Java™ Platform, Enterprise Edition 6 Compatibility Test Suite User's Guide

For Technology Licensees

Part No: 821–0947–11
July 1, 2011

110701@25097

Contents

Preface

This book introduces the Compatibility Test Suite (CTS) for the Java Platform, Enterprise Edition 6 (Java EE 6) and Java Platform, Enterprise Edition 6 Web Profile (Java EE 6 Web Profile), and explains how to configure and run the test suite. It also provides information for troubleshooting problems you may encounter as you run the test suite.

The Java Platform, Enterprise Edition 6 Compatibility Test Suite (Java EE 6 CTS) is a portable, configurable automated test suite for verifying the compatibility of a licensee's implementation of the Java EE 6 Specification (hereafter referred to as the licensee implementation). The Java EE 6 CTS uses the JavaTest harness version 3.2.1 to run the test suite.

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

Visit the Java Licensee Engineering (`https://java-partner.sun.com`) Web site for answers to frequently asked questions and send questions you may have to yourJava Licensee Engineering contact.

Who Should Use This Book

This guide is for licensees of Sun Microsystems's Java EE 6 6 technology to assist them in running the test suite that verifies compatibility of their implementation of the Java EE 6 Specification.

Before You Read This Book

Before reading this guide, you should familiarize yourself with the Java programming language, the *Java Platform, Enterprise Edition 6 (Java EE 6) Specification*, and the JavaTest documentation.

Java Programming Language, *Java Language Specification Fourth Edition*, and *Java Virtual Machine Specification Second Edition* are good sources of information about the Java Programming language, as is the Sun Microsystems, Inc. (`http://java.sun.com`) Web site.

The *Java Platform, Enterprise Edition 6 (Java EE 6) Specification* can be downloaded from http://jcp.org/en/jsr/detail?id=316 (`http://jcp.org/en/jsr/detail?id=316`).

The JavaTest documentation is included in the Java EE 6 CTS documentation bundle.

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:`

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	`#`

CHAPTER 1

Introduction

This document provides instructions for installing, configuring, and running the Java Platform, Enterprise Edition 6 Compatibility Test Suite (Java EE 6 CTS). Sun Microsystems, Inc. is committed to making Write Once, Run Anywhere a reality for the Java Platform, Enterprise Edition platform by providing the following for each Java technology:

- A specification
- A Software Development Kit (SDK)
- A Reference Implementation (RI)
- A Compatibility Test Suite (CTS)
- A brand

This chapter includes the following topics:

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 Compatibility Rules

Compatibility criteria for all technology implementations are embodied in the Compatibility Rules that apply to a specified technology. The Java EE 6 CTS tests for adherence to these Rules as described in Chapter 2, "Procedure for Java Platform, Enterprise Edition 6 Certification," for Java EE 6 and Chapter 3, "Procedure for Java Platform, Enterprise Edition 6 Web Profile Certification," for Java EE 6 Web Profile.

1.1.3 CTS Overview

A Java EE 6 CTS 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 CTS are based on the written specifications for the Java platform. The CTS 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 the Java EE 6 CTS is called the *test suite*. All tests in the CTS 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 Java EE 6 CTS. The definition of required tests will change over time. See "1.2.3 Exclude Lists" on page 21 for more information.

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/home/index`).

1.2 About the Java EE 6 CTS

The Java EE 6 CTS is designed as a portable, configurable, automated test suite for verifying the compliance of a licensee's implementation of theJava EE 6 technologies. The Java EE 6 CTS includes version 3.2.1 of the JavaTest harness.

The Java EE 6 CTS test suite includes compatibility tests for the following Java EE 6 technologies:

- Enterprise JavaBeans (EJB) 3.1
- Java Servlet 3.0
- JavaServer Pages (JSP) 2.2
- Expression Language (EL) 2.2
- Java Message Service (JMS) 1.1
- Java Transaction API (JTA) 1.1
- JavaMail 1.4
- Java EE Connector Architecture 1.6
- Web Services for Java EE 1.3
- Java API for XML-Based RPC (JAX-RPC) 1.1
- Java API for XML Web Services (JAX-WS) 2.2
- Java API for RESTful Web Services (JAX-RS) 1.1
- Java Architecture for XML Binding (JAXB) 2.2
- SOAP with Attachments API for Java (SAAJ) 1.3

- Java API for XML Registries (JAXR) 1.0
- Java EE Management 1.1
- Java EE Deployment 1.2
- Java Authorization Contract for Containers (JACC) 1.3
- Java Authentication Service Provider Interface for Containers (JASPIC) 1.0
- Debugging Support for Other Languages 1.0
- Standard Tag Library for JavaServer Pages (JSTL) 1.2
- Web Services Metadata for the Java Platform (JWS) 2.1
- JavaServer Faces 2.0
- Common Annotations for the Java Platform 1.1
- Java Persistence 2.0
- Bean Validation 1.0
- Managed Beans 1.0
- Interceptors 1.1
- JSR 330 1.0

The Java EE 6 CTS test suite can also be used to test compatibility for the following Java EE 6 Web Profile technologies:

- Java Servlet 3.0
- JavaServer Pages (JSP) 2.2
- Expression Language (EL) 2.2
- Standard Tag Library for JavaServer Pages (JSTL) 1.2
- JavaServer Faces 2.0
- Common Annotations for the Java Platform (JSR 250) 1.1
- Enterprise JavaBeans (EJB) 3.1 Lite
- Java Transaction API (JTA) 1.1
- Java Persistence (JPA) 2.0
- JSR 299 1.0

1.2.1 CTS Tests

The Java EE 6 CTS contains API tests and enterprise edition tests, which are tests that start in the Java EE 6 platform and use the underlying enterprise service(s) as specified. For example, a JDBC enterprise edition test connects to a database, uses SQL commands and the JDBC 4.0 API to populate the database tables with data, queries the database, and compares the returned results against the expected results.

