Technology Compatibility Kit User's Guide for Java API for XML Web Services (JAX-WS) 2.2

For Technology Licensees

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–7829–10
September 2009

091030@22749

Contents

Preface

This guide describes how to install, configure, and run the Technology Compatibility Kit (TCK) that is used to test the Common Java API for XML Web Services 2.2 (JAX-WS 2.2) (JSR 224) technology.

The JAX-WS TCK is designed as a portable, configurable automated test suite for verifying the compatibility of a licensee's implementation of the JAX-WS 2.2 Specification (hereafter referred to as the licensee implementation). The JAX-WS TCK uses the JavaTest harness version 3.2.1 to run the test suite

---

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

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

Who Should Use This Book

This guide is for licensees of Sun Microsystems's JAX-WS 2.2 technology to assist them in running the test suite that verifies compatibility of their implementation of the JAX-WS 2.2 Specification.

Before You Read This Book

Before reading this guide, you should familiarize yourself with the Java programming language and the JAX-WS 2.2 Specification. A good resource for the Java Programming language is the Sun Microsystems, Inc. (`http://java.sun.com`) Web site.

The Java API for XML Web Services 2.2 is based on the JAX-WS Specification 2.2. Links to the specification and other product information can be found on the Web at `http://jcp.org/en/jsr/detail?id=224`. Sun Microsystems recommends that before you run the tests in the JAX-WS TCK you have read and are familiar with the JAX-WS 2.2 Specification and the JavaTest User's Guide, which describes the main JavaTest harness. The JavaTest User's Guide is located at `<install_dir>/docs/javatest/javatest.pdf` in the JAX-WS TCK distribution.

How This Book is Organized

If you are installing and using the JAX-WS TCK for the first time, it is recommended that you read Chapter 1 and Chapter 2 completely for the necessary background information, and then perform the steps outlined in the remaining chapters, while referring to the appendixes as necessary.

- Chapter 1 gives an overview of the principles that apply generally to all Technology Compatibility Kits (TCKs) and describes the JAX-WS TCK. It also includes a listing of the basic steps needed to get up and running with the JAX-WS TCK.
- Chapter 2 describes the conformance testing procedure and testing requirements.
- Chapter 3 explains how to install the JAX-WS TCK.
- Chapter 4 describes how to set up the JAX-WS TCK and how to start and configure the JavaTest harness software to be used with the JAX-WS TCK.
- Chapter 5 describes how to start the JavaTest harness and use the JAX-WS TCK.
- Chapter 6 describes several approaches for dealing with tests that fail to execute properly.
- Appendix A provides answers to frequently asked questions.
- Appendix B explains how to rebuild the TCK tests using a Vendor toolkit implementation; includes detailed references for the `wsgen` and `wsimport` commands

Typographic Conventions

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

TABLE P–1 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`**
		`Password:`

TABLE P–1 Typographic Conventions *(Continued)*

Typeface	Meaning	Example
aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

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.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell	`machine_name%`
C shell for superuser	`machine_name#`
Bourne shell and Korn shell	`$`
Bourne shell and Korn shell for superuser	`#`

Related Documentation

TABLE P–3 Related Documentation Titles

Application	Title
Using JavaTest software	*JavaTest User's Guide* and JavaTest online help (located in the `TS_HOME/docs` directory in the JAX-WS TCK distribution)
JAX-WS reference	The *Java API for XML Web Services 2.2 Specification*
Using Java technology	*Java Programming Language*, *Java Language Specification Second Edition*, *Java Virtual Machine Specification Second Edition*, *Java 2 Platform*

Accessing Sun Documentation Online

The Java Developer Connection (`http://developer.java.sun.com/developer/infodocs/`)^{SM} Web site enables you to access Java platform technical documentation.

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. You can email your comments to us at `docs@java.sun.com`.

◆ ◆ ◆ CHAPTER 1

Introduction

This chapter provides an overview of the principles that apply generally to all Technology Compatibility Kits (TCKs) and describes the Java API for XML Web Services 2.2 TCK (JAX-WS TCK 2.2) (JSR 224). It also includes a high level listing of what is needed to get up and running with the JAX-WS TCK.

1.1 Compatibility Testing

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

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

Compatibility test development for a given feature relies on a complete specification and reference implementation for that feature. Compatibility testing is not primarily concerned with robustness, performance, or ease of use.

1.1.1 Why Compatibility Testing is Important

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

- Compatibility testing is the means by which Sun Microsystems ensures that the Java platform does not become fragmented as it is ported to different operating systems and hardware environments.
- Compatibility testing benefits developers working in the Java programming language, allowing them to write applications once and then to deploy them across heterogeneous computing environments without porting.

- Compatibility testing allows application users to obtain applications from disparate sources and deploy them with confidence.
- Conformance testing benefits Java platform implementors by ensuring a level playing field for all Java platform ports.

1.1.2 TCK Compatibility Rules

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

1.1.3 TCK Overview

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

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

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

1.1.4 Java Community Process (JCP) Program and Compatibility Testing

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

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

For further information about the JCP program, go to Java Community Process (`http://jcp.org/en/jsr/detail?id=224`).

1.2 About the JAX-WS TCK 2.2

The JAX-WS TCK 2.2 is designed as a portable, configurable, automated test suite for verifying the compatibility of a licensee's implementation of Sun Microsystems's JAX-WS 2.2 Specification.

1.2.1 JAX-WS TCK Specifications and Requirements

This section lists the applicable requirements and specifications.

- **Specification Requirements** — Software requirements for a JAX-WS implementation are described in detail in the JAX-WS 2.2 Specification. Links to the JAX-WS specification and other product information can be found at `http://jcp.org/en/jsr/detail?id=224`.
- **JAX-WS Version** — The JAX-WS TCK 2.2 is based on the JAX-WS Specification, Version 2.2.
- **Reference Implementation** — The Reference Implementation (RI) for JAX-WS 2.2 can be downloaded from the Java Partner Engineering (`https://javapartner.sun.com`) Web site. This RI can be used with any web container that meets the minimum requirements for a container as defined in the JAX-WS 2.2 Specification. The JAX-WS 2.2 RI has also been integrated into the Java EE 6 RI bundle which is also available from Java Partner Engineering.

  See the *Java API for XML Web Services TCK 2.2 Release Notes* for more specific information about JDK version requirements, supported platforms, restrictions, etc.

