run and determine to which Servlet–compliant Web server the XML Web Services TCK applications will be published.

Package the XML Web Services test applications for that XML Web Services implementation and Servlet–compliant Web server.

See Appendix B, "Packaging the Test Applications in Servlet-Compliant WAR Files With VI-Specific Information," for information about repackaging the XML Web Services test application.

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 XML Web Services TCK 3.0 software is installed
    3. `PATH` to include the following directories: `JAVA_HOME/bin`, `JAXWS_HOME/bin`, and `<TS_HOME>/tools/ant/bin`
2. Edit your `<TS_HOME>/bin/ts.jte` file and set the following environment variables:
    1. Set the `webServerHost` property to the hostname where the web server that is configured with the Vendor Implementation is running.
       The default setting is `localhost`.

2. Set the `webServerPort` property to the port number where the web server that is configured with the Vendor Implementation is running.
   The default setting is `8080`.
3. Set the `jaxws.home` property to the location where the Vendor Implementation is installed.
   The default setting is `${webcontainer.home}`. The value of this property must match the value of the `webcontainer.home` property that is set in step g.
4. Set the `jaxws.classes` property to point to the Vendor Implementation classes/JAR files.
   As an example, the `ts.jte` file contains the property `jaxws.classes.ri`, which contains the classes/jar files that the Java EE 8 RI uses. The `jaxws.classes.ri` settings for the Java EE 8 RI web container are as follows:

```
jaxws.home.ri=${webcontainer.home.ri}
jaxws.lib.ri=${jaxws.home.ri}/modules
endorsed.dirs.ri=${jaxws.home.ri}/modules/endorsed

jaxws.classes.ri=${endorsed.dirs.ri}/webservices-api-osgi.jar:
${endorsed.dirs.ri}/jaxb-api-osgi.jar:
${jaxws.lib.ri}/webservices-osgi.jar:
${jaxws.lib.ri}/jaxb-osgi.jar:
${jaxws.lib.ri}/gmbal.jar:
${jaxws.lib.ri}/management-api.jar:
${jaxws.lib.ri}/mimepull.jar
```

   The `jaxws.classes.ri` settings if using the Apache Tomcat web container with the Standalone JAX-WS 3.0 RI would be as follows:

```
jaxws.home.ri=${webcontainer.home.ri}
jaxws.lib.ri=${jaxws.home.ri}/shared/lib
endorsed.dirs.ri=${jaxws.home.ri}/common/endorsed
jaxws.classes.ri=${endorsed.dirs.ri}/jaxws-api.jar:
${endorsed.dirs.ri}/jaxb-api.jar:
${jaxws.lib.ri}/FastInfoset.jar:${jaxws.lib.ri}/gmbal-api-only.jar:\
${jaxws.lib.ri}/ha-api.jar:${jaxws.lib.ri}/jakarta.annotation-api.jar:\
${jaxws.lib.ri}/javax.xml.soap-api.jar:${jaxws.lib.ri}/jaxb-core.jar:\
${jaxws.lib.ri}/jaxb-impl.jar:${jaxws.lib.ri}/jaxb-jxc.jar:\
${jaxws.lib.ri}/jaxb-xjc.jar:${jaxws.lib.ri}/jaxws-rt.jar:\
${jaxws.lib.ri}/jaxwstck.jar:${jaxws.lib.ri}/jaxws-tools.jar:\
${jaxws.lib.ri}/jsr181-api.jar:${jaxws.lib.ri}/mail.jar:\
${jaxws.lib.ri}/management-api.jar:${jaxws.lib.ri}/mimepull.jar:\
${jaxws.lib.ri}/policy.jar:${jaxws.lib.ri}/resolver.jar:\
${jaxws.lib.ri}/saaj-impl.jar:${jaxws.lib.ri}/sdo-eclipselink-plugin.jar:\
${jaxws.lib.ri}/stax2-api.jar:${jaxws.lib.ri}/stax-ex.jar:\
${jaxws.lib.ri}/streambuffer.jar:${jaxws.lib.ri}/woodstox-core-asl.jar:
```

5. Verify that the `tools.jar` property is set to the location of the `tools.jar` file that is distributed with Java SE.
6. Set the `impl.vi`, `impl.vi.deploy.dir`, `impl.vi.host`, and `impl.vi.port` properties to the supported web container, deploy directory, host and port for the Vendor Implementation.
   As an example, the `ts.jte` file contains the settings for the Java EE 8 RI, which are as follows:

   ```
   webcontainer.home.ri=/sun/javaee6
   impl.ri.deploy.dir=${webcontainer.home.ri}/domains/domain1/autodeploy
   impl.ri=glassfish.xml
   impl.ri.host=${webServerHost.2}
   impl.ri.port=${webServerPort.2}
   ```

   The RI settings using the Apache Tomcat web container with the Standalone JAX-WS 3.0 RI would be as follows:

   ```
   webcontainer.home.ri=/tomcat
   impl.ri=tomcat
   impl.ri.deploy.dir=${webcontainer.home.ri}/webapps
   impl.ri.host=${webServerHost.2}
   impl.ri.port=${webServerPort.2}
   ```

7. Set the `webcontainer.home` property to the location where the web container for the Vendor Implementation is installed.
8. Set the `porting.ts.url.class.1` property to the porting implementation class that is used for obtaining URLs.
   The default setting for the JAX-WS RI porting implementation is:

   ```
   com.sun.ts.lib.implementation.sun.common.SunRIURL
   ```

9. Set the `user` and `password` properties to the user name and password used for the basic authentication tests.
   The default setting for both is `j2ee`.
10. Set the `authuser` and `authpassword` properties to the user name and password used for the basic authentication tests.
    The default setting for both is `javajoe`.
11. Set the `http.server.supports.endpoint.publish` property based on whether Endpoint Publish APIs are supported on the container.
12. If using Java SE 8 or above, verify that the property `endorsed.dirs` is set to the location of the RI API jars for those technologies you wish to override. Java SE 8 contains an implementation of JAX-WS 2.2 which will conflict with JAX-WS 3.0, therefore this property must be set so that JAX-

WS 3.0 will be used during the building of tests and during test execution.