FIGURE 1–1 Typical Java Platform, Enterprise Edition Workflow

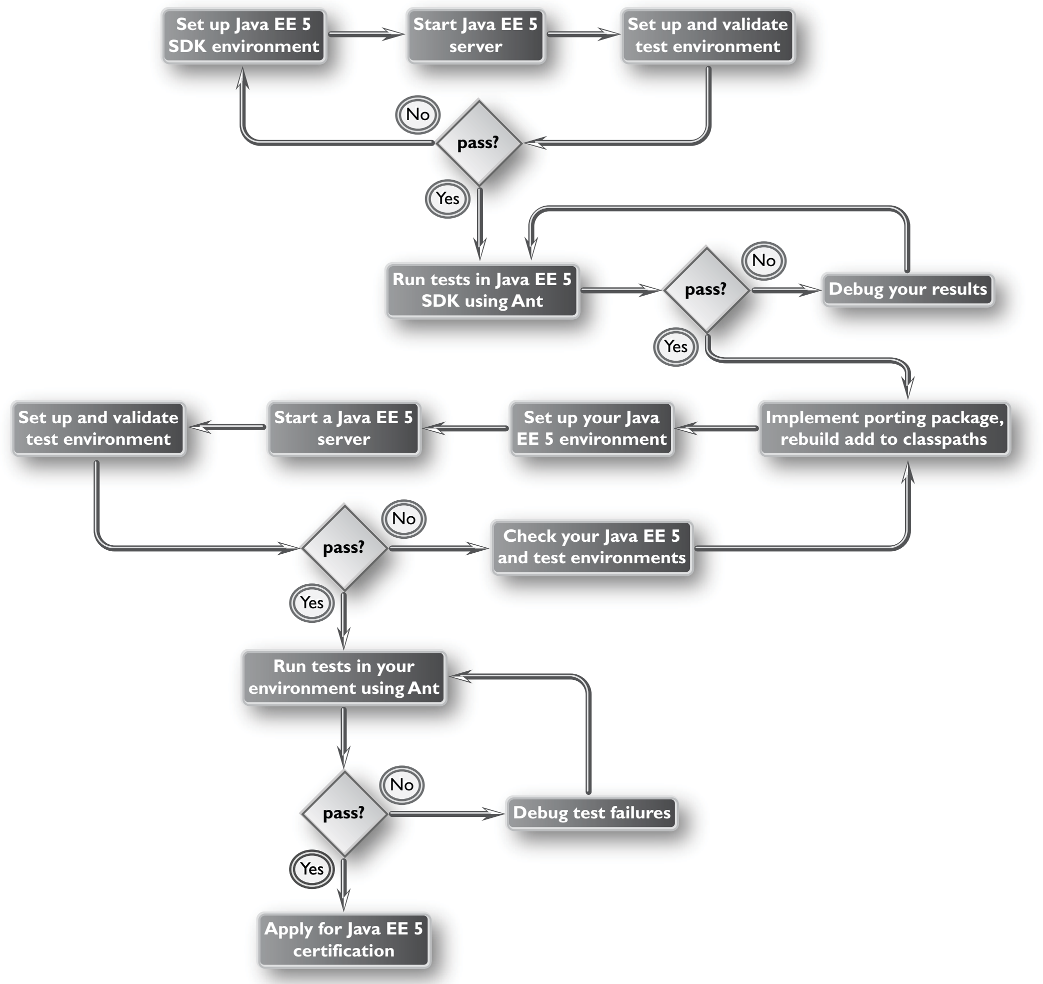

Set up Java EE 5 SDK environment
Start Java EE 5 server
Set up and validate test environment
No
pass?
Yes
Run tests in Java EE 5 SDK using Ant
pass?
No
Debug your results
Yes
Implement porting package, rebuild add to classpaths
Set up your Java EE 5 environment
Start a Java EE 5 server
Set up and validate test environment
pass?
No
Check your Java EE 5 and test environments
Yes
Run tests in your environment using Ant
pass?
No
Debug test failures
Yes
Apply for Java EE 5 certification

Figure 1–1 shows how most licensees will use the test suite. They will set up and run the test suite with the Java Platform, Enterprise Edition 6 Reference Implementation (Java EE 6 RI) first to become familiar with the testing process. Then they will set up and run the test suite with their own Java EE 6 implementation. When they pass all of the tests, they will apply for and be granted certification.

- Before you do anything with the test suite, we recommend that you read the rules in Chapter 2, “Procedure for Java Platform, Enterprise Edition 6 Certification,” or Chapter 3, “Procedure for Java Platform, Enterprise Edition 6 Web Profile Certification.” These chapters explain the certification process and provides a definitive list of certification rules for Java EE 6 and Java EE 6 Web Profile implementations.

- Next, take a look at the test assertions in the Assertion List, which you can find in the Java EE 6 CTS documentation bundle. The assertions explain what each test is testing. When you run the tests with the JavaTest GUI, the assertion being tested as part of the test description of the currently selected test is displayed.
- Third, install and configure the Java EE 6 CTS software and the Java EE 6 RI or Java EE 6 Web Profile RI and run the tests as described in this guide. This will familiarize you with the testing process.
- Finally, set up and run the test suite with your own Java EE 6 or Java EE 6 Web Profile implementation.

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

1.2.2 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 7, "Executing Tests."

The tests that make up the CTS are precompiled and indexed within the CTS 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.

1.2.3 Exclude Lists

The Java EE 6 CTS 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 CTS 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 Java EE 6 CTS.

Note – From time to time, updates to the Exclude List are made available on the Java Licensee Engineering (`https://java-partner.sun.com`) web site. You should always make sure you are using an up-to-date copy of the Exclude List before running the Java EE 6 CTS test suite to verify your implementation.

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

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

In addition, all tests are run against the 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 "2.2.3 Java EE 6 Test Appeals Process" on page 32 and "3.2.3 Java EE 6 Web Profile Test Appeals Process" on page 41.

1.2.4 Apache Ant

TheJava EE 6 RI, Java EE 6 Web Profile RI, and Java EE 6 CTS include implementations of Apache Ant 1.7.1 from the Apache Ant Project (`http://ant.apache.org/`). Apache Ant is a free, open-source, Java-based build tool, similar in some ways to the make tool, but more flexible, cross-platform compatible, and centered around XML-based configuration files.

Ant is invoked in the Java EE 6 RI, Java EE 6 Web ProfileRI, and Java EE 6 CTS in conjunction with various XML files containing Ant targets. These Ant targets provide a convenient way to automate various configuration tasks for Java EE 6 CTS. For example, the initial configuration of the Java EE 6 RI or Java EE 6 Web Profile RI for CTS is done by means of the `config.vi` Ant target.

The Ant configuration targets are there for your convenience. When configuring your Java EE 6 or Java EE 6 Web Profile implementation for the Java EE 6 CTS, you can choose to set up your environment to leverage the Ant tools, or you can perform some or all of your configuration procedures manually. The Java EE 6 CTS includes the Ant Contrib package, and the tasks included with Ant Contrib are used within the CTS build files. See `http://ant-contrib.sourceforge.net` for more information about Ant Contrib.

This User's Guide does not provide in-depth instruction on Ant internals or how to configure Ant targets for your particular Java EE 6 or Java EE 6 Web Profile implementation. For complete information about Ant, refer to the extensive documentation on the Apache Ant Project site. The Apache Ant Manual is available at `http://ant.apache.org/manual/index.html`.

Apache Ant is protected under the Apache Software, License 1.1. See the `LICENSE` in the `<TS_HOME>/tools/ant` directory, or on the Apache Ant Project license page at `http://ant.apache.org/license.html`.

1.3 Hardware Requirements

You can run the Java EE 6 CTS software on compatible Java platforms on both Sun workstations and on personal computers. The following section lists the hardware requirements for both, using the Java EE 6 RI or Java EE 6 Web Profile RI. Hardware requirements for other reference implementations will vary.

All systems must meet the following recommended and minimum hardware requirements:

- CPU running at 500 MHz minimum
- 512MB of RAM minimum
- 1024MB of swap space minimum
- 2048 MB of free disk space minimum for writing data to log files, the Java EE 6 repository, and the database
- Network access

1.4 Software Requirements

You can run the Java EE 6 CTS software on platforms running the Sun Solaris, Linux, Windows, and Mac OS software that meet the following minimum software requirements:

- Operating Systems:
  - Sun Solaris SPARC 9, 10 and Sun Solaris x86 9, 10
  - Microsoft Windows XP Professional Edition
  - Microsoft Windows Vista Professional Edition
  - Red Hat AS 4.0, 5.0
  - Mac OS 10.5
  - SuSE Linux 10.2
  - Ubuntu Linux 7.0, 8.0
- Java SE 6 SDK
- Java EE 6 RI or Java EE 6 Web Profile RI
- Sendmail running on `localhost` or network-accessible machine

Note – CTS software has been tested using Oracle, Sybase, DB2, Microsoft SQL Server, PostgreSQL, MySQL, and JavaDB.

1.5 TCK Requirements

In addition to the instructions and requirements described in this document, the following requirements must also be met:

- All Java EE 6 implementations must also pass the Java Architecture for XML Binding (JAXB), Contexts Dependency Injection for the Java EE Platform (JSR 299), Dependency Injection for Java (JSR 330), and Bean Validation TCKs.
- All Java EE 6 Web Profile implementations must also pass the Contexts Dependency Injection for the Java EE Platform (JSR 299) and Bean Validation TCKs.