1.2.2 JAX-WS TCK Components

The JAX-WS TCK 2.2 includes the following components:

- **JavaTest harness** version 3.2.1 and related documentation. See the `README-javatest.html` file, the *JavaTest Users Guide*, and the `ReleaseNotes-javatest.html` file for additional information.
- **JAX-WS TCK signature tests**; check that all public APIs are supported and/or defined as specified in the JAX-WS Version 2.2 implementation under test.
- **API tests** for all of the JAX-WS API in all related packages:
  - `javax.xml.ws`
  - `javax.xml.ws.handler`
  - `javax.xml.ws.handler.soap`
  - `javax.xml.ws.http`
  - `javax.xml.ws.soap`
  - `javax.xml.ws.spi`
  - `javax.xml.ws.spi.http`

  - `javax.xml.ws.wsaddressing`
- **End-to-end tests** that demonstrate marshalling/unmarshalling of all XML supported data types when going from WSDL-to-Java and all JAX-WS supported Java data types when going from Java-to-WSDL as JAX-WS Soap Messages both through synchronous (request/response) and asynchronous (request/response) and (one-way) RPC invocations. All client programming model invocation methods (Stub-based, Dynamic Proxy, and Dispatch interface) are tested..
- **WS-I Conformance Tests** that test and check for conformance to the WS-I Basic Profile 1.1, Attachment Profile 1.0, and Simple Soap Binding Profile 1.0 conformance specifications.
- **Java-to-WSDL/WSDL-to-Java Mapping Tests** that test and check for conformance to the Java-to-WSDL and WSDL-to-Java mappings.
- **Web Services Addressing Conformance Tests** that test and check for conformance to the Web Services Addressing 1.0 - Core, Web Services Addressing 1.0 - Metadata, and Web Services Addressing 1.0 - SOAP Binding conformance specifications.

The JAX-WS TCK tests have been tested with the following:

- JAX-WS 2.2 Reference Implementation
- Java 2 Platform, Standard Edition, Version 5
- Java 2 Platform, Standard Edition, Version 6

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

- Solaris 9/10 Sparc/x86/Opteron
- Windows XP Professional Edition
- Windows Vista Business Edition
- RedHat Enterprise Linux 4/5
- MacOS X 10.5

1.2.3 JavaTest Harness

The JavaTest™ harness version 3.2.1 is a set of tools designed to run and manage test suites on different Java platforms. The JavaTest harness can be described as both a Java application and a set of compatibility testing tools. It can run tests on different kinds of Java platforms and it allows the results to be browsed online within the JavaTest GUI, or offline in the HTML reports that the JavaTest harness generates.

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

- Sequencing of tests, allowing them to be loaded and executed automatically
- Graphic user interface (GUI) for ease of use
- Automated reporting capability to minimize manual errors
- Failure analysis

- Test result auditing and auditable test specification framework
- Distributed testing environment support

To run tests using the JavaTest harness, you specify which tests in the test suite to run, how to run them, and where to put the results as described in Chapter 4.

1.2.4 TCK Compatibility Test Suite

The *test suite* is the collection of tests used by the JavaTest harness to test a particular technology implementation. In this case, it is the collection of tests used by the JAX-WS TCK 2.2 to test a JAX-WS 2.2 implementation. The tests are designed to verify that a licensee's runtime implementation of the technology complies with the appropriate specification. The individual tests correspond to assertions of the specification.

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

1.2.5 Exclude Lists

Each version of a TCK includes an Exclude List contained in a `.jtx` file. This is a list of test file URLs that identify tests which do not have to be run for the specific version of the TCK being used. Whenever tests are run, the JavaTest harness automatically excludes any test on the Exclude List from being executed.

A licensee is not required to pass any test—or even run any test—on the Exclude List. The Exclude List file, `<TS_HOME>/bin/ts.jtx`, is included in the JAX-WS TCK.

Note – From time to time, updates to the Exclude List are made available on the Java Partner Engineering (`https://javapartner.sun.com`) Web site. You should always make sure you are using an up-to-date copy of the Exclude List before running the JAX-WS 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 Sun Microsystems reference implementations. Any tests that fail when run on a reference Java platform are put on the Exclude List. Any test that is not specification-based, or for which the specification is vague, may be excluded. Any test that is found to be implementation dependent (based on a particular thread scheduling model, based on a particular file system behavior, and so on) may be excluded.

Note – Licensees are not permitted to alter or modify Exclude Lists. Changes to an Exclude List can only be made by using the procedure described in Java API for XML Web Services 2.2 Test Appeals Process.

1.2.6 JAX-WS TCK Configuration

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

1.3 Getting Started With the JAX-WS TCK

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

▼ To get started with the JAX-WS TCK

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

   - Java SE 5 or higher
   - JAX-WS TCK version 2.2
   - The JAX-WS 2.2 Reference Implementation (RI)
   - The JAX-WS 2.2 Vendor Implementation (VI)
   - The Java EE 6 RI which contains the JAX-WS 2.2 Reference Implementation (RI)

   See the documentation for each of these software applications for installation instructions. See Chapter 3 for instructions on installing the JAX-WS TCK.

2. **Set up the JAX-WS TCK software.**

   See Chapter 4 for details about the following steps.

   a. **Set up your shell environment.**

   b. **Modify the required properties in the `<TS_HOME>/bin/ts.jte` file.**

   c. **Configure the JavaTest harness.**

3. **Test the JAX-WS 2.2 implementation.**

   Test the JAX-WS implementation installation by running the test suite. See Chapter 5.

◆ ◆ ◆ CHAPTER 2

Procedure for Java API for XML Web Services 2.2 Certification

This chapter describes the compatibility testing process and compatibility requirements for Java API for XML Web Services 2.2 (JSR 224). This chapter contains the following sections:

2.1 Certification Overview

The certification process for Java API for XML Web Services 2.2 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.
- Certify to Java Licensee Engineering that you have finished testing and that you meet all the compatibility requirements.

2.2 Compatibility Requirements

The compatibility requirements for Java API for XML Web Services 2.2 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
Computational Resource	A piece of hardware or software that may vary in quantity, existence, or version, which may be required to exist in a minimum quantity and/or at a specific or minimum revision level so as to satisfy the requirements of the Test Suite. Examples of computational resources that may vary in quantity are RAM and file descriptors. Examples of computational resources that may vary in existence (that is, may or may not exist) are graphics cards and device drivers. Examples of computational resources that may vary in version are operating systems and device drivers.
Configuration Descriptor	Any file whose format is well defined by a specification and which contains configuration information for a set of Java classes, archive, or other feature defined in the specification.
Conformance Tests	All tests in the Test Suite for an indicated Technology Under Test, as distributed by the Maintenance Lead, excluding those tests on the Exclude List for the Technology Under Test.
Documented	Made technically accessible and made known to users, typically by means such as marketing materials, product documentation, usage messages, or developer support programs.
Exclude List	The most current list of tests, distributed by the Maintenance Lead, that are not required to be passed to certify conformance. The Maintenance Lead may add to the Exclude List for that Test Suite as needed at any time, in which case the updated Exclude List supplants any previous Exclude Lists for that Test Suite.
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 JAX-WS Specification.
Libraries	The class libraries, as specified through the Java Community Process (JCP), for the Technology Under Test. The Libraries for Java API for XML Web Services 2.2 are listed at the end of this chapter.
Location Resource	A location of classes or native libraries that are components of the test tools or tests, such that these classes or libraries may be required to exist in a certain location in order to satisfy the requirements of the test suite. For example, classes may be required to exist in directories named in a `CLASSPATH` variable, or native libraries may be required to exist in directories named in a `PATH` variable.