3. Edit the catalog file `<TS_HOME>/src/com/sun/ts/tests/jaxws/common/xml/catalog /META-INF/jax-ws-catalog.xml`, replacing the host and port settings of `systemId` with the value of your host and port setting where the WSDL is published.

4. Provide your own implementation of the porting package interface provided with the JAX-WS TCK. The porting package interface, `TSURLInterface.java`, obtains URL strings for web resources in an implementation-specific manner. API documentation for the `TSURLInterface.java` porting package interface is available in the JAX-WS TCK documentation bundle.
   The `<TS_HOME>/bin/jaxws-url-props.dat` file contains the webservice endpoint and WSDL URLs that the TCK tests use when running against the JAX-WS RI. In the porting package that the TCK uses, the URLs are returned as is since this is the form that the JAX-WS RI expects. You may need an alternate form of these URLs in order to run the TCK tests in your environment. However, you MUST NOT modify the `jaxws-url-props.dat` file, but instead make any necessary changes in your own porting implementation class to transform the URLs appropriately for your environment.

4.3 Additional JAX-WS TCK Instructions

4.3.1 Configuring Your Environment to Rebuild and Run the JAX-WS TCK Against the JAX-WS RI

This section describes the steps needed to configure the JAX-WS TCK so that the tests can be rebuilt (using the Vendor Implementation toolset), and then deployed and run against the JAX-WS Reference Implementation.

If you are not ready to proceed with this portion of the testing process, skip this section for now and proceed to Configuring and Starting Your Application Server or Servers. After configuring your environment, continue with the instructions in 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 XP/Vista. In addition, the forward slashes (`/`) used in all of the examples need to be replaced with backslashes (`\`) for Windows XP/Vista. 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. For example, if the Java EE 8 RI installation is `C:\JavaEE8`, you must specify it as `C:\\JavaEE8` or `C:/JavaEE8`.

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

   1. `JAVA_HOME` to the directory in which Java SE is installed
   2. `TS_HOME` to the directory in which the JAX-WS TCK 3.0 software is installed
   3. `ANT_HOME` should not be set in your environment. If it is set, unset it.
2. Edit your `<TS_HOME>/bin/ts.jte` file and set the following environment variables:
   1. Set the `webServerHost` property to the hostname where the web server for the Vendor Implementation is running.
      The default setting is `localhost`.
   2. Set the `webServerPort` property to the port number where the web server for the Vendor Implementation is running.
      The default setting is`8080`
   3. Set the `webServerHost.2` property to the hostname where the web server for the JAX-WS RI is running.
   4. Set the `webServerPort.2` property to the port number where the web server for the JAX-WS RI is running.
   5. Set the `jaxws.home` property to the location where the Vendor Implementation is installed.
      The default setting is `${webcontainer.home}`. The value of this property must match the value of the `webcontainer.home` property that is set in step m.
   6. Set the `jaxws.classes` property to point to the Vendor Implementation classes/JAR files.
   7. Set the `jaxws.home.ri` property to the location where the JAX-WS RI is installed.
      The default setting is `${webcontainer.home.ri}`. The value of this property must match the value of the `webcontainer.home.ri` property that is set in step n.
   8. The `jaxws.classes.ri` property is already configured to point to the JAX-WS RI classes/JAR files.
      No changes are necessary for this property.
   9. Set the `wsgen.ant.classname` property to the Vendor Implementation class that mimics the JAX-WS RI Ant task, which in turn calls the `wsgen` Java-to-WSDL tool.
   10. Set the `wsimport.ant.classname` property to the Vendor Implementation class that mimics the JAX-WS RI Ant task, which in turn calls the `wsimport` WSDL-to-Java tool.
   11. Set the `impl.vi`, `impl.vi.deploy.dir`, `impl.vi.host`, and `impl.vi.port` properties to the supported web container, deploy directory, host and port used for the Vendor Implementation.
   12. Set the `impl.ri`, `impl.ri.deploy.dir`, `impl.ri.host`, and `impl.ri.port` properties to the supported web container, deploy directory, host and port used for the JAX-WS Reference Implementation.
       The supported web container for standalone web applications is the Java EE 8 RI. The default settings for these properties are as follows:

       ```
       impl.ri.deploy.dir=${webcontainer.home.ri}/domains/domain1/autodeploy
       impl.ri=glassfish.xml
       impl.ri.host=${webServerHost.2}
       impl.ri.port=${webServerPort.2}
       ```

   13. Set the `webcontainer.home` property to the location where the web container for the Vendor Implementation is installed.
   14. Set the `webcontainer.home.ri` property to the location where the web container for the JAX-WS Reference Implementation is installed.
   15. Set the `porting.ts.url.class.1` property to your porting implementation class that is used for obtaining URLs.
       The default setting points to the JAX-WS RI porting implementation which is:

       ```
       com.sun.ts.lib.implementation.sun.common.SunRIURL
       ```

   16. Set the `porting.ts.url.class.2` property to the JAX-WS RI porting implementation class that is used for obtaining URLs.
       No changes are necessary for this property.
   17. Set the `user` and `password` properties to the user name and password used for the basic authentication tests.
       The default setting for both is `j2ee`.
   18. Set the `authuser` and `authpassword` properties to the user name and password used for the basic authentication tests.
       The default setting for both is `javajoe`.
   19. Set the `http.server.supports.endpoint.publish` property based on whether Endpoint Publish APIs are supported on the container.
   20. If using Java SE 8 or above, verify that the property `endorsed.dirs` is set to the location of the VI API jars for those technologies you wish to override. Java SE 8 contains an implementation of JAX-WS 2.2 which will conflict with JAX-WS 3.0, therefore this property must be set so that JAX-WS 3.0 will be used during the building of tests and during test execution.
   21. If using Java SE 8 or above, verify that the property `endorsed.dirs.ri` is set to the location of the RI API jars for those technologies you wish to override. Java SE 8 contains an implementation of JAX-WS 2.2 which will conflict with JAX-WS 3.0, therefore this property must be set so that JAX-WS 3.0 will be used during the building of tests and during test execution.