  Java EE 6 Web Profile implementations do *not* need to pass the JAXB or the Dependency Injection for Java (JSR 330) TCKs.

These TCKs can be downloaded from the Java Licensee Engineering (`https://java-partner.sun.com`) web site.

For additional information about the JAXB TCK, see `https://jaxb.dev.java.net/tck.html`. For for more information about the JAXB 2.2 technology, see the specification at `http://jcp.org/en/jsr/detail?id=222`.

For more information about the Contexts Dependency Injection for the Java EE Platform (CDIJ) technology, see the specification at `http://jcp.org/en/jsr/detail?id=299`.

For more information about the Bean Validation technology, see the specification at `http://jcp.org/en/jsr/detail?id=303`.

1.6 Getting Started With the Java EE 6 CTS Test Suite

Briefly, installing, configuring, and using the Java EE 6 CTS involves the following general steps:

1. Download, install, and configure the Java EE 6 RI or Java EE 6 Web Profile RI.
2. Download and install the Java EE 6 CTS package.
3. Configure your database to work with your RI.
4. Configure CTS to work with your database and RI.
5. Run the CTS tests.

The remainder of this guide explains these steps in detail. If you just want to get started quickly with the Java EE 6 CTS using the most basic test configuration, refer to Chapter 4, "Installation."

CHAPTER 2

Procedure for Java Platform, Enterprise Edition 6 Certification

This chapter describes the compatibility testing process and compatibility requirements for Java Platform, Enterprise Edition 6 (Java EE 6). This chapter contains the following sections:

2.1 Certification Overview for Java EE 6

The certification process for Java Platform, Enterprise Edition 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 "2.2 Compatibility Requirements for Java EE 6" on page 25.
- Certify to Java Licensee Engineering that you have finished testing and that you meet all the compatibility requirements.

2.2 Compatibility Requirements for Java EE 6

The compatibility requirements for Java Platform, Enterprise Edition consist of meeting the requirements set forth by the rules and associated definitions contained in this section.

Note – As stated in Rule 18 in “2.2.2 Rules for Java EE 6 Products” on page 29, part of the Java EE 6 CTS compatibility requirements is that your Java EE 6 implementation must also pass the Java Architecture for XML Binding (JAXB), Contexts Dependency Injection for the Java EE Platform (JSR 299), Dependency Injection for Java (JSR 330), and Bean Validation TCKs. See “1.5 TCK Requirements” on page 23 for more information about this requirement.

2.2.1 Definitions for Java EE 6 Compatibility Requirements

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

Phrases of the form "Java EE 6 <*term*>" are a reference to the <*term*> as defined in Chapter 2, “Procedure for Certification,” in the associated *Java Test Compatibility Kit User's Guide*, available on the Java Licensee Engineering (`https://java-partner.sun.com`) Web site.

TABLE 2–1 Definitions

Term	Definition
Application	A collection of components contained in a single application package (such as an EAR file or JAR file) and deployed at the same time using a Deployment Tool.
Computational Resource	A piece of hardware or software that may vary in quantity, existence, or version, which may be required to exist in a minimum quantity and/or at a specific or minimum revision level so as to satisfy the requirements of the Test Suite. Examples of computational resources that may vary in quantity are RAM and file descriptors. Examples of computational resources that may vary in existence (that is, may or may not exist) are graphics cards and device drivers. Examples of computational resources that may vary in version are operating systems and device drivers.
Compatibility 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.
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.
Container	An implementation of the associated Libraries, as specified in the Specifications, and a version of a Java SE Runtime Product, as specified in the Specifications, or a later version of a Java SE Runtime Product that also meets these compatibility requirements.

TABLE 2–1 Definitions *(Continued)*

Term	Definition
Deployment Tool	A tool used to deploy applications or components in a Product, as described in the Specifications.
Documented	Made technically accessible and made known to users, typically by means such as marketing materials, product documentation, usage messages, or developer support programs.
Edition	A Version of the Java Platform, Standard Edition Technology, or a Version of the Java Platform, Enterprise Edition Technology.
Endorsed Standard	A Java API defined through a standards process other than the Java Community Process. The Endorsed Standard packages are listed later in this chapter.
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 *Java Platform, Enterprise Edition 6 Specification*.
JSP Page	A text-based document that uses JavaServer Pages technology.
JSP Page Implementation Class	A program constructed by transforming the JSP Page text into a Java language program using the transformation rules described in the Specifications.
Libraries	The class libraries, as specified through the Java Community Process (JCP), for the Technology Under Test. The Libraries for Java EE 6 are listed at the end of this chapter.
Location Resource	A location of classes or native libraries that are components of the test tools or tests, such that these classes or libraries may be required to exist in a certain location in order to satisfy the requirements of the test suite. For example, classes may be required to exist in directories named in a `CLASSPATH` variable, or native libraries may be required to exist in directories named in a `PATH` variable.
Maintenance Lead	The Java Community Process member responsible for maintaining the Specification, reference implementation, and TCK for the Technology. Sun is the Maintenance Lead for Java EE 6.

TABLE 2–1 Definitions *(Continued)*

Term	Definition
Operating Mode	Any Documented option of a Product that can be changed by a user in order to modify the behavior of the Product. For example, an Operating Mode can be binary (enable/disable optimization), an enumeration (select from a list of protocols), or a range (set the maximum number of active threads).
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 6 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.
Runtime	The Containers specified in the Specifications.
Security Resource	A security privilege or policy necessary for the proper execution of the Test Suite. For example, the user executing the Test Suite will need the privilege to access the files and network resources necessary for use of the Product.
Specifications	The documents produced through the 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 EE 6.
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.
WSDL-to-Java Output	Output of a WSDL-to-Java tool that is required for Web service deployment and invocation.

TABLE 2–1 Definitions *(Continued)*

Term	Definition
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 *Java Platform, Enterprise Edition 6 Specification*.

2.2.2 Rules for Java EE 6 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 Compatibility 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 Compatibility 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 Compatibility Tests to fail may be tested using a Product Configuration that allows all Compatibility 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.

   c. A Product may contain an Operating Mode that provides compatibility with previous versions of the Product that would not otherwise meet these compatibility requirements. The previous versions of the Product must have been shipped in a final form to customers before Jan. 1, 2001. At least the default Product Configuration of this Operating Mode must meet these compatibility requirements without invoking this rule; testing may always use such a Product Configuration. This Operating Mode must affect no smaller unit of execution than an entire Application. Any Product Configuration that invokes this rule must be clearly Documented as not meeting the requirements of the Specifications.

   d. A Product may contain an Operating Mode that selects the Edition with which it is compatible. The Product must meet the compatibility requirements for the corresponding Edition for all Product Configurations of this Operating Mode. This Operating Mode must affect no smaller unit of execution than an entire Application.

2. Some Compatibility 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 Compatibility Test may be altered in any way without prior written permission. Any such allowed alterations to the Compatibility 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 Compatibility 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.
   a. If a Product includes Technologies in addition to the Technology Under Test, then it must contain the full set of combined public and protected classes and interfaces. The API of the Product must contain the union of the included Technologies. No further modifications to the APIs of the included Technologies are allowed.
   b. A Product may provide a newer version of an Endorsed Standard. Upon request, the Maintenance Lead will make available alternate Compatibility Tests as necessary to conform with such newer version of an Endorsed Standard. Such alternate tests will be made available to and apply to all licensees. If a Product provides a newer version of an Endorsed Standard, the version of the Endorsed Standard supported by the Product must be Documented.
   c. The Maintenance Lead may authorize the use of newer Versions of a Technology included in the Technology Under Test. A Product that provides a newer Version of a Technology must meet the Compatibility Requirements for that newer Version, and must Document that it supports the newer Version.