TABLE 2–1 Definitions *(Continued)*

Term	Definition
Maintenance Lead	The Java Community Process member responsible for maintaining the Specification, reference implementation, and TCK for the Technology. Sun is the Maintenance Lead for Java API for XML Web Services 2.2.
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).
Product	A licensee product in which the Technology Under Test is implemented or incorporated, and that is subject to compatibility testing.
Product Configuration	A specific setting or instantiation of an Operating Mode. For example, a Product supporting an Operating Mode that permits user selection of an external encryption package may have a Product Configuration that links the Product to that encryption package.
Rebuildable Tests	Tests that must be built using an implementation specific mechanism. This mechanism must produce specification defined artifacts. Rebuilding and running these tests against the Java EE 5 RI verifies that the mechanism generates compatible artifacts.
Resource	A Computational Resource, a Location Resource, or a Security Resource.
Rules	These definitions and rules in this Compatibility Requirements section of this User's Guide.
Security Resource	A security privilege or policy necessary for the proper execution of the Test Suite. For example, the user executing the Test Suite will need the privilege to access the files and network resources necessary for use of the Product.
Specifications	The documents produced through the Java Community Process that define a particular Version of a Technology. The Specifications for the Technology Under Test can be found later in this chapter.
Technology	Specifications and a reference implementation produced through the Java Community Process.
Technology Under Test	Specifications and the reference implementation for Java API for XML Web Services 2.2.
Test Suite	The requirements, tests, and testing tools distributed by the Maintenance Lead as applicable to a given Version of the Technology.
Version	A release of the Technology, as produced through the Java Community Process.

TABLE 2–1 Definitions *(Continued)*

Term	Definition
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 JAX-WS Specification.

2.2.2 Rules for Java API for XML Web Services 2.2 Products

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

1. 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.

   a. 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.

   b. 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.
2. Some Conformance Tests may have properties that may be changed. Properties that can be changed are identified in the TCK configuration interview. Apart from changing such properties and other allowed modifications described in this User's Guide (if any), no source or binary code for a Conformance Test may be altered in any way without prior written permission. Any such allowed alterations to the Conformance Tests would be posted to the Java Licensee Engineering web site and apply to all licensees.
3. The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.
4. The Exclude List associated with the Test Suite cannot be modified.

5. The Maintenance Lead can define exceptions to these Rules. Such exceptions would be made available to and apply to all licensees.
6. 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.
7. 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.
8. 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.
9. The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.
10. Source Code in WSDL-to-Java Output when compiled by a Reference Compiler must execute properly when run on a Reference Runtime.
11. Source Code in WSDL-to-Java Output must be in source file format defined by the Java Language Specification (JLS).
12. Java-to-WSDL Output must fully meet W3C requirements for the Web Services Description Language (WSDL) 1.1.
13. 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).

2.2.3 Java API for XML Web Services 2.2 Test Appeals Process

Sun has a well established process for managing challenges to its Java technology Test Suites and plans to continue using a similar process in the future. Because Java API for XML Web Services 2.2 requires one or more subcomponent TCKs, the Java API for XML Web Services 2.2 test appeals process will be consistent with the existing subcomponent TCK test appeals processes. Sun, as Java API for XML Web Services 2.2, will authorize representatives from the Java Licensee Engineering group to be the point of contact for all test challenges. Typically this will be the engineer assigned to a company as part of its Java API for XML Web Services TCK support.

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

- Test has bugs (i.e., program logic errors)
- Specification item covered by the test is ambiguous
- Test does not match the specification
- Test assumes unreasonable hardware and/or software requirements
- Test is biased to a particular implementation

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

Test challenges must be made in writing to Java Licensee Engineering and include all relevant information as described in the Test Challenge Form. The process used to determine the validity or invalidity of a test (or related group of tests) is described in "Java API for XML Web Services TCK Test Appeals Steps."

All tests found to be invalid will either be placed on the Exclude List for that version of the Java API for XML Web Services TCK or have an alternate test made available.

- Tests that are placed on the Exclude List will be placed on the Exclude List within one business day after the determination of test validity. The new Exclude List will be made available to all Java API for XML Web Services TCK licensees on the Java API for XML Web Services TCK Web site.
- Sun, as Maintenance Lead has the option of creating alternative tests to address any challenge. Alternative tests (and criteria for their use) will be made available on the Java API for XML Web Services TCK Web site.

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

▼ Java API for XML Web Services TCK Test Appeals Steps

1. **Java API for XML Web Services TCK licensee writes a test challenge to Java Licensee Engineering contesting the validity of one or a related set of Java API for XML Web Services 2.2 tests.**

   A detailed justification for why each test should be invalidated must be included with the challenge as described by the Test Challenge Form.

2. **Java Licensee Engineering evaluates the challenge.**

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

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

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

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

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

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

2.2.3.1 Test Challenge Form

Provide the following information to Java Licensee Engineering:

```
Test Challenger Name and Company:
Specification Name(s) and Version(s):
Test Suite Name and Version:
Exclude List Version:
Test Name:
Complaint (argument for why test is invalid):
```

2.2.3.2 Test Challenge Response Form

Provide the following information in response to a test challenge.

```
Test Defender Name and Company:
Test Defender Role in Defense (e.g., test developer, Maintenance
   Lead, etc.):
Specification Name(s) and Version(s):
Test Suite Name and Version:
Test Name:
Defense (argument for why test is valid):
```

```
[Multiple challenges and corresponding responses may be listed
   here.]
Implications of test invalidity (e.g., other affected tests and test
   framework code, creation or exposure of ambiguities in spec (due
   to unspecified requirements), invalidation of the reference
   implementation, creation of serious holes in test suite):
Alternatives (e.g., are alternate test(s) appropriate?):
```

2.3 Reference Runtime for Java API for XML Web Services 2.2

Designated Reference Implementation for compatibility testing of Java API for XML Web Services 2.2 is as follows:

- Java SE for Solaris SPARC, and Win32
- Sun Reference Implementation of JAX-WS Version 2.2.
- Solaris 9/10 Sparc/x86/Opteron, RedHat Enterprise Linux 4/5, Windows XP Professional Edition, Windows Vista Business Edition, MacOS X 10.5

2.4 Specifications for Java API for XML Web Services 2.2

The Java API for XML Web Services 2.2 specification is available on the JSR 224 Web site at `http://jcp.org/en/jsr/detail?id=224` or on the Java Community Process (`http://jcp.org/en/jsr/detail?id=224`) site.