3. Edit the catalog file `<TS_HOME>/src/com/sun/ts/tests/jaxws/common/xml/catalog /META-INF/jax-ws-catalog.xml`, replacing the host and port settings of `systemId` with the value of your host and port setting where the WSDL is published.

4.3.2 Configuring Your Environment to Simultaneously Run the JAX-WS TCK Against the VI and the JAX-WS RI

This section describes the steps needed to configure the JAX-WS TCK so that all tests can be run; forward tests against the Vendor Implementation and reverse tests against the JAX-WS Reference Implementation.

Since the JAX-WS TCK needs to be tested against both the JAX-WS Reference Implementation and the Vendor Implementation, two separate Web servers need to be configured. Two individual Web servers are required, and the same steps, below, must be performed to configure each Web server.

If you are not going to perform this kind of testing at this time, skip this section and proceed to Configuring and Starting Your Application Server or Servers, otherwise perform the steps described in the following sections:

- Configuring Your Environment to Run the JAX-WS TCK Against the Vendor Implementation
- Configuring Your Environment to Rebuild and Run the JAX-WS TCK Against the JAX-WS RI

4.3.3 Configuring and Starting Your Application Server or Servers

Complete the following two procedures to configure your application server environments for the RI and VI.

4.3.3.1 To Configure the Vendor Implementation as your VI Environment

1. Set the following environment variables in your shell environment:
    1. `JAVA_HOME` to the directory in which Java SE is installed
    2. `TS_HOME` to the directory in which the JAX-WS TCK 3.0 software is installed
    3. `ANT_HOME` should not be set in your environment. If it is set, unset it.
2. Ensure that the `ts.jte` settings for Vendor specific properties have been configured.
3. Run the `ant config.vi` target to configure for the Vendor Implementation.

```
cd <TS_HOME>/bin
<TS_HOME>/tools/ant/bin/ant config.vi
```

4.3.3.2 To Configure the JAX-WS Reference Implementation as your RI Environment

1. Set the following environment variables in your shell environment:
    1. `JAVA_HOME` to the directory in which Java SE is installed
    2. `TS_HOME` to the directory in which the JAX-WS TCK 3.0 software is installed
    3. `ANT_HOME` should not be set in your environment. If it is set, unset it.

2. Ensure that the `ts.jte` settings for RI specific properties have been configured.
3. Run the `ant config.ri` target to configure for RI implementation.

   ```
   cd <TS_HOME>/bin
   <TS_HOME>/tools/ant/bin/ant config.ri
   ```

4.3.4 Deploying the JAX-WS TCK Tests

The JAX-WS TCK provides an automatic way of deploying both the prebuilt and Vendor-built archives to the configured web container or container by using deployment handlers.

The handler file (`<TS_HOME>/bin/xml/impl/glassfish/deploy.xml`) is written to be used with the Java EE 8 RI. If the Vendor chooses not to use Java EE 8 RI with their implementation, or chooses to rebuild the JAX-WS TCK tests using some other method than the infrastructure provided, they should create their own version handler file to provide this functionality.

This section describes the various commands used for deploying the WAR files to the configured web container.

- Generic Deployment Command Scenarios
- Deploying the JAX-WS TCK Prebuilt Archives
- Deploying the Rebuilt JAX-WS TCK Tests Against the JAX-WS Reference Implementation

4.3.4.1 Generic Deployment Command Scenarios

The `keywords` system property enables you to deploy a subset of the tests that would normally be deployed in batch mode by means of `<TS_HOME>/tools/ant/bin/ant deploy`. To specify it, add the option `-Dkeywords=`value to the `ant` command, where value is either `forward`, `reverse`, or `all`. The supported values control the directions in which the rebuildable tests are deployed.

- Setting this property to `all` (the default) deploys both the prebuilt and Vendor build tests.
- Setting the property to `forward` deploys the prebuilt tests in the forward direction only.
- Setting the property to `reverse` deploys the Vendor rebuilt tests in the reverse direction only.

4.3.4.1.1 To Deploy all the WAR Files From the `<TS_HOME>/dist` Directory to Both Web Servers

Enter the following command:

```
<TS_HOME>/tools/ant/bin/ant deploy.all
```

or

```
<TS_HOME>/tools/ant/bin/ant -Dkeywords=all deploy.all
```

4.3.4.1.2 To Deploy a Single Test Directory in the Forward Direction

Enter the following commands:

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

4.3.4.1.3 To Deploy a Subset of Test Directories in the Reverse Direction

Enter the following commands:

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

The `-Dkeywords` option is supported by the `deploy`, `undeploy`, `deploy.all`, and `undeploy.all` commands.

4.3.4.2 Deploying the JAX-WS TCK Prebuilt Archives

This section explains issues regarding the deployment of the JAX-WS TCK prebuilt archives. Before conducting any deployment, ensure that your environment has been configured by following the instructions in either the Configuring Your Environment to Run the JAX-WS TCK Against the JAX-WS Reference Implementation or the Configuring Your Environment to Run the JAX-WS TCK Against the Vendor Implementation sections.

The `<TS_HOME>/dist` directory contains all the WAR files for the JAX-WS TCK web service endpoint tests that have been compiled and generated using the JAX-WS Reference Implementation and packaged for deployment on a Servlet-compliant web container using the standard Web Archive (WAR) format.

These WAR files contain only portable artifacts for all the TCK web service endpoint tests, and are tailored to run against the JAX-WS Reference Implementation via the `web.xml` file in addition to a runtime file, `sun-jaxws.xml`. These WAR files allow you to deploy (without any additional setup or modification) against the JAX-WS Reference Implementation to test the various features and functionality of this implementation.

A Vendor is required to deploy the prebuilt WAR files as is on their JAX-WS implementation without any changes to the WAR archives with the exception of replacing and/or removing only the `web.xml` and the `sun-jaxws.xml` files.