      For example, the Java EE Maintenance Lead could authorize use of a newer version of a Java technology such as JavaServer Faces, that is also available independently of Java EE and is evolving quickly to meet market demands.

8. Except for tests specifically required by this TCK to be rebuilt (if any), the binary Compatibility 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.
   a. A Product may contain Operating Modes that meet all of these requirements, except Rule 8, provided that:
      i. At least the default Product Configuration of each Operating Mode must meet these requirements, without invoking this rule; testing may always use such a Product Configuration.
      ii. The Operating Modes must not violate the Java SE Rules.
      iii. The Product Configurations of Operating Modes of an application and its components are configured at deployment time, or by administrative action, and can not be changed during the runtime of that application.
      iv. Some Product Configurations of such Operating Modes may provide only a subset of the functional programmatic behavior required by the Specifications. The behavior of applications that use more than the provided subset, when run in such Product Configurations, is unspecified.
      v. The functional programmatic behavior of any binary class or interface in the above defined subset must be that defined by the Specifications.
      vi. Any Product Configuration that invokes this rule must be clearly Documented as not fully meeting the requirements of the Specifications
10. Each Container must make technically accessible all Java SE Runtime interfaces and functionality, as defined by the Specifications, to programs running in the Container, except only as specifically exempted by these Rules.
    a. Containers may impose security constraints, as defined by the Specifications.
11. A web Container must report an error, as defined by the Specifications, when processing a JSP Page that does not conform to the Specifications.
12. The presence of a Java language comment or Java language directive in a JSP Page that specifies "java" as the scripting language, when processed by a web Container, must not cause the functional programmatic behavior of that JSP Page to vary from the functional programmatic behavior of that JSP Page in the absence of that Java language comment or Java language directive.
13. The contents of any fixed template data (defined by the Specifications) in a JSP Page, when processed by a web Container, must not affect the functional programmatic behavior of that JSP Page, except as defined by the Specifications.
14. The functional programmatic behavior of a JSP Page that specifies "java" as the scripting language must be equivalent to the functional programmatic behavior of the JSP Page Implementation Class constructed from that JSP Page.

15. A Deployment Tool must report an error when processing a configuration descriptor that does not conform to the Specifications.
16. The presence of an XML comment in a configuration descriptor, when processed by a Deployment Tool, must not cause the functional programmatic behavior of the Deployment Tool to vary from the functional programmatic behavior of the Deployment Tool in the absence of that comment.
17. A Deployment Tool must report an error when processing an EJB Configuration Descriptor that includes an EJB QL expression that does not conform to the Specifications.
18. Compatibility testing for Java EE 6 consists of running the following Technology Compatibility Kits (TCKs): Java EE 6 CTS, JAXB 2.2 TCK, Contexts Dependency Injection for the Java EE Platform 1.0 (JSR 299) TCK, Dependency Injection for Java 1.0 (JSR 330) TCK, and Bean Validation 1.0 TCK. In addition to the compatibility rules outlined in this CTS User's Guide, Java EE 6 implementations must also adhere to any compatibility rules defined in the documentation for the aforementioned TCKs.
19. Source Code in WSDL-to-Java Output when compiled by a Reference Compiler must execute properly when run on a Reference Runtime.
20. Source Code in WSDL-to-Java Output must be in source file format defined by the Java Language Specification (JLS).
21. Java-to-WSDL Output must fully meet W3C requirements for the Web Services Description Language (WSDL) 1.1.
22. 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 EE 6 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. Sun, as Java Platform, Enterprise Edition 6 Maintenance Lead, will authorize representatives from the Java Licensee Engineering organization 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 EE 6 CTS 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 EE 6 CTS. 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 the Java Licensee Engineering organization and include all relevant information as described in step 1 in "Java EE 6 CTS Test Appeals Steps" on page 33. The process used to determine the validity or invalidity of a test (or related group of tests) is also described in that section.

All tests found to be invalid will either be placed on the Exclude List for that version of the Java EE 6 CTS or have an alternate test made available.

- Tests that are placed on the Exclude List will be placed on the Exclude List after the determination of test validity. The new Exclude List will be made available to all Java EE 6 CTS licensees on the Java Licensee Engineering 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 Licensee Engineering (`https://java-partner.sun.com`) web site.

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

▼ Java EE 6 CTS Test Appeals Steps

1 **Java EE 6 CTS licensee writes a test challenge to Java Licensee Engineering contesting the validity of one or a related set of Java EE 6 tests.**

A challenge must be provided to Java Licensee Engineering and include the following information justifying why each test should be invalidated.

```
Date:
Licensee:
License Contact:
TCK name and version:
RI version:
Java SE version:
Exclude List version:
Specification version & section:
Problem Description:
Tests Affected:
Output (inline/attach .jtr file and traces):
```

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. 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 assigned Java Licensee engineer 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 EE 6 CTS (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.**

2.3 Reference Runtime for Java EE 6

The designated Reference Runtime for compatibility testing of version 6 of the Java Platform, Enterprise Editionis the Sun Reference Implementation of Java Platform, Enterprise Edition 6 for Solaris/SPARC, using the Sun Reference Implementation of Java SE 6 for Solaris/SPARC.

The Java SE 6 Reference Implementation is also available for Solaris/IA, Windows Vista/XP, Mac OS, and Linux.

2.4 Specifications for Java EE 6

The Specification for Java Platform, Enterprise Edition 6 is found on the JCP web site at http://jcp.org/en/jsr/detail?id=316 (`http://jcp.org/en/jsr/detail?id=316`).

2.5 Libraries for Java EE 6

The following location provides a list of packages that constitute the required class libraries for Java Platform, Enterprise Edition6 (except any class defined in a package that starts with `com.sun`):

`http://java.sun.com/javaee/6/docs/api/`

◆ ◆ ◆ CHAPTER 3

Procedure for Java Platform, Enterprise Edition 6 Web Profile Certification

This chapter describes the compatibility testing process and compatibility requirements for Java Platform, Enterprise Edition 6 Web Profile (Java EE 6 Web Profile). This chapter contains the following sections:

3.1 Certification Overview for Java EE 6 Web Profile

The certification process for Java EE 6 Web Profile consists of the following activities:

- Install the appropriate version of the Technology Compatibility Kit (TCK) and execute it in accordance with the instructions in this User's Guide.
- Ensure that you meet the requirements outlined in "3.2 Compatibility Requirements for Java EE 6 Web Profile" on page 35.
- Certify to Java Licensee Engineering that you have finished testing and that you meet all the compatibility requirements.

3.2 Compatibility Requirements for Java EE 6 Web Profile

The compatibility requirements for Java EE 6 Web Profile consist of meeting the requirements set forth by the rules and associated definitions contained in this section.

Note – As stated in Rule 15 in "3.2.2 Rules for Java EE 6 Web Profile Products" on page 39, part of the Java EE 6 CTS compatibility requirements is that your Java EE 6 Web Profile implementation must also pass the Contexts Dependency Injection for the Java EE Platform (JSR 299) and Bean Validation TCKs. See "1.5 TCK Requirements" on page 23 for more information about this requirement.

3.2.1 Definitions for Java EE 6 Web Profile Compatibility Requirements

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

Phrases of the form "Java EE 6 Web Profile <*term*>" are a reference to the <*term*> as defined in Chapter 2, "Procedure for Certification," in the associated *Java Test Compatibility Kit User's Guide*, available on the Java Licensee Engineering (`https://java-partner.sun.com`) web site.