2.5 Libraries for Java API for XML Web Services 2.2

The following is a list of the packages comprising the required class libraries JAX-WS 2.2:

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

For the latest list of packages, also see:

- `http://java.sun.com/j2se/1.5/docs/api/index.html`
- `http://java.sun.com/javase/6/docs/api/index.html`

◆ ◆ ◆

CHAPTER 3

Installation

This chapter explains how to install the Java API for XML Web Services TCK 2.2 (JAX-WS TCK) software. After installing the software according to the instructions in this chapter, proceed to Chapter 4 for instructions on configuring your test environment.

3.1 Obtaining the JAX-WS 2.2 Reference Implementation

You can obtain the Sun Microsystems JAX-WS Reference Implementation (RI), Version 2.2 software from Java Partner Engineering (`https://javapartner.sun.com`) Web site.

3.2 Installing the Software

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

- Java SE 5 or higher
- JAX-WS TCK version 2.2
- The JAX-WS 2.2 Reference Implementation (RI)
- The JAX-WS 2.2 Vendor Implementation (VI)
- The Java EE 6 RI which contains the JAX-WS 2.2 Reference Implementation (RI)

▼ To install the JAX-WS software

1 **Install the Java SE software, if it is not already installed.**

Download and install the Java SE software from the Java Software (`http://java.sun.com/products`) Web site. Refer to the installation instructions that accompany the software for additional information.

2 Install the JAX-WS TCK 2.2 software

a. Copy or download the JAX-WS TCK software to your local system.

You can obtain the JAX-WS TCK software from the Java Partner Engineering (`https://javapartner.sun.com`) Web site. The JAX-WS TCK software is located in the `OPTPKG-XML/jaxws` directory in the Web site's Download Center area.

b. Use the `unzip` command to extract the bundle in the directory of your choice:

`unzip` jaxws-2_2-tck.zip

This creates the `jaxwstck` directory. The `<install_directory>/jaxwstck` directory is the test suite home, `<TS_HOME>`.

3 Install the Sun Java EE 6 RI software which contains the Sun JAX-WS 2.2 Reference Implementation or install the Standalone Sun JAX-WS 2.2 RI software.

You can obtain the Sun Java EE 6 RI software and the Standalone Sun JAX-WS 2.2 RI software from Java Partner Engineering (`https://javapartner.sun.com`) Web site.

The Sun Java EE 6 RI software contains the Sun JAX-WS 2.2 Reference Implementation and is used to validate your initial configuration and setup of the JAX-WS TCK 2.2, as well as to run the reverse tests which are explained further in Chapter 4 and Appendix B.

The Standalone Sun JAX-WS 2.2 RI software contains the Sun JAX-WS 2.2 Reference Implementation and can be used with any web container that meets the minimum requirements for a container as defined in the JAX-WS 2.2 Specification such as the Apache Tomcat web container.

If you use the Standalone Sun JAX-WS 2.2 RI software with the Apache Tomcat web container then you need to copy the jars/classes from the Standalone Sun JAX-WS 2.2 RI software to the correct location under the Apache Tomcat web container. Assuming the Apache Tomcat webcontainer is installed under: ${tomcat.home} then you would copy the jars/classes as follows:

- **cp jaxws-api.jar jsr181-api.jar jsr250-api.jar jsr173_api.jar activation.jar management-api.jar saaj-api.jar jaxb-api.jar ${tomcat.home}/common/endorsed**
- **cp jaxws-rt.jar jaxws-tools.jar resolver.jar http.jar saaj-impl.jar FastInfoset.jar jaxb-impl.jar jaxb-xjc.jar woodstox.jar stax-ex.jar gmbal.jar mimepull.jar policy.jar streambuffer.jar ${tomcat.home}/shared/lib**

4 Install the JAX-WS 2.2 Vendor Implementation (VI) to be tested.

Follow the installation instructions for the particular VI under test.

Download, install, and configure the JAX-WS 2.2 VI configuration that is under test. To familiarize yourself with the JAX-WS TCK suite and the JavaTest software before you begin testing with your own implementation, you can optionally do a trial run using the Java EE 6 RI.

◆ ◆ ◆

CHAPTER 4

Setup and Configuration

This chapter describes how to set up the JAX-WS 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.

Note – The JAX-WS TCK 2.2 contains two configuration values in `<TS_HOME>/bin/ts.jte` that correspond to the two implementations used during test execution. These two properties are `jaxws.home` and `jaxws.home.ri`.

The `jaxws.home` property should point to the implementation under test. In most cases, this will be your implementation of JAX-WS. However, it is recommended that you initially set this to the location of the Sun JAX-WS RI to familiarize yourself with the JAX-WS TCK suite and JavaTest software, as well as ensure that your environment is correctly configured.

The `jaxws.home.ri` property must always be set to the location of the Sun Reference Implementation. The installation to which `jaxws.home.ri` points to is used when running reverse tests. These tests can only be run after you have successfully rebuilt the tests using your implementation of the JAX-WS 2.2 toolset. Please see Appendix B for more information on rebuilding tests.

After completing the instructions in this chapter, proceed to Chapter 5 for instructions on running the JAX-WS TCK.

This chapter includes the following topics:

- “4.1 Configuring Your Environment to Run the JAX-WS TCK Against the Sun Reference Implementation” on page 30
- “4.2 Configuring Your Environment to Run the JAX-WS TCK Against the Vendor Implementation” on page 32
- “4.3 Configuring Your Environment to Rebuild and Run the JAX-WS TCK Against the Sun RI” on page 36

4.1 Configuring Your Environment to Run the JAX-WS TCK Against the Sun Reference Implementation

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

Note –

- 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 Sun Java EE 6 RI installation is `C:\Sun\JavaEE6`, you must specify it as `C:\\Sun\\JavaEE6` or `C:/Sun/JavaEE6`.

▼ To Configure Your Environment to Run the JAX-WS TCK Against the Sun RI

1. **The term Sun RI refers to the Sun Java EE 6 RI which contains the JAX-WS RI implementation classes/jars.**

2. **Set the following environment variables in your shell environment:**

   a. `JAVA_HOME` **to the directory in which Java SE is installed**

   b. `TS_HOME` **to the directory in which the JAX-WS TCK 2.2 software is installed**

   c. `ANT_HOME` **must be set to** `<TS_HOME>/tools/ant`

3 **Edit your `<TS_HOME>/bin/ts.jte` file and set the following environment variables:**

a. **Set the `webServerHost` property to the hostname where the web server is running that is configured with the Sun RI.**

The default setting is `localhost`.

b. **Set the `webServerPort` property to the port number where the web server is running that is configured with the Sun RI.**

The default setting is `8000`.

c. **Set the `jaxws.home` property to the location where the Sun RI is installed.**

The default setting is `${webcontainer.home}`.

d. **Set the `jaxws.classes` property to point to the Sun RI classes/jars.**

Note that this property is already set and should not require any changes.