To deploy the tests, the Vendor should perform a deployment using either the `deploy` or `deployall` batch command as described in Generic Deployment Command Scenarios, and specify the `-Dkeywords=forward` option.

4.3.4.3 Deploying the Rebuilt JAX-WS TCK Tests Against the JAX-WS Reference Implementation

This section describes how to deploy the Vendor rebuilt JAX-WS TCK tests against the Vendor Implementation. Before conducting the deployment, ensure that you have followed the instructions in Configuring Your Environment to Run the JAX-WS TCK Against the JAX-WS Reference Implementation.

This deployment scenario assumes that the Vendor has rebuilt all the JAX-WS TCK tests using the existing infrastructure, and that the WAR files exist alongside the prebuilt war files in the `<TS_HOME>/dist` directory. The rebuilt WAR files will be prepended with the text `vi_built_`.

If the Vendor chooses some other method of rebuilding the tests, they may still be able to use this deployment method as long as the WAR files are built correctly and are prepended with the text `vi_built_`. Refer to the Appendix B, "Rebuilding the JAX-WS TCK Using the Vendor's Toolset" to learn about rebuilding the JAX-WS TCK tests.

To deploy the tests, the Vendor should perform a deployment using either the `deploy` or `deployall` batch command, as described in Generic Deployment Command Scenarios, and specify the `-Dkeywords=reverse` option

4.4 Custom Configuration Handlers

Configuration handlers are used to configure and unconfigure a XML Web Services 3.0 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 Eclipse EE4J Jakarta EE 8 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 XML Web Services 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 XML Web Services 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 XML Web Services.

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 XML Web Services 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 XML Web Services TCK test environment. This section covers the following topics:

- Configuration GUI Overview
- Starting the Configuration GUI
- To Configure the JavaTest Harness to Run the XML Web Services 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 XML Web Services 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 XML Web Services 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 XML Web Services 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.

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

5.1 Starting JavaTest

There are two general ways to run the XML Web Services 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 XML Web Services 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

As explained in Appendix B, the JAX-WS TCK introduces the concept of rebuilding the TCK tests. To provide the user an ability to run the TCK against the Vendor Implementation and the Reference Implementation, the use of the `keywords` feature (in GUI mode) or option `-Dkeywords=`value (in command line mode) is provided.

By default, the TCK is configured to run all tests in both directions for all the tests except the signature tests. Setting `keywords` allows the user to change which tests will be run.

- Setting the `keywords` property to `all` (the default) does not filter out any tests, and results in the prebuilt tests to be run in the forward direction, and the Vendor rebuilt tests to be run in the reverse direction.
- Setting `keywords` to `forward` causes the prebuilt tests to be run in the forward direction only.
- Setting `keywords` to `reverse` causes the Vendor rebuilt tests to be run in the reverse direction only.

Refer to the JavaTest User's Guide and Reference in the documentation bundle for information regarding how to configure the `keywords` feature in GUI mode. For command line mode, add the following to your command line `-Dkeywords=``value`, where value is either `forward`, `reverse`, or `all`.

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

   ```
   <TS_HOME>/tools/ant/bin/ant [-Dkeywords=forward|reverse|all] runclient
   ```

The `-Dkeywords` option is supported by the `runclient` command in batch mode only, not in GUI mode.

Example 5-1 Running JAX-WS TCK Signature Tests

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

```
cd <TS_HOME>/src/com/sun/ts/tests/signaturetest/jaxws
<TS_HOME>/tools/ant/bin/ant [-Dkeywords=forward|reverse|all] runclient
```