TABLE 3–1 Definitions

Term	Definition
Application	A collection of components contained in a single application package (such as an EAR file or JAR file) and deployed at the same time using a Deployment Tool.
Computational Resource	A piece of hardware or software that may vary in quantity, existence, or version, which may be required to exist in a minimum quantity and/or at a specific or minimum revision level so as to satisfy the requirements of the Test Suite. Examples of computational resources that may vary in quantity are RAM and file descriptors. Examples of computational resources that may vary in existence (that is, may or may not exist) are graphics cards and device drivers. Examples of computational resources that may vary in version are operating systems and device drivers.
Compatibility 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.
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.
Container	An implementation of the associated Libraries, as specified in the Specifications, and a version of a Java SE Runtime Product, as specified in the Specifications, or a later version of a Java SE Runtime Product that also meets these compatibility requirements.

TABLE 3–1 Definitions *(Continued)*

Term	Definition
Deployment Tool	A tool used to deploy applications or components in a Product, as described in the Specifications.
Documented	Made technically accessible and made known to users, typically by means such as marketing materials, product documentation, usage messages, or developer support programs.
Edition	A Version of the Java Platform, Standard Edition Technology, or a Version of the Java Platform, Enterprise Edition Technology.
Endorsed Standard	A Java API defined through a standards process other than the Java Community Process. The Endorsed Standard packages are listed later in this chapter.
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 *Java Platform, Enterprise Edition 6 Web Profile Specification*.
JSP Page	A text-based document that uses JavaServer Pages technology.
JSP Page Implementation Class	A program constructed by transforming the JSP Page text into a Java language program using the transformation rules described in the Specifications.
Libraries	The class libraries, as specified through the Java Community Process (JCP), for the Technology Under Test. The Libraries for Java EE 6 Web Profile are listed at the end of this chapter.
Location Resource	A location of classes or native libraries that are components of the test tools or tests, such that these classes or libraries may be required to exist in a certain location in order to satisfy the requirements of the test suite. For example, classes may be required to exist in directories named in a `CLASSPATH` variable, or native libraries may be required to exist in directories named in a `PATH` variable.
Maintenance Lead	The Java Community Process member responsible for maintaining the Specification, reference implementation, and TCK for the Technology. Sun is the Maintenance Lead for Java EE 6 Web Profile.

TABLE 3–1 Definitions *(Continued)*

Term	Definition
Operating Mode	Any Documented option of a Product that can be changed by a user in order to modify the behavior of the Product. For example, an Operating Mode can be binary (enable/disable optimization), an enumeration (select from a list of protocols), or a range (set the maximum number of active threads).
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 6 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.
Runtime	The Containers specified in the Specifications.
Security Resource	A security privilege or policy necessary for the proper execution of the Test Suite. For example, the user executing the Test Suite will need the privilege to access the files and network resources necessary for use of the Product.
Specifications	The documents produced through the 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 EE 6 Web Profile.
Test Suite	The requirements, tests, and testing tools distributed by the Maintenance Lead as applicable to a given Version of the Technology.
Version	A release of the Technology, as produced through the Java Community Process.
WSDL-to-Java Output	Output of a WSDL-to-Java tool that is required for Web service deployment and invocation.

TABLE 3–1 Definitions *(Continued)*

Term	Definition
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 *Java Platform, Enterprise Edition 6 Web Profile Specification*.

3.2.2 Rules for Java EE 6 Web Profile 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 Compatibility 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 Compatibility 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 Compatibility Tests to fail may be tested using a Product Configuration that allows all Compatibility 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 Compatibility Tests may have properties that may be changed. Properties that can be changed are identified in the JavaTest Environment (`.jte`) files in the `bin` directory of the Test Suite installation. Apart from changing such properties and other allowed modifications described in this User's Guide (if any), no source or binary code for a Compatibility Test may be altered in any way without prior written permission. Any such allowed alterations to the Compatibility 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 Compatibility 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.

   a. If a Product includes Technologies in addition to the Technology Under Test, then it must contain the full set of combined public and protected classes and interfaces. The API of the Product must contain the union of the included Technologies. No further modifications to the APIs of the included Technologies are allowed.

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

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

      For example, the Java EE 6 Web Profile Maintenance Lead could authorize use of a newer version of a Java technology such as JavaServer Faces, that is also available independently of Java EE 6 Web Profile and is evolving quickly to meet market demands.

8. Except for tests specifically required by this TCK to be rebuilt (if any), the binary Compatibility 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. Each Container must make technically accessible all Java SE Runtime interfaces and functionality, as defined by the Specifications, to programs running in the Container, except only as specifically exempted by these Rules.

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

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

12. The presence of a Java language comment or Java language directive in a JSP Page that specifies "java" as the scripting language, when processed by a web Container, must not cause the functional programmatic behavior of that JSP Page to vary from the functional programmatic behavior of that JSP Page in the absence of that Java language comment or Java language directive.
13. The contents of any fixed template data (defined by the Specifications) in a JSP Page, when processed by a web Container, must not affect the functional programmatic behavior of that JSP Page, except as defined by the Specifications.
14. The functional programmatic behavior of a JSP Page that specifies "java" as the scripting language must be equivalent to the functional programmatic behavior of the JSP Page Implementation Class constructed from that JSP Page.
15. Compatibility testing for the Java EE 6 Web Profile consists of running the tests in this Technology Compatibility Kit defined as required by the designation of the `javaee_web_profile` keyword. If optional technologies defined in the Java EE 6 Web Profile platform are implemented in addition to the Java EE 6 Web Profile, corresponding tests within this TCK for those additional technologies must also be run. If tests for a particular technology are not available in the Java EE 6 CTS test suite, then the standalone TCK(s) for said technologies must be run instead. In this case, in addition to the compatibility rules outlined in this CTS User's Guide, Java EE 6 Web Profile implementations must also adhere to any compatibility rules defined in the documentation for these additional TCKs.
16. Source Code in WSDL-to-Java Output when compiled by a Reference Compiler must execute properly when run on a Reference Runtime.
17. Source Code in WSDL-to-Java Output must be in source file format defined by the Java Language Specification (JLS).
18. Java-to-WSDL Output must fully meet W3C requirements for the Web Services Description Language (WSDL) 1.1.
19. A Java-to-WSDL Tool must not produce Java-to-WSDL Output from source code that does not conform to the Java Language Specification (JLS).

3.2.3 Java EE 6 Web Profile 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. Sun, as Java EE 6 Web Profile Maintenance Lead, will authorize representatives from the Java Licensee Engineering organization 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 EE 6 CTS 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 EE 6 CTS. 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 the Java Licensee Engineering organization and include all relevant information as described in step 1 in "3.2.3 Java EE 6 Web Profile Test Appeals Process" on page 41. The process used to determine the validity or invalidity of a test (or related group of tests) is also described in that section.

All tests found to be invalid will either be placed on the Exclude List for that version of the Java EE 6 CTS or have an alternate test made available.

- Tests that are placed on the Exclude List will be placed on the Exclude List after the determination of test validity. The new Exclude List will be made available to all Java EE 6 CTS licensees on the Java Licensee Engineering 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 Licensee Engineering (`https://java-partner.sun.com`) web site.

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

Java EE 6 CTS Test Appeals Steps

1. **Java EE 6 CTS licensee writes a test challenge to Java Licensee Engineering contesting the validity of one or a related set of Java EE 6 Web Profile tests.**

   A challenge must be provided to Java Licensee Engineering and include the following information justifying why each test should be invalidated.

```
Date:
Licensee:
License Contact:
TCK name and version:
RI version:
Java SE version:
Exclude List version:
Specification version & section:
Problem Description:
Tests Affected:
Output (inline/attach .jtr file and traces):
```

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. 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 assigned Java Licensee engineer 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 EE 6 CTS (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.**

3.3 Reference Runtime for Java EE 6 Web Profile

The designated Reference Runtime for compatibility testing ofJava EE 6 Web Profile is the Sun Reference Implementation of Java EE 6 Web Profile for Solaris/SPARC, using the Sun Reference Implementation of Java SE 6 for Solaris/SPARC.