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

f. **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 Sun RI.**

The supported web container for the Sun RI is the Sun Java EE 6 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}
```

g. **Set the `webcontainer.home` property to the location where the web server is installed for the Sun RI. This will be where the Sun Java EE 6 RI is installed.**

h. **Set the `porting.ts.url.class.1` property to the porting implementation class that is used for obtaining URLs.**

The default setting for the Sun RI porting implementation is:

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

i. **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.

- j. **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`.

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

- l. **If using Java SE 6 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 6 contains an implementation of JAX-WS 2.1 which will conflict with JAX-WS 2.2, therefore this property must be set so that JAX-WS 2.2 will be used during the building of tests and during test execution.**

4. **Edit the catalog file `<TS_HOME>/bin/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.**

5. **Provide your own implementations of the porting package interfaces provided with the JAX-WS TCK.**

   The porting interfaces are:

   - `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 `<TS_HOME>/docs/api` directory.

6. **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 Sun RI. In the Sun porting package that the TCK uses, the URLs are returned as is since this is the form that the Sun 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.**

4.2 Configuring Your Environment to Run the JAX-WS TCK Against the Vendor Implementation

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

Note –

- 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 Sun Java EE 6 RI installation is `C:\Sun\JavaEE6`, you must specify it as `C:\\Sun\\JavaEE6` or `C:/Sun/JavaEE6`.

▼ To Configure Your Environment to Run the JAX-WS TCK Against the VI

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

   a. `JAVA_HOME` **to the directory in which Java SE is installed**

   b. `TS_HOME` **to the directory in which the JAX-WS TCK 2.2 software is installed**

   c. `ANT_HOME` **must be set to** `<TS_HOME>/tools/ant`

2. **Edit your** `<TS_HOME>/bin/ts.jte` **file and set the following environment variables:**

   a. **Set the** `webServerHost` **property to the hostname where the web server is running that is configured with the Vendor Implementation.**

      The default setting is `localhost`.

   b. **Set the** `webServerPort` **property to the port number where the web server is running that is configured with the Vendor Implementation.**

      The default setting is `8000`.

   c. **Set the** `jaxws.home` **property to the location where the Vendor Implementation is installed.**

      The default setting is `${webcontainer.home}`.

d. **Set the `jaxws.classes` property to point to the Vendor Implementation classes/jars.**

   As an example, the `ts.jte` file contains the property `jaxws.classes.ri`, which contains the classes/jar files that the Sun Java EE 6 RI uses. The `jaxws.classes.ri` settings for the Sun Java EE 6 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 2.2 Sun 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}/jsr181-api.jar:
${endorsed.dirs.ri}/jsr250-api.jar:
${endorsed.dirs.ri}/jsr173_api.jar:
${endorsed.dirs.ri}/activation.jar:
${endorsed.dirs.ri}/management-api.jar:
${endorsed.dirs.ri}/saaj-api.jar:
${endorsed.dirs.ri}/jaxb-api.jar:
${jaxws.lib.ri}jaxws-rt.jar:
${jaxws.lib.ri}/jaxws-tools.jar:
${jaxws.lib.ri}/resolver.jar:
${jaxws.lib.ri}/http.jar:
${jaxws.lib.ri}/saaj-impl.jar:
${jaxws.lib.ri}/FastInfoset.jar:
${jaxws.lib.ri}/jaxb-impl.jar:
${jaxws.lib.ri}/jaxb-xjc.jar:
${jaxws.lib.ri}/woodstox.jar:
${jaxws.lib.ri}/stax-ex.jar:
${jaxws.lib.ri}/gmbal.jar:
${jaxws.lib.ri}/mimepull.jar:
${jaxws.lib.ri}/policy.jar:
${jaxws.lib.ri}/streambuffer.jar
```

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

f. **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 the `ts.jte` file contains the ri settings for the Sun Java EE 6 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 is using the Apache Tomcat web container with the Standalone JAX-WS 2.2 Sun 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}
   ```

g. **Set the** `webcontainer.home` **property to the location where the web container for the Vendor Implementation is installed.**

h. **Set the** `porting.ts.url.class.1` **property to the porting implementation class that is used for obtaining URLs.**

   The default setting for the Sun RI porting implementation is:

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

i. **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`.

j. **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`.

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

l. **If using Java SE 6 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 6 contains an implementation of JAX-WS 2.1 which will conflict with JAX-WS 2.2, therefore this property must be set so that JAX-WS 2.2 will be used during the building of tests and during test execution.**

3. **Edit the catalog file** `<TS_HOME>/bin/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 implementations of the porting package interfaces provided with the JAX-WS TCK.**

   The porting interfaces are:

   - `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 `<TS_HOME>/docs/api` directory.

5. **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 Sun RI. In the Sun porting package that the TCK uses, the URLs are returned as is since this is the form that the Sun 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 Configuring Your Environment to Rebuild and Run the JAX-WS TCK Against the Sun RI

This section describes the steps needed to configure the JAX-WS TCK so that the tests can be rebuilt (using the Vendors Implementation toolset), and then deployed and run against the Sun 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(s). After configuring your environment, continue with the instructions in Using the JavaTest Harness Software.

Note –

- 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 Sun Java EE 6 RI installation is `C:\Sun\JavaEE6`, you must specify it as `C:\\Sun\\JavaEE6` or `C:/Sun/JavaEE6`.

▼ To Configure Your Environment to Rebuild and Run Against the Sun RI

1. **The term Sun RI refers to the Sun Java EE 6 RI which contains the JAX-WS RI implementation classes/jars.**

2. **Set the following environment variables in your shell environment:**

   a. `JAVA_HOME` **to the directory in which Java SE is installed**

   b. `TS_HOME` **to the directory in which the JAX-WS TCK 2.2 software is installed**

   c. `ANT_HOME` **must be set to** `<TS_HOME>/tools/ant`

3. **Edit your** `<TS_HOME>/bin/ts.jte` **file and set the following environment variables:**

   a. **Set the** `webServerHost` **property to the hostname where the web server for the Vendor Implementation is running.**

   b. **Set the** `webServerPort` **property to the port number where the web server for the Vendor Implementation is running.**

   c. **Set the** `webServerHost.2` **property to the hostname where the web server for the Sun RI is running.**

   The default setting is `localhost`.

d. **Set the `webServerPort.2` property to the port number where the web server for the Sun RI is running.**

   The default setting is `8000`.

e. **Set the `jaxws.home` property to the location where the Vendor Implementation is installed.**

f. **Set the `jaxws.classes` property to point to the Vendors Implementation classes/jars.**

g. **Set the `jaxws.home.ri` property to the location where the Sun RI is installed.**

h. **The `jaxws.classes.ri` property is already configured to point to the Sun RI classes/jars.**

   No changes are necessary for this property.

i. **Set the `wsgen.ant.classname` property to the Vendor Implementation class that mimics the Sun RI ant task, which in turn calls the Sun `wsgen` Java-to-WSDL tool.**

j. **Set the `wsimport.ant.classname` property to the Vendor Implementation class that mimics the Sun RI ant task, which in turn calls the Sun `wsimport` WSDL-to-Java tool.**

k. **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.**

l. **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 Sun Reference implementation.**

   The supported web container for standalone web applications is the Sun Java EE 6 RI. So 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}
```

m. **Set the `webcontainer.home` property to the location where the web container for the Vendor Implemenation is installed.**

n. **Set the `webcontainer.home.ri` property to the location where the web container for the Sun Reference Implementation is installed.**

o. **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 Sun RI porting implementation which is:

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

p. **Set the `porting.ts.url.class.2` property to the Sun RI porting implementation class that is used for obtaining URLs.**

   No changes are necessary for this property.

q. **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`.

r. **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`.