Example 5-2 Running a Single Test Directory

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

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

Example 5-3 Running a Subset of Test Directories

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

```
cd <TS_HOME>/src/com/sun/ts/tests/jaxws/api
<TS_HOME>/tools/ant/bin/ant -Dkeywords=reverse 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 XML Web Services 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 XML Web Services 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`.

5.6 Running the JAX-WS TCK Against the JAX-WS RI

This test scenario is ensures that the configuration and deployment of all the prebuilt JAX-WS TCK tests against the JAX-WS 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 Configuring Your Environment to Run the JAX-WS TCK Against the JAX-WS Reference Implementation.
2. Specify `forward` for the `keywords` option.
3. Verify that you have completed the steps in Deploying the JAX-WS TCK Prebuilt Archives.
4. Run the tests, as described in Starting JavaTest and, if desired, Running a Subset of the Tests.

5.7 Running the JAX-WS TCK Against a Vendor's Implementation

This test scenario is one of the compatibility test phases that all Vendors must pass. This ensures that the prebuilt JAX-WS TCK tests built against the JAX-WS RI can be successfully run against the Vendor

Implementation (VI).

1. Verify that you have followed the configuration instructions in Configuring Your Environment to Run the JAX-WS TCK Against the Vendor Implementation.
2. Specify `forward` for the `keywords` option.
3. Verify that you have completed the steps in Deploying the JAX-WS TCK Prebuilt Archives
4. Run the tests, as described in Starting JavaTest and, if desired, Running a Subset of the Tests.

5.8 Running the Rebuilt JAX-WS TCK Against the JAX-WS RI

This test scenario is one of the compatibility test phases that all Vendors must pass. This ensures that the JAX-WS TCK tests that are rebuilt using the Vendor's toolset can be successfully run against the Reference Implementation.

1. Verify that you have followed the configuration instructions in Configuring Your Environment to Rebuild and Run the JAX-WS TCK Against the JAX-WS RI.
2. Refer to Appendix B, to learn about rebuilding the JAX-WS TCK tests.
3. Specify `reverse` for the `keywords` option.
4. Verify that you have completed the steps in Deploying the Rebuilt JAX-WS TCK Tests Against the JAX-WS Reference Implementation.
5. Run the tests, as described in Starting JavaTest and, if desired, Running a Subset of the Tests.

5.9 Testing Interoperability Between a Vendor Implementation and the JAX-WS Reference Implementation

1. Specify `all` for the `keywords` option.
2. Verify that you have completed the steps in Deploying the JAX-WS TCK Prebuilt Archives.
3. Verify that you have completed the steps in Deploying the Rebuilt JAX-WS TCK Tests Against the JAX-WS Reference Implementation
4. Run the tests, as described in Starting JavaTest and, if desired, Running a Subset of the Tests.

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.

- 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 Rebuilding the JAX-WS TCK Using the Vendor's Toolset

The JAX-WS 3.0 specification requires that each implementation has a way to generate WSDL from Java, and to generate Java from WSDL. To verify that implementations do this in a compatible manner, half of the tests in the JAX-WS TCK require that you first rebuild them using your generation tools.

This appendix contains the following sections:

B.1 Overview

The set of prebuilt archives and classes that ship with the JAX-WS TCK were built using the JAX-WS Reference Implementation tools (wsgen Reference and wsimport Reference), and must be deployed and run against your implementation of JAX-WS. These tests are referred to as `forward` tests.

You must also rebuild the archives and classes associated with these tests using your generation tools, and then deploy and run them against the JAX-WS Reference Implementation. These tests are known as `reverse` tests. The test names of all the tests that will be run against the Reference Implementation are identical to the `forward` test names found in the client Java source files, with the added suffix of `_reverse`. Essentially, for each `forward` test, there is an identical `reverse` test. This ensures that the same behaviors are verified on each JAX-WS implementation.

The same test client source file is used for each `forward` and `reverse` test. Likewise, they also share the same test description, which only appears under the `forward` test name in the client Java source file.

To be able to run the entire test suite in a single run, you must have your implementation and the Reference Implementation configured simultaneously. Please see Configuring Your Environment to Simultaneously Run the JAX-WS TCK Against the VI and the JAX-WS RI for more information.

B.2 Rebuilding the JAX-WS TCK Classes Using Ant

Instead of rebuilding and overwriting the TCK prebuilt classes and archives for each test directory, the JAX-WS TCK provides a way for you to plug in your generation tools so that you may leverage the existing build infrastructure that creates new classes and archives alongside those that ship with the JAX-WS TCK.

1. Create your own version of the Ant tasks `WsImport` and `WsGen`.
   Documentation for these tasks can be found later in this appendix:
   - wsgen Reference
   - wsimport Reference
2. Set the `wsimport.ant.classname` and `wsgen.ant.classname` properties in `<TS_HOME>/bin/ts.jte` to point to your implementations of the above two tasks.
3. If using Java SE 8 or above:
   1. Verify that the property `endorsed.dirs` is set to the location of the VI API jars for those technologies you wish to override. Java SE 8 contains an implementation of JAX-WS 2.2 which will conflict with JAX-WS 3.0, therefore this property must be set so that JAX-WS 3.0 will be used during the building of tests and during test execution.
   2. Set the `java.endorsed.dirs` property in the `ANT_OPTS` environment variable to point to the location in which only the JAX-WS 3.0 API jars exist; for example:

      ```
      ANT_OPTS="-Djava.endorsed.dirs=PATH_TO_YOUR_ENDORSED_DIR"
      ```
4. Change to the `jaxws` test directory and execute the following build target:

   ```
   <TS_HOME>/tools/ant/bin/ant -Dbuild.vi=true clean build
   ```

   The clean and build targets may be executed in any subdirectory under `<TS_HOME>/src/com/sun/ts/tests/jaxws` as long as the `-Dbuild.vi=true` system property is also set. Failure to set this property while invoking these targets will result in the prebuilt classes and archives being deleted and/or overwritten.
   After completing the steps above, rebuilt class files will appear under `<TS_HOME>/classes_vi_built` so as not to affect class files that were generated and compiled with the JAX-WS Reference Implementation. Rebuilt archives will be prefixed with the string, `vi_built`, and will be created in the same directory (under `<TS_HOME>/dist`) as those built using the Reference Implementation.

None of the JAX-WS test client source code or the service endpoint implementation test source code is to be altered in any way by a Vendor as part of the rebuild process.

Example B-1 Rebuilding a Single Test Directory

To further illustrate this process, the example below walks you through the rebuilding of a single test directory.

1. Change to the `<TS_HOME>/src/com/sun/ts/tests/jaxws/ee/w2j/document/literal/httptest` directory.
2. Run `<TS_HOME>/tools/ant/bin/ant llc`.
   The following is a listing of classes built using the JAX-WS RI.

```
$TS_HOME/tools/ant/bin/ant llc
/var/tmp/jaxwstck/classes/com/sun/ts/tests/jaxws/ee/w2j/document/literal/httptest
------------------------------------------------------------------------------
total 60
-rw-r--r--   1 root root   13825 Apr 12 08:32 Client.class
-rw-r--r--   1 root root    2104 Apr 12 08:32 HelloImpl.class
-rw-r--r--   1 root root    1153 Apr 12 08:32 Hello.class
-rw-r--r--   1 root root     793 Apr 12 08:32 HelloOneWay.class
-rw-r--r--   1 root root     796 Apr 12 08:32 HelloRequest.class
-rw-r--r--   1 root root     799 Apr 12 08:32 HelloResponse.class
-rw-r--r--   1 root root    1564 Apr 12 08:32 HttpTestService.class
-rw-r--r--   1 root root    2845 Apr 12 08:32 ObjectFactory.class
drwxr-xr-x   3 root root     512 Apr 12 08:32 generated_classes/
-rw-r--r--   1 root root     293 Apr 12 08:32 package-info.class
drwxr-xr-x   3 root root     512 Apr 12 08:31 generated_sources/
```

3. Run `<TS_HOME>/tools/ant/bin/ant lld`.
   This shows you the listing of archives built using the JAX-WS RI.