The Java SE 6 Reference Implementation is also available for Solaris/IA, Windows, Mac OS, and Linux.

3.4 Specifications for Java EE 6 Web Profile

The Specification for Java Platform, Enterprise Edition 6 Web Profile is found on the JCP web site at http://jcp.org/en/jsr/detail?id=316 (`http://jcp.org/en/jsr/detail?id=316`).

3.5 Libraries for Java EE 6 Web Profile

The following location provides a list of packages that constitute the required class libraries for Java EE 6 Web Profile (except any class defined in a package that starts with `com.sun`):

`http://java.sun.com/javaee/6/docs/api/`

This URL includes javadocs for the full Java EE 6. The APIs for Java EE 6 Web Profile are a subset of the full Java EE 6 APIs. See the *Java EE 6 Web Profile Specification* under http://jcp.org/en/jsr/detail?id=316 (`http://jcp.org/en/jsr/detail?id=316`) for the list of technologies (and packages) that are included in the Java EE 6 Web Profile.

◆ ◆ ◆ CHAPTER 4

Installation

This chapter explains how to install the Java EE 6 CTSsoftware and perform a sample test run to verify your installation and familiarize yourself with the CTS.

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

4.1 Installing the Software

Installing the software required to run the Java EE 6 CTS involves three general steps:

- "Installing the Java Platform, Enterprise Edition RI" on page 45
- "Installing the Java EE 6 CTS" on page 46
- "Verifying Your Installation (Optional)" on page 47

▼ Installing the Java Platform, Enterprise Edition RI

Before You Begin If a Java Platform, Enterprise Edition Reference Implementation (RI) is already installed on your workstation, it is recommended that you shut it down and start with a new, clean RI installation.

1 **Create or change to the directory in which you will install the Java Platform, Enterprise Edition RI.**

2 **Copy or download the Java Platform, Enterprise Edition RI from the Java Partner Engineering (`https://javapartner.sun.com`) Web site.**

3 **Unzip the Java Platform, Enterprise Edition RI bundle.**

This creates a `javaeetck` directory, which will be your `JAVAEE_HOME`.

4 **Change to the `javaeetck` directory.**

```
cd javaeetck
```

5 **Install the J2SE SDK 6 bundle, if it is not already installed.**

Refer to the JDK installation instructions for details. The J2SE SDK 6 bundle can be downloaded from `http://java.sun.com/j2se/downloads/index.jsp`.

6 **Set the following environment variables:**

- `JAVAEE_HOME` to the RI directory you just created
- `JAVA_HOME` to the J2SE SDK 6 you want to use

7 **Set up the Java Platform, Enterprise Edition RI by executing the following Ant target:**

```
<JAVAEE_HOME>/lib/ant/bin/ant -buildfile ./setup.xml setup
```

If you get a port error when running the setup target, modify your `setup.xml` file or pass the correct properties on the Ant command line. The important thing here is to make sure whatever values you use for the Ant target match those that you use in the `<TS_HOME>/bin/ts.jte` CTS configuration file. Some Java Platform, Enterprise Edition RI server configuration properties and their default values in `setup.xml` include:

- `<property name="domain.name" value="domain1"/>`
- `<property name="instance.name" value="server"/>`
- `<property name="admin.user" value="admin"/>`
- `<property name="admin.password" value=""/>`
- `<property name="admin.port" value="4848"/>`
- `<property name="instance.port" value="8080"/>`
- `<property name="orb.port" value="3700"/>`
- `<property name="imq.port" value="7676"/>`
- `<property name="https.port" value="1043"/>`

8 **Start the Java Platform, Enterprise Edition RI by executing the following command:**

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

▼ Installing the Java EE 6 CTS

Complete the following procedure to install the Java EE 6 CTS on a system running the Solaris, Linux, or Windows operating system.

Note – When installing in the Windows environment, the Java Platform, Enterprise Edition RI, J2SE, and CTS should be installed on the same drive. If you must install these components on different drives, be sure to change the `ri.applicationRoot` and `s1as.applicationRoot` properties as needed in the `<TS_HOME>/bin/ts.jte` CTS configuration file. See "5.4.2 Windows-Specific Properties" on page 64 for more information.

---

1 **Copy or download the CTS 6 software from the Java Partner Engineering (`https://javapartner.sun.com`) Web site.**

2 **Change to the directory in which you want to install the Java Platform, Enterprise Edition 6 Compatibility Test Suite software and use the `unzip` command to extract the bundle:**

```
cd <install_directory>
unzip javaeetck-nnn.zip
```

This creates the `javaeetck` directory. The `<install_directory>/javaeetck` directory will be `TS_HOME`.

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

4 **Unset your current `ANT_HOME` variable, if it is currently set.**

The CTS test suite uses the Ant package included in the CTS bundle.

5 **After you complete the installation, follow the directions in Chapter 5, "Setup and Configuration," to set up and configure the Java EE 6 CTS test suite.**

▼ Verifying Your Installation (Optional)

After installing the Java Platform, Enterprise Edition RI and the CTS bundle, you may want to quickly verify your installation by running the CTS samples against the Java RI. Again, this procedure is optional, and complete configuration instructions are provided in Chapter 5, "Setup and Configuration."

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

- `javaee.home=<JAVAEE_HOME>`
- `javaee.home.ri=<JAVAEE_HOME>`

2 **Run the following commands:**

```
<TS_HOME>/tools/ant/bin/ant config.vi
<TS_HOME>/tools/ant/bin/ant -f xml/s1as.xml start.javadb
<TS_HOME>/tools/ant/bin/ant init.
```

CHAPTER 5

Setup and Configuration

This chapter describes how to set up the Java EE 6 CTS test suite and configure it to work with your test environment. It is recommended that you first set up the testing environment using the Java EE 6 RI and then with your Java EE 6 server.

This chapter includes the following topics:

5.1 Allowed Modifications

You can modify the following test suite components only:

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

5.2 Configuring the Test Environment

The instructions in this section and in "Configuring Your Application Server as the VI" on page 55 step you through the configuration process for the Sun Solaris, Microsoft Windows, and Linux platforms.

All CTS test configuration procedures are based on running the Ant scripts against a set of build targets. The primary location of any configuration settings you are likely to make is the `<TS_HOME>/bin/ts.jte` environment file. You may also want to modify the `javaee_vi.xml` and `initdb.xml` Ant configuration files and the vendor-specific SQL files. These two files contain predefined Ant targets that are implemented such that they automatically configure the Java EE 6 RI and its associated database in order to pass the CTS. A licensee may choose to implement these targets to work with their server environment to perform the steps described in "Configuring Your Application Server as the VI" on page 55.

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

▼ General Configuration Steps

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

1 **Identify the software pieces and assemble them into the Java EE 6 platform to be tested for certification.**

2 **Implement the porting package APIs.**

Some functionality in the Java EE 6 platform is not completely specified by an API. To handle this situation, the Java EE 6 CTS test suite defines a set of interfaces in the `com.sun.cts.porting` package, which serve to abstract any implementation-specific code. You must create your own implementations of the porting package interfaces to work with your particular Java EE 6 server environment. See "11.2 Porting Package APIs" on page 141 for additional information about the porting APIs. API documentation for the porting package interfaces is available in the `<TS_HOME>/docs/api` directory.

3 **Set up the Java Platform, Enterprise Edition Reference Implementation (RI) server.**

See "Configuring the Java Platform, Enterprise Edition RI as the VI" on page 53 for a list of the modifications that must be made to run CTS against the Java Platform, Enterprise Edition RI.