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

t. **If using Java SE 6 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 6 contains an implementation of JAX-WS 2.1 which will conflict with JAX-WS 2.2, therefore this property must be set so that JAX-WS 2.2 will be used during the building of tests and during test execution.**

u. **If using Java SE 6 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 6 contains an implementation of JAX-WS 2.1 which will conflict with JAX-WS 2.2, therefore this property must be set so that JAX-WS 2.2 will be used during the building of tests and during test execution.**

4 **Edit the catalog file `<TS_HOME>/bin/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.**

5 **Provide your own implementations of the porting package interfaces provided with the JAX-WS TCK.**

The porting interfaces are:

- `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 `<TS_HOME>/docs/api` directory.

6 **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 Sun RI. In the Sun porting package that the TCK uses, the URLs are returned as is since this is the form that the Sun 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.

Refer to Appendix B for instructions on rebuilding the JAX-WS TCK.

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

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

Since the JAX-WS TCK needs to be tested against both the Sun Reference Implementation and the Vendors 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(s), otherwise perform the steps described in the following sections:

- "4.2 Configuring Your Environment to Run the JAX-WS TCK Against the Vendor Implementation" on page 32
- "4.3 Configuring Your Environment to Rebuild and Run the JAX-WS TCK Against the Sun RI" on page 36

4.5 Configuring and Starting Your Application Server(s)

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

▼ To Configure the Vendor implementation as your VI environment

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

a. `JAVA_HOME` **to the directory in which Java SE is installed**

b. `TS_HOME` **to the directory in which the JAX-WS TCK 2.2 software is installed**

  c. **`ANT_HOME` must be set to `<TS_HOME>/tools/ant`**

2 **Ensure `ts.jte` settings for Vendor specific properties have been configured.**

3 **Run `ant config.vi` target to configure for Vendor implementation.**

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

▼ To Configure the Sun Reference implementation as your RI environment

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

  a. **`JAVA_HOME` to the directory in which Java SE is installed**

  b. **`TS_HOME` to the directory in which the JAX-WS TCK 2.2 software is installed**

  c. **`ANT_HOME` must be set to `<TS_HOME>/tools/ant`**

2 **Ensure `ts.jte` settings for RI specific properties have been configured.**

3 **Run `ant config.ri` target to configure for RI implementation.**

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

4.6 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(s) via deployment handlers.

The handler file (`<TS_HOME>/bin/xml/impl/glassfish/deploy.xml` is written to be used with the Sun Java EE 6 RI. If the Vendor chooses not to use Sun Java EE 6 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.

- "4.6.1 Generic Deployment Command Scenarios" on page 42
- "4.6.2 Deploying the JAX-WS TCK Prebuilt Archives" on page 43
- "4.6.3 Deploying the Rebuilt JAX-WS TCK Tests Against the Sun Reference Implementation" on page 43

4.6.1 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 `<ANT_HOME>/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.

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

● **Enter the following command:**

```
<ANT_HOME>/bin/ant deploy.all
```

or

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

▼ To Deploy a Single Test Directory in the Forward Direction

● **Enter the following commands:**

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

▼ 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
<ANT_HOME>/bin/ant -Dkeywords=reverse deploy
```

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

4.6.2 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 Sun 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 Sun Reference Implementation and packaged for deployment on a Servlet 2.4-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 Sun 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 Sun 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 the Generic Deployment Command Scenarios, and specify the `-Dkeywords=forward` option.

4.6.3 Deploying the Rebuilt JAX-WS TCK Tests Against the Sun 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 ins Configuring Your Environment to run the JAX-WS TCK against the Sun 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 Sun 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 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 the "4.6.1 Generic Deployment Command Scenarios" on page 42 section, and specify the `-Dkeywords=reverse` option

4.7 Using the JavaTest Harness Software

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

- Through the JavaTest GUI; if using this method, please continue on to 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.

4.8 Using the JavaTest Harness Configuration GUI

You can use the JavaTest harness GUI to modify general test settings and to quickly get started with the default JAX-WS TCK test environment. This section covers the following topics:

- "4.8.1 Configuration GUI Overview" on page 44
- "4.8.2 Starting the Configuration GUI" on page 45
- "4.8.3 The Configuration Interview" on page 45
- "4.8.4 Modifying the Default Test Configuration" on page 47

Note – 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.

4.8.1 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 *Run Tests->Start* to begin a test run, the JavaTest harness determines whether all of the required configuration information has been supplied:

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

4.8.2 Starting the Configuration GUI

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

The JAX-WS 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 The Configuration Interview.

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 The Configuration Interview.

4.8.3 The Configuration Interview

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.

▼ To configure the JavaTest harness to run the JAX-WS TCK tests

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 Starting JavaTest or run a subset of the tests by completing the steps in 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_HOME>/bin/ant gui
   ```

   If the Welcome Screen does not appear do the following, otherwise skip to next step.

2. **Click** ***File->Open Quick Start Wizard*****.**

   The Welcome screen displays.

3. **Click** ***Start a new test run*****, and then click** ***Next*****.**

   You are prompted to Create a new configuration or use a configuration template.

4. **Select** ***Create a new configuration*****, and then click** ***Next*****.**

   You are prompted to select a test suite.

5. **Accept the default suite (`<TS_HOME>/src`), and then click** ***Next*****.**

   You are prompted to specify a work directory to use to store your test results.

6. **Type a work directory name or use the** ***Browse*** **button to select a work directory, and then click** ***Next*****.**

   You are prompted to start the configuration editor or start a test run. At this point, the JAX-WS TCK is configured to run the default test suite.

7. **Uncheck the** ***Start the configuration editor*** **option, and then click** ***Finish*****.**

8. **Click Run Tests->Start.**

   The JavaTest harness starts running the tests.

9. **To Reconfigure the JavaTest test harness, click** ***Configuration->New Configuration*** **or** ***Configuration->Change Configuration*****.**

10. **Click** ***Report->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.**

4.8.4 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 that is most convenient for you. Please refer to the JavaTest Harness documentation or the online help for complete information about these various options.