```
$TS_HOME/tools/ant/bin/ant lld
/var/tmp/jaxwstck/dist/com/sun/ts/tests/jaxws/ee/w2j/document/literal/httptest
------------------------------------------------------------------------------
total 286
-rw-r--r--   1 root root  113318 Apr 12 08:32 WSW2JDLHttpTest.war
```

4. Once your `<TS_HOME>/bin/ts.jte` file is configured and your implementations of the `wsgen` and `wsimport` tasks are specified, run the following command:

```
<TS_HOME>/tools/ant/bin/ant -Dbuild.vi=true build
```

   This builds the classes and archives using your implementation. Once this has been done successfully, proceed to the next step.

5. Run `<TS_HOME>/tools/ant/bin/ant -Dbuild.vi=true llc`.
   This shows you the listing of classes (under `<TS_HOME>/classes_vi_built`) built using your Implementation.

```
$TS_HOME/tools/ant/bin/ant -Dbuild.vi=true llc
/var/tmp/jaxwstck/classes_vi_built/com/sun/ts/tests/jaxws/ee/w2j/document/literal/http
test
------------------------------------------------------------------------------
total 60
-rw-r--r--   1 root root    1153 Apr 12 12:01 Hello.class
-rw-r--r--   1 root root     793 Apr 12 12:01 HelloOneWay.class
-rw-r--r--   1 root root     796 Apr 12 12:01 HelloRequest.class
-rw-r--r--   1 root root     799 Apr 12 12:01 HelloResponse.class
-rw-r--r--   1 root root    1564 Apr 12 12:01 HttpTestService.class
-rw-r--r--   1 root root    2845 Apr 12 12:01 ObjectFactory.class
drwxr-xr-x   3 root root     512 Apr 12 12:01 generated_classes/
-rw-r--r--   1 root root     293 Apr 12 12:01 package-info.class
drwxr-xr-x   3 root root     512 Apr 12 12:01 generated_sources/
-rw-r--r--   1 root root    2104 Apr 12 08:33 HelloImpl.class
-rw-r--r--   1 root root   13825 Apr 12 08:33 Client.class
```

6. Run `<TS_HOME>/tools/ant/bin/ant lld`.
   This shows the listing of all archives and JAX-WS RI deployment plan descriptors for this test directory. Those built using your implementation are prepended with `vi_built_`.

```
$TS_HOME/tools/ant/bin/ant lld
/var/tmp/jaxwstck/dist/com/sun/ts/tests/jaxws/ee/w2j/document/literal/httptest
------------------------------------------------------------------------------
total 286
-rw-r--r--   1 root root   22676 Apr 12 12:01 vi_built_WSW2JDLHttpTest.war
-rw-r--r--   1 root root  113318 Apr 12 08:32 WSW2JDLHttpTest.war
```

7. Running the `clean` target while specifying the `build.vi` system property will only clean the classes and archives that you rebuilt. To clean them, run:

```
<TS_HOME>/tools/ant/bin/ant -Dbuild.vi=true clean
```

   Notice that the `vi_built` classes and archives are deleted.

Next Steps

Once you have successfully built the archives using your implementation, you can then proceed to the configuration section to learn how to deploy these archives and how to run the `reverse` tests.

B.3 Rebuilding the JAX-WS TCK Classes Manually

When rebuilding the JAX-WS TCK classes, it is strongly recommended that you use the procedure described in the previous section, Rebuilding the JAX-WS TCK Classes Using Ant. However, if you choose not to use the existing Ant-based TCK infrastructure to rebuild the tests, you can use the following procedure to rebuild the classes manually.

1. Run your tools in each of the JAX-WS test directories under `<TS_HOME>/src/com/sun/ts/tests/jaxws`, being sure to place all newly compiled classes under `<TS_HOME>/classes_vi_built`.
   Also be sure not to overwrite any of the compiled classes under `<TS_HOME>/classes`.
2. Use the existing customization files and/or any handler files that exist in each of the test directories.
3. Package the newly generated artifacts and all the other required classes into new WAR files, prepeded with the string `vi_built_`.
   These WAR files should reside in the same directory with the prebuilt WAR files under `<TS_HOME>/dist` directory.

Next Steps

As part of the manual rebuild process, you may also need to modify some of the following files. However, this is not recommended, since doing so can result in the JAX-WS TCK not being able to be built or run the prebuilt archives shipped with the TCK. The files you may need to modify are:

- XML files in `<TS_HOME>/bin/xml`; these files are used to generate the various WARs.
- Any `build.xml` file in `<TS_HOME>/src/com/sun/ts/tests/jaxws`.
- The `<TS_HOME>/src/com/sun/ts/tests/jaxws/common/common.xml` file, which is the main build file used for the `jaxws` build process. This `common.xml` file contains all the Ant tasks specific to invoking the JAX-WS TCK `jaxws` tools.

None of the JAX-WS TCK test client source code or the service endpoint implementation test source code is to be altered in any way by a Vendor as part of the rebuild process.

Once you have successfully built the archives, you can proceed to the Chapter 4, "Setup and Configuration" to learn how to deploy these archives and how to run the reverse tests.

B.4 wsgen Reference

The `wsgen` tool generates JAX-WS portable artifacts used in JAX-WS Web services. The tool reads a Web service endpoint class and generates all the required artifacts for Web service deployment and invocation.

B.4.1 wsgen Syntax

```
wsgen [options] SEI
```

where SEI is the service endpoint interface implementation class.

Table B-1 wsgen Command Syntax

Option	Description
`-classpath` path	Specify where to find input class files.
`-cp` path	Same as `-classpath` path.
`-d` directory	Specify where to place generated output files.
`-extension`	Allow vendor extensions (functionality not specified by the specification). Use of extensions may result in applications that are not portable or may not interoperate with other implementations.
`-help`	Display help.
`-keep`	Keep generated files.
`-r` directory	Used only in conjunction with the `-wsdl` option. Specify where to place generated resource files such as WSDLs.
`-s` directory	Specify where to place generated source files.
`-verbose`	Output messages about what the compiler is doing.
`-version`	Print version information. Use of this option will ONLY print version information; normal processing will not occur.
`-wsdl[:`protocol`]``	By default `wsgen` does not generate a WSDL file. This flag is optional and will cause `wsgen` to generate a WSDL file and is usually only used so that the developer can look at the WSDL before the endpoint is deployed. The protocol is optional and is used to specify what protocol should be used in the `wsdl:binding`. Valid protocols include: `soap1.1` and `Xsoap1.2`. The default is `soap1.1`. `Xsoap1.2` is not standard and can only be used in conjunction with the `-extension` option.
`-servicename` name	Used only in conjunction with the `-wsdl` option. Used to specify a particular `wsdl:service` name to be generated in the WSDL; for example: `-servicename "{http://mynamespace/}MyService"`

Option	Description
-portname name	Used only in conjunction with the -wsdl option. Used to specify a particular wsdl:port name to be generated in the WSDL; for example: `-portname "{http://mynamespace/}MyPort"`

B.4.2 wsgen Ant Task

An Ant task for the wsgen tool is provided along with the tool. The attributes and elements supported by the Ant task are listed below.