4 **Set up the vendor's Java Platform, Enterprise Edition server implementation (VI).**

See "Configuring Your Application Server as the VI" on page 55 for a list of the modifications that must be made to run CTS against the vendor's Java Platform, Enterprise Edition server.

5 **Modify the `ts.jte` file to match your test environment.**

Use the JavaTest configuration GUI to configure your basic test environment, as described in "5.4 Modifying Environment Settings for Specific Technology Tests" on page 63. Also refer to Appendix B, "`ts.jte` Modifiable Properties," for information about the various `ts.jte` properties you may want to modify for your particular test environment.

Note – If the Java Platform, Enterprise Edition RI is installed on a machine other than the machine on which the vendor's Java Platform, Enterprise Edition server has been installed, the `javaee.home.ri` property must be defined so both sets of porting package classes can be accessed from the remote machine (such as by using `/net/machine-name/vendors-JAVAEE_HOME`, for example).

6 **Start your backend database.**

Execute the `start.javadb` Ant target:

```
<TS_HOME>/tools/ant/bin/ant -f xml/s1as.xml start.javadb
```

7 **Configure your backend database.**

Execute the `init.javadb` Ant target. If you are not using JavaDB as your backend database, refer to Appendix F, "Configuring Your Backend Database."

Note – If you are using MySQL or MS SQL Server as your backend database, see "5.4.25 Backend Database Setup" on page 108 for additional database setup instructions.

8 **Validate your configuration.**

Run the sample tests provided. If the tests pass, your basic configuration is valid. See "7.3 Validating Your Test Configuration" on page 121 for information about using JavaTest to run the sample tests.

9 **Run the CTS tests.**

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

5.3 Configuring a Java Platform, Enterprise Edition Server

This section describes how to configure the Java Platform, Enterprise Edition server under test. You can run the CTS tests against the Sun Java Platform, Enterprise Edition RI or your own Java Platform, Enterprise Edition server. When performing interoperability (interop) tests or web service-based tests, you will be running two Java Platform, Enterprise Edition servers, one of which must be the Sun Java Platform, Enterprise Edition RI using the bundled JavaDB database.

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

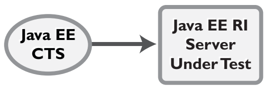
- **Reference Implementation (RI)** — Sun Java Platform, Enterprise Edition RI
- **Vendor Implemention (VI)** — Java Platform, Enterprise Edition implementation from a vendor other than Sun; typically, the goal of running the Java Platform, Enterprise Edition CTS is to certify a Java Platform, Enterprise Edition VI; in some cases, for purposes of familiarizing yourself with the Java Platform, Enterprise Edition CTS, you may choose to run the Sun Java Platform, Enterprise Edition RI as the VI

- **Bundled JavaDB** — JavaDB database bundled with the Sun Java Platform, Enterprise Edition RI

5.3.1 Java Platform, Enterprise Edition Server Configuration Scenarios

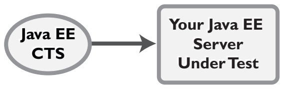
There are three general scenarios for configuring Java Platform, Enterprise Edition servers for Java EE 6 CTS testing:

- **Configure the Java Platform, Enterprise Edition RI as the server under test**



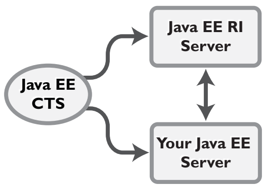
  Use the Java Platform, Enterprise Edition RI as the Java Platform, Enterprise Edition VI; you may want to do this as a "sanity check" to make sure you are comfortable with using the Java EE 6 CTS against a known standard RI with certified sample applications before proceeding with tests against your Java Platform, Enterprise Edition VI. See "Configuring the Java Platform, Enterprise Edition RI as the VI" on page 53 for instructions.

- **Configure your Java Platform, Enterprise Edition VI as Server Under Test**



  This is the primary goal of using the Java EE 6 CTS; you will eventually need to configure the Java Platform, Enterprise Edition implementation you want to certify. See "Configuring Your Application Server as the VI" on page 55 for instructions.

- **Configure two Java Platform, Enterprise Edition servers for the purpose of interop testing**



Rebuildable tests and Interop tests require that you configure two Java Platform, Enterprise Edition servers on one or two machines. One server will be your Java Platform, Enterprise Edition VI running a database of your choice with JDBC 3.0-compliant drivers. The second server must be the Sun Java Platform, Enterprise Edition RI using the bundled JavaDB database. See "Configuring a Java Platform, Enterprise Edition RI and VI for Interop/Rebuildable Tests" on page 60 for instructions.

In terms of the Java EE 6 CTS, all CTS configuration settings are made in the `<TS_HOME>/bin/ts.jte` file. When configuring a Java Platform, Enterprise Edition server, the important thing is to make sure that the settings you use for your server match those in the CTS `ts.jte` file.

These configuration scenarios are described in the following sections.

▼ Configuring the Java Platform, Enterprise Edition RI as the VI

To configure the Sun Java Platform, Enterprise Edition RI as the server under test-that is, to use the Java Platform, Enterprise Edition RI as the Java Platform, Enterprise Edition VI-follow the steps listed below. In this scenario, the goal is simply to test the Java Platform, Enterprise Edition RI against the CTS for the purposes of familiarizing yourself with CTS test procedures. You may also want to refer to the Quick Start guides included with the Java EE 6 CTS for similar instructions.

1 **Set server properties in your `<TS_HOME>/bin/ts.jte` file to suit your test environment.**

Be sure to set the following properties:

a. **Set the `webServerHost` property to the name of the host on which your Web server is running that is configured with the Sun RI.**

The default setting is `localhost`.

b. **Set the `webServerPort` property to the port number of the host on which the Web server is running and configured with the Sun RI.**

The default setting is `8001`.

c. **Set the `wsgen.ant.classname` property to the Vendor's implementation class that mimics the Sun RI Ant task that in turn calls the Sun wsgen Java-to-WSDL tool.**

The default setting is `com.sun.tools.ws.ant.WsGen`.

d. **Set the `wsimport.ant.classname` property to the Vendor's implementation class that mimics the Sun RI Ant task that in turn calls the Sun wsimport Java-to-WSDL tool.**

The default setting is `com.sun.tools.ws.ant.WsImport`.

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

   The default setting for the Sun RI porting implementation is `com.sun.ts.lib.implementation.sun.common.SunRIURL`.

As mentioned previously, these settings can vary, but must match whatever you used when setting up the Java Platform, Enterprise Edition RI server. See Appendix B, "`ts.jte` Modifiable Properties," for more details about `ts.jte` properties.

2 **Install the Java Platform, Enterprise Edition RI and configure basic settings, as described in Chapter 4, "Installation."**

3 **Start the Java Platform, Enterprise Edition RI Application Server.**

Refer to the Application Server documentation for complete instructions.

4 **Run the Java Platform, Enterprise Edition RI configuration Ant target.**

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

This also starts your configured database.

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

```
ant -Dskip.config.connector=true config.vi
```

---

5 **Initialize your database.**

For example, to initialize a JavaDB database:

```
$TS_HOME/tools/ant/bin/ant init.javadb
```

See Appendix F, "Configuring Your Backend Database," for configuration and initialization instructions for other databases.

6 **Build the special web services clients.**

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

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

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

```
<TS_HOME>/tools/ant/bin/ant build.special.webservices.clients
```

This predeploys all the special webservices endpoints, builds all the special webservices clients, and then undeploys the special webservices endpoints. See "11.2.2 Ant-Based Deployment Interface" on page 142 for more information about the Ant-based deployment interface, including guidelines for creating your own Ant-based deployment implementation.