◆ ◆ ◆ CHAPTER 5

Executing Tests

The JAX-WS 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:

Note – The instructions in this chapter assume that you have installed and configured your test environment as described in Chapter 3 and Chapter 4, respectively.

5.1 Overview

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 Sun 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 Sun 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 Sun 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.

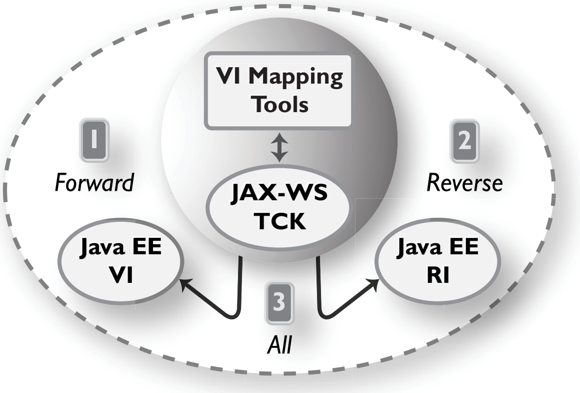

FIGURE 5–1 JAX-WS TCK Test Paths

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`.

5.2 Starting JavaTest

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

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

Note – The `<ANT_HOME>/bin/ant` command referenced in the following two procedures and elsewhere in this guide is a wrapper around the Ant build tool, which is included in the JAX-WS TCK bundle. The `build.xml` file in `<TS_HOME>/bin` contains the various Ant targets for the JAX-WS TCK test suite

---

▼ To Start JavaTest in GUI Mode

1. **Change to the `<TS_HOME>/bin` directory and execute the `<ANT_HOME>/bin/ant gui` target:**

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

2. **Configure the `keywords` feature within the JavaTest GUI.**

Note – 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.

▼ To Start JavaTest in Command-Line Mode

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

2. **Start JavaTest using the following command:**

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

Note – 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
<ANT_HOME>/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/javax_xml_ws/Dispatch
<ANT_HOME>/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
<ANT_HOME>/bin/ant -Dkeywords=reverse runclient
```

5.3 Running a Subset of the Tests

▼ To Run a Subset of Tests in GUI Mode

1. **Click *Configure->Change Configuration->Tests to Run* from the JavaTest main menu.**

   The tabbed Configuration Editor dialog 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.

4. **Click *Save File*.**

5. **Click *Run Tests->Start* to run the tests you selected.**

   Alternatively, you can right-click the test you want from the test tree in the left pane of the JavaTest main window, and choose *Execute These Tests* from the popup menu.

6. **Click *Report->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.**

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

1. **Change to the directory containing the tests you want to run.**

   For example, `<TS_HOME>/src/com/sun/ts/tests/jaxws/api`.

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

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

The tests in `<TS_HOME>/src/com/sun/ts/tests/jaxws/api` and its subdirectories are run in the direction(s) you specified with the `keywords` option.

▼ 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_HOME>/bin/ant`.

- **Invoke `<ANT_HOME>/bin/ant` with the `priorStatus` property.**

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

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

For example, you could run all the JAX-WS tests with a status of failed and error by invoking the following commands:

```
cd <TS_HOME>/src/com/sun/ts/tests/jaxws
<ANT_HOME>/bin/ant -DpriorStatus="fail,error" \
[-Dkeywords=forward|reverse|all] runclient
```

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

5.4 Running the JAX-WS TCK against the Sun RI

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

▼ To Run the JAX-WS TCK Against the Sun RI

1 **Verify that you have followed the configuration instructions in "4.1 Configuring Your Environment to Run the JAX-WS TCK Against the Sun Reference Implementation" on page 30.**

2 **Specify `forward` for the `keywords` option.**

3 **Verify that you have completed the steps in "4.6.2 Deploying the JAX-WS TCK Prebuilt Archives" on page 43.**

4 **Run the tests, as described in "5.2 Starting JavaTest" on page 50 and, if desired, "5.3 Running a Subset of the Tests" on page 52.**

5.5 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 Sun prebuilt JAX-WS TCK tests built against the Sun RI can be successfully run against the Vendors Implementation (VI).

▼ To Run the JAX-WS TCK Against a VI

1 **Verify that you have followed the configuration instructions in "4.2 Configuring Your Environment to Run the JAX-WS TCK Against the Vendor Implementation" on page 32.**

2 **Specify `forward` for the `keywords` option.**

3 **Verify that you have completed the steps in "4.6.2 Deploying the JAX-WS TCK Prebuilt Archives" on page 43**

4 **Run the tests, as described in "5.2 Starting JavaTest" on page 50 and, if desired, "5.3 Running a Subset of the Tests" on page 52.**

5.6 Running the Rebuilt JAX-WS TCK Against the Sun 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 Sun Reference Implementation.

▼ To Run the Rebuilt JAX-WS TCK Against the Sun RI

1 **Verify that you have followed the configuration instructions in "4.3 Configuring Your Environment to Rebuild and Run the JAX-WS TCK Against the Sun RI" on page 36.**

2 **Refer to Appendix B, Rebuilding the JAX-WS TCK Using the Vendor's Toolset 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 "4.6.3 Deploying the Rebuilt JAX-WS TCK Tests Against the Sun Reference Implementation" on page 43.**

5 **Run the tests, as described in "5.2 Starting JavaTest" on page 50 and, if desired, "5.3 Running a Subset of the Tests" on page 52.**

5.7 Testing Interoperability Between a Vendor Implementation and the Sun Reference Implementation

▼ To Test Interoperability Between a VI and the Sun RI

1 **Specify `all` for the `keywords` option.**

2 **Verify that you have completed the steps in "4.6.2 Deploying the JAX-WS TCK Prebuilt Archives" on page 43.**

3 **Verify that you have completed the steps in "4.6.3 Deploying the Rebuilt JAX-WS TCK Tests Against the Sun Reference Implementation" on page 43**

4 **Run the tests, as described in "5.2 Starting JavaTest" on page 50 and, if desired, "5.3 Running a Subset of the Tests" on page 52.**

5.8 Test Reports

A set of report files is created for every test run. These report files can be found in the report directory you specify. After a test run is completed, the JavaTest harness writes HTML reports for the test run. You can view these files in the JavaTest ReportBrowser when running in GUI mode, or in the web browser of your choice outside the JavaTest interface.

To see all of the HTML report files, enter the URL of the `report.html` file. This file is the root file that links to all of the other HTML reports.

The JavaTest harness also creates a `summary.txt` file in the report directory that you can open in any text editor. The `summary.txt` file contains a list of all tests that were run, their test results, and their status messages.

5.8.1 Creating Test Reports

▼ To Create a Test Report in GUI Mode

1. **Click *Report->Create Report* from the JavaTest main menu.**

   You are prompted to specify a directory to use for your test reports. The default location is `<TS_HOME>/src/com/sun/ts/tests/signaturetests/jaxws`.

2. **Specify the directory you want to use for your reports, and then click *OK*.**

   Use the *Filter* drop-down 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.**

▼ To Create a Test Report in Command-Line Mode

- **Specify where you want to create the test report.**

  a. **To specify the report directory from the command line at runtime, use:**