```
<wsgen
   sei="..."
   destdir="directory for generated class files"
   classpath="classpath" | cp="classpath"
   resourcedestdir="directory for generated resource files such as WSDLs"
   sourcedestdir="directory for generated source files"
   keep="true|false"
   verbose="true|false"
   genwsdl="true|false"
   protocol="soap1.1|Xsoap1.2"
   servicename="..."
   portname="...">
   extension="true|false"
   <classpath refid="..."/>
</wsgen>
```

Table B-2 wsgen Attributes and Elements

Attribute	Description	Command Line
sei	Name of the service endpoint interface implementation class.	SEI
destdir	Specify where to place output generated classes.	-d
classpath	Specify where to find input class files.	-classpath
cp	Same as -classpath.	-cp

Attribute	Description	Command Line
resourcedestdir	Used only in conjunction with the -wsdl option. Specify where to place generated resource files such as WSDLs.	-r
sourcedestdir	Specify where to place generated source files.	-s
keep	Keep generated files.	-keep
verbose	Output messages about what the compiler is doing.	-verbose
genwsdl	Specify that a WSDL file should be generated.	-wsdl
protocol	Used in conjunction with genwsdl to specify the protocol to use in the wsdl:binding. Value values are soap1.1\`or \`Xsoap1.2, default is soap1.1. Xsoap1.2\`is not standard and can only be used in conjunction with the \`-extensions option.	-wsdl:soap1.1
servicename	Used in conjunction with the genwsdl option. Used to specify a particular wsdl:service name for the generated WSDL; for example: `servicename="{http://mynamespace/}MyService"`	-servicename
portname	Used in conjunction with the genwsdl option. Used to specify a particular wsdl:portmame name for the generated WSDL; for example: `portname="{http://mynamespace/}MyPort"`	-servicename
extension	Allow vendor extensions (functionality not specified by the specification). Use of extensions may result in applications that are not portable or may not interoperate with other implementations.	-extension

The classpath attribute is a path-like structure (http://ant.apache.org/manual/using.html#path) and can also be set by using nested <classpath> elements. Before this task can be used, a <taskdef> element needs to be added to the project as shown below.

```
<taskdef name="wsgen" classname="com.sun.tools.ws.ant.WsGen">
   <classpath path="jaxws.classpath"/>
</taskdef>
```

where jaxws.classpath is a reference to a path-like structure (`http://ant.apache.org/manual/using.html#path`), defined elsewhere in the build environment, and contains the list of classes required by the JAX-WS tools.

B.4.3 wsgen Example

```
<wsgen
   resourcedestdir=""
   sei="fromjava.server.AddNumbersImpl">
   <classpath refid="compile.classpath"/>
</wsgen>
```

B.5 wsimport Reference

The `wsimport` tool generates JAX-WS portable artifacts, such as:

- Service Endpoint Interface (SEI)
- Service
- Exception class mapped from `wsdl:fault` (if any)
- Async Reponse Bean derived from response `wsdl:message` (if any)
- JAXB generated value types (mapped Java classes from schema types)

These artifacts can be packaged in a WAR file with the WSDL and schema documents along with the endpoint implementation to be deployed.

The `wsimport` tool can be launched using the command line script `wsimport.sh` (UNIX) or `wsimport.bat` (Windows). There is also an Ant task to import and compile the WSDL. See the below for further details.

This section contains the following topics:

- wsimport Syntax
- wsimport Ant Task
- wsimport Examples

B.5.1 wsimport Syntax

```
wsimport [options] wsdl
```

where wsdl is the WSDL file.

Table B-3 wsimport Command Syntax

Option	Description
`-d` directory	Specify where to place generated output files.
`-b` path	Specify external JAX-WS or JAXB binding files (Each `<file>` must have its own `-b`).
`-B` jaxbOption	Pass this option to JAXB schema compiler
`-catalog`	Specify catalog file to resolve external entity references, it supports `TR9401`, `XCatalog`, and OASIS XML Catalog format. Please read the Catalog Support document or see the `wsimport_catalog` sample.
`-extension`	Allow vendor extensions (functionality not specified by the specification). Use of extensions may result in applications that are not portable or may not interoperate with other implementations.
`-help`	Display help.
`-httpproxy:`host`:`port	Specify an HTTP proxy server (port defaults to `8080`).
`-keep`	Keep generated files.
`-p`	Specifying a target package with this command-line option overrides any WSDL and schema binding customization for package name and the default package name algorithm defined in the specification.
`-s` directory	Specify where to place generated source files.
`-verbose`	Output messages about what the compiler is doing.
`-version`	Print version information.
`-wsdllocation` location	`@WebService.wsdlLocation` and `@WebServiceClient.wsdlLocation` value.
`-target`	Generate code for the specified version of the JAX-WS specification. For example, a value of 2.0 generates code that is compliant with the JAX-WS 2.0 Specification. The default value is 3.0.
`-quiet`	Suppress `wsimport` output.
`-XadditionalHeaders`	Map the headers not bound to request or response message to Java method parameters.

Option	Description
`-Xauthfile`	File to carry authorization information in the format `http://username:password@example.org/stock?wsdl`. Default value is `$HOME/.metro/auth`.
`-Xdebug`	Print debug information.
`-Xno-addressing-databinding`	Enable binding of W3C `EndpointReferenceType` to Java.
`-Xnocompile`	Do not compile generated Java files.
`-XdisableSSLHostnameVerification`	Disables the SSL Hostname verification while fetching the wsdls.

Multiple JAX-WS and JAXB binding files can be specified using the `-b` option, and they can be used to customize various things, such as package names and bean names. More information on JAX-WS and JAXB binding files can be found in the customization documentation.

B.5.2 wsimport Ant Task

An Ant task for the `wsimport` tool is provided along with the tool. The attributes and elements supported by the Ant task are listed below.

```
<wsimport
   wsdl="..."
   destdir="directory for generated class files"
   sourcedestdir="directory for generated source files"
   keep="true|false"
   extension="true|false"
   verbose="true|false"
   version="true|false"
   wsdlLocation="..."
   catalog="catalog file"
   package="package name"
   target="target release"
   binding="..."
   quiet="true|false"
   xadditionalHeaders="true|false"
   xauthfile="authorization file"
   xdebug="true|false"
   xNoAddressingDatabinding="true|false"
   xnocompile="true|false"
   <binding dir="..." includes="..." />
   <arg value="..."/>
   <xjcarg value="..."/>
   <xmlcatalog refid="another catalog file"/>>
</wsimport>
```

Table B-4 wsimport Attributes and Elements

Attribute	**Description**	**Command Line**
`wsdl`	WSDL file.	`WSDL`
`destdir`	Specify where to place output generated classes	`-d`
`sourcedestdir`	Specify where to place generated source files, keep is turned on with this option	`-s`
`keep`	Keep generated files, turned on with `sourcedestdir` option	`-keep`
`verbose`	Output messages about what the compiler is doing	`-verbose`
`binding`	Specify external JAX-WS or JAXB binding files	`-b`

Attribute	Description	Command Line
`extension`	Allow vendor extensions (functionality not specified by the specification). Use of extensions may result in applications that are not portable or may not interoperate with other implementations	`-extension`
`wsdllocation`	The WSDL URI passed through this option is used to set the value of `@WebService.wsdlLocation` and `@WebServiceClient.wsdlLocation` annotation elements on the generated SEI and Service interface	`-wsdllocation`
`catalog`	Specify catalog file to resolve external entity references, it supports `TR9401`, `XCatalog`, and OASIS XML Catalog format. Additionally, the Ant `xmlcatalog` type can be used to resolve entities. See the `wsimport_catalog` sample for more information.	`-catalog`
`package`	Specifies the target package	`-p`
`target`	Generate code for the specified version of the JAX-WS specification. For example, a value of 2.0 generates code that is compliant with the JAX-WS 2.0 Specification. The default value is 3.0.	`-target`
`quiet`	Suppress `wsimport` output.	`-quiet`
`xadditionalHeaders`	Map headers not bound to request or response message to Java method parameters.	`-XadditionalHeaders`
`xauthfile`	File to carry authorization information in the format `http://username:password@example.org/stock?wsdl`.	`-Xauthfile`
`xdebug`	Print debug information.	`-Xdebug`
`xNoAddressingDatabinding`	Enable binding of W3C EndpointReferenceType to Java.	`-Xno-addressing-databinding`
`xnocompile`	Do not compile generated Java files.	`-Xnocompile`

The `binding` attribute is like a path-like structure (`http://ant.apache.org/manual/using.html#path`) and can also be set by using nested `<binding>` elements, respectively. Before this task can be used, a `<taskdef>` element needs to be added to the project as shown below.

```
<taskdef name="wsimport" classname="com.sun.tools.ws.ant.WsImport">
   <classpath path="jaxws.classpath"/>
</taskdef>
```

where `jaxws.classpath` is a reference to a path-like structure (`http://ant.apache.org/manual/using.html#path`), defined elsewhere in the build environment, and contains the list of classes required by the JAX-WS tools.

B.5.2.1 Nested Elements

`wsimport` supports the following nested element parameters.

- `binding`: To specify more than one binding file at the same time, use a nested `<binding>` element, which has the same syntax as `<fileset>`. See `http://ant.apache.org/manual/Types/fileset.html` for more information.
- `arg`: Additional command line arguments passed to the `wsimport` Ant task. For details about the syntax, see the arg section (`http://ant.apache.org/manual/using.html#arg`) in the Ant manual. This nested element can be used to specify various options not natively supported in the `wsimport` Ant task. For example, currently there is no native support for the `-XdisableSSLHostnameVerification` command-line option for `wsimport`. This nested element can be used to pass `–X` command-line options directly, as done with `–XadditionalHeaders`. To use any of these features from the `wsimport` Ant task, you must specify the appropriate nested `<arg>` elements.
- `xjcarg`: The usage of `xjcarg` is similar to that of the `<arg>` nested element, except that these arguments are passed directly to the XJC tool (JAXB Schema Compiler), which compiles the schema that the WSDL references. For details about the syntax, see the arg section (`http://ant.apache.org/manual/using.html#arg`) in the Ant manual.
- `xmlcatalog`: The xmlcatalog (`http://ant.apache.org/manual/Types/xmlcatalog.html`) element is used to resolve entities when parsing schema documents.

B.5.3 wsimport Examples

```
<wsimport
   destdir=""
   debug="true"
   wsdl="AddNumbers.wsdl"
   binding="custom.xml"/>
```

The above example generates client-side artifacts for `AddNumbers.wsdl` and stores `.class` files in the destination directory using the `custom.xml` customization file. The `classpath` used is `xyz.jar` and compiles with debug information on.

```
<wsimport
   keep="true"
   sourcedestdir=""
   destdir=""
   wsdl="AddNumbers.wsdl">
   <binding dir="${basedir}/etc" includes="custom.xml"/>
</wsimport>
```

The above example generates portable artifacts for `AddNumbers.wsdl`, stores `.java` files in the destination directory, and stores `.class` files in the same directory.