  ```
  <ANT_HOME>/bin/ant -Dreport.dir="<report_dir>"
  ```

  Reports are written for the last test run to the directory you specify. The default location is `<TS_HOME>/src/com/sun/ts/tests/signaturetests/jaxws`.

  b. **To specify the default report directory, set the `report.dir` property in `<TS_HOME>/bin/ts.jte`.**

  For example, `report.dir="/home/josephine/reports"`.

  c. **To disable reporting, set the `report.dir` property to `"none"`, either on the command line or in `ts.jte`.**

  For example:

  ```
  <ANT_HOME>/bin/ant -Dreport.dir="none"
  ```

5.8.2 Viewing an Existing Test Report

▼ To View an Existing Report in GUI Mode

1. **Click Report->Open Report from the JavaTest main menu.**

   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.

▼ 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.jte`**.**

◆ ◆ ◆ CHAPTER 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.

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.

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 tabbed pane.

Choose the *Error* and the *Failed* panes 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 pane. The tabbed pane contains detailed information about the test run and, at the bottom of the pane, 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 panes listed in order of importance:

- **Test Run Messages** contains a Message list and a Message pane 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.

Note – 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 5.3 “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).

◆ ◆ ◆ APPENDIX A

Frequently Asked Questions

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

From the JavaTest GUI, you can view recently run tests using the Test Results Summary, by selecting the red Failed tab or the blue Error tab. See Chapter 6 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 (`*.jtx`) contains all tests that are not required by Sun Microsystems to be run. The following is a list of reasons for a test to be included in the Exclude List:

- An error in a Sun Microsystems 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.

◆ ◆ ◆

APPENDIX B

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

The JAX-WS 2.2 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 Sun Reference Implementation tools (`wsgen` and `wsimport`), 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 Sun Reference Implementation. These tests are known as `reverse` tests. The test names of all the tests that will be run against the Sun 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.

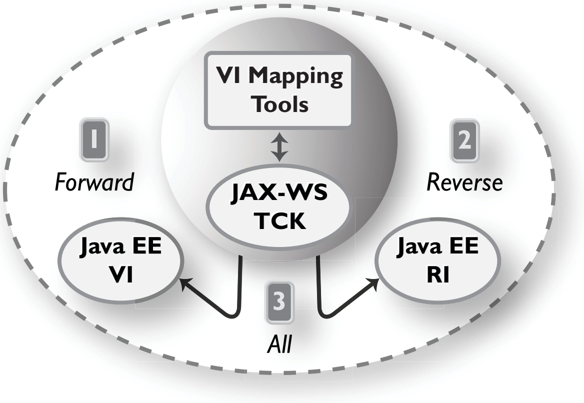

FIGURE B–1 JAX-WS TCK Test Paths

Note – 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 Sun Reference Implementation configured simultaneously. Please see "4.4 Configuring Your Environment to Simultaneously Run the JAX-WS TCK Against the VI and the Sun RI" on page 40 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.

▼ To Rebuild the JAX-WS TCK Classes Using Ant

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 6 or above:**

a. **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 6 contains an implementation of JAX-WS 2.0 which will conflict with JAX-WS 2.2, therefore this property must be set so that JAX-WS 2.2 will be used during the building of tests and during test execution.**

b. **Set the `java.endorsed.dirs` property in the `ANT_OPTS` environment variable to point to the location in which only the JAX-WS 2.2 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**

```
<ANT_HOME>/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 Sun 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 Sun Reference Implementation.

Note – 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 `<ANT_HOME>/bin/ant llc`.

   The following is a listing of classes built using the Sun RI.

```
$ANT_HOME/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 `<ANT_HOME>/bin/ant lld`.

   This shows you the listing of archives built using the Sun RI.

   ```
   $ANT_HOME/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:

   ```
   <ANT_HOME>/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 `<ANT_HOME>/bin/ant -Dbuild.vi=true llc`.

   This shows you the listing of classes (under `<TS_HOME>/classes_vi_built`) built using your Implementation.

```
$ANT_HOME/bin/ant -Dbuild.vi=true llc
/var/tmp/jaxwstck/classes_vi_built/com/sun/ts/tests/jaxws/ee/w2j/document/literal/httptest
-----------------------------------------------------------------------------------------------
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 `<ANT_HOME>/bin/ant lld`.

   This shows the listing of all archives and Sun RI deployment plan descriptors for this test directory. Those built using your implementation are prepended with `vi_built_`.

   ```
   $ANT_HOME/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:

   ```
   <ANT_HOME>/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.

▼ To Rebuild the JAX-WS TCK 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.

Note – 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 configuration section 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>
```

TABLE B–1 `wsgen` Command Syntax

Option	Description
`-classpath <path>`	Specify where to find input class files.
`-cp <path>`	Same as `-classpath <path>`.

TABLE B–1 `wsgen` Command Syntax *(Continued)*

Option	Description
`-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"`
`-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 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`
`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`

TABLE B–2 `wsgen` Attributes and Elements *(Continued)*

Attribute	Description	Command Line
`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 via 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.

B.5.1 `wsimport` Syntax

```
wsimport [options] <wsdl>
```

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

TABLE B–3 `wsimport` Command Syntax *(Continued)*

Option	Description
`-catalog`	Specify catalog file to resolve external entity references, it supports `TR9401`, `XCatalog`, and OASIS XML Catalog format. Please read the XML Entity and URI Resolvers (`https://jax-ws.dev.java.net/nonav/2.2/docs/catalog-support.html`) 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 per the given JAX-WS specification version. Version 2.0 will generate compliant code per the JAX-WS 2.0 specification.
`-quiet`	Suppress `wsimport` output.
`-XadditionalHeaders`	Map the headers not bound to request or response message to Java method parameters.
`-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.

TABLE B–3 `wsimport` Command Syntax *(Continued)*

Option	Description
`-XdisableSSLHostnameVerification`	Disbales 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 like package names, bean names, etc. More information on JAX-WS and JAXB binding files can be found in the customization documentation (`https://jax-ws.dev.java.net/nonav/2.2/docs/customizations.html`).

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`
`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`

TABLE B–4 `wsimport` Attributes and Elements *(Continued)*

Attribute	Description	Command Line
`target`	Generate code per the given JAX-WS specification version. Version 2.0 will generate compliant code per the JAX-WS 2.0 specification.	`-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 via 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.5.3 `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.