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

▼ Configuring Your Application Server as the VI

To use a Java Platform, Enterprise Edition server other than the Sun Java Platform, Enterprise Edition RI, follow the steps below.

1 **Set server properties in your `<TS_HOME>/bin/ts.jte` file to suit your test environment.**

Be sure to set the following properties:

a. **Set the `webServerHost` property to the name of the host on which your Web server is running that is configured with the Sun RI.**

The default setting is `localhost`.

b. **Set the `webServerPort` property to the port number of the host on which the Web server is running and configured with the Sun RI.**

The default setting is `8001`.

c. **Set the `wsgen.ant.classname` property to the Vendor's implementation class that mimics the Sun RI Ant task that in turn calls the Sun `wsgen` Java-to-WSDL tool.**

The default setting is `com.sun.tools.ws.ant.WsGen`.

d. **Set the `wsimport.ant.classname` property to the Vendor's implementation class that mimics the Sun RI Ant task that in turn calls the Sun `wsimport` Java-to-WSDL tool.**

The default setting is `com.sun.tools.ws.ant.WsImport`.

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

These settings can vary, but must match whatever you used when setting up your Java Platform, Enterprise Edition server. See Appendix B, "`ts.jte` Modifiable Properties," for more details about `ts.jte` properties.

2 **Install the Java Platform, Enterprise Edition VI and configure basic settings.**

If you want to configure your Java Platform, Enterprise Edition server using Ant configuration target similar to the target for the Java Platform, Enterprise Edition RI, as described in Chapter 4, "Installation," you will need to modify the `<TS_HOME>/bin/xml/javaee_vi.xml` file to implement the defined Ant targets for your application server. Then run:

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

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

3 **Provide alternate endpoint and WSDL URLs, if necessary.**

The `<TS_HOME>/bin` directory contains the following `.dat` files:

- `jaxrpc-url-props.dat`
- `jaxws-url-props.dat`
- `jws-url-props.dat`
- `webservices12-url-props.dat`

These files contain the webservice endpoint and WSDL URLs that the CTS tests use when running against the Sun RI. In the Sun porting package used by the CTS, 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 to run the CTS tests in your environment. However, you MUST NOT modify the existing `.dat` files, but instead make any necessary changes in your own porting implementation class to transform the URLs appropriately for your environment.

4 **Start your database.**

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

6 **Initialize your database for CTS tests.**

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

(JavaDB Example)

```
<TS_HOME>/tools/ant/bin/ant -Dtarget.dml.file=tssql.stmt /
-Ddml.file=javadb/javadb.dml.sql copy.dml.file
```

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

```
<TS_HOME>/tools/ant/bin/ant init.<database>
```

For example, to initialize a JavaDB database:

```
<TS_HOME>/tools/ant/bin/ant init.javadb
```

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

7 **Start your Java Platform, Enterprise Edition server.**

8 **Set up required users and passwords.**

a. **Set up database users and passwords.**

The Java EE 6 CTS requires several user names, passwords, and user-to-role mappings. These need to match those set in your `ts.jte` file. By default, `user1`, `user2`, `password1`, and `password2` are set to `cts1`.

b. **Set up users and passwords for your Java Platform, Enterprise Edition server.**

For the purpose of running the CTS test suite, these should be set as follows:

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

Note that adding the `asadmin` group is only necessary when running against the Java Platform, Enterprise Edition Application Server. It is required in such case because the management EJB (MEjb) in the Java Platform, Enterprise Edition server is protected with the `asadmin` group. Other appservers may or may not choose to protect their MEjb. If necessary for your appserver implementation, you should also add the group name with which your MEjb is protected.

Also make sure the principal to role-mappings that are specified in the runtime XML files (see "5.4.20.1 Mapping Roles" on page 93) are properly mapped in your environment. Note that the principal-to-role mappings may vary for each application.

9 **Make sure that the appropriate JDBC 3.0-compliant database driver class, any associated database driver native libraries, and the correct database driver URL are available.**

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

11 **Configure your Java Platform, Enterprise Edition server to use the appropriate logical name (`jdbc/DBTimer`) when accessing your EJB timer.**

12 **Provide access to a JNDI lookup service.**

13 **Provide access to a Web server.**

14 Provide access to a JavaMail server that supports the SMTP protocol.

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

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

---

16 Install the client-side certificate in the `trustStore` on the Java Platform, Enterprise Edition server. See "5.4.20 CSIv2 Test Setup" on page 91 for more information.

Certificates are located `<TS_HOME>/bin/certificates`. Use the certificate that suits your environment.

a. `cts_cert` – For importing the CTS client certificate into a `truststore`

b. `clientcert.jks` – Used by the J2SE runtime to identify the CTS client's identity

c. `clientcert.p12` – Contains CTS client certificate in `pkcs12` format

17 Append the file `<TS_HOME>/bin/server_policy.append` to the Java policy file(s) on your Java Platform, Enterprise Edition server.

This file contains the grant statements used by the test harness, signature tests, and API tests.

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

19 Make the appropriate transaction interoperability setting on the Java Platform, Enterprise Edition server and the server that is running the Java Platform, Enterprise Edition RI.

See "5.4.20 CSIv2 Test Setup" on page 91.

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

For example, see "5.4.20 CSIv2 Test Setup" on page 91 for configuration settings required for CSIv2 tests.

21 Restart your Java Platform, Enterprise Edition server.

22 Build the special Web services clients.

This step may be bypassed at this time if you are not going to immediately run the tests under `<TS_HOME>/src/com/sun/ts/tests/webservices12`. However, you must return to this configuration section and complete it inorder to run these tests.

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

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

a. **Install the Sun Java Platform, Enterprise Edition Reference Implementation.**

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

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

   - **Set the `javaee.home.ri` property to the location where the Sun Java Platform, Enterprise Edition Reference Implementation is installed.**
   - **Set the `wsgen.ant.classname` property to the Sun Java Platform, Enterprise Edition application server Ant task that in turn calls the Sun wsimport Java-to-WSDL tool. It must be set to:**

     ```
     com.sun.tools.ws.ant.WsGen
     ```
   - **Set the `wsgen.classpath` property to:**

     ```
     ${javaee.classes.ri}:${tools.jar}
     ```
   - **Set the `wsimport.ant.classname` property to the Sun Java Platform, Enterprise Edition application server Ant task that in turn calls the Sun wsimport WSDL-to-Java tool.**

     It must be set to `com.sun.tools.ws.ant.WsImport`
   - **Set the `wsimport.classpath` property to the following value:**

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

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

   ```
   <TS_HOME>/tools/ant/bin/ant build.special.webservices.clients
   ```

   This predeploys all the special Web services endpoints, builds all the special webservices clients, and then undeploys the special webservices endpoints. See "11.2.2 Ant-Based Deployment Interface" on page 142 for more information about the Ant-based deployment interface, including guidelines for creating your own Ant-based deployment implementation.

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

   - `wsgen.ant.classname`
   - `wsgen.classpath`

- `wsimport.ant.classname`
- `wsimport.classpath`

For more information regarding their use in the CTS, refer to Appendix H, "Rebuilding the JAX-WS and JWS Tests."

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

- `specialcases.defaultserviceref.wsdlloc`
- `specialcases.nameattrserviceref.wsdlloc`
- `specialcases.providerserviceref.wsdlloc`

23 **Continue on to Chapter 7, "Executing Tests."**

▼ Configuring a Java Platform, Enterprise Edition RI and VI for Interop/Rebuildable Tests

Use the following procedure to configure the Java EE 6 CTS for interoperability and rebuildable testing. Note that you must complete all of the setup instructions in this section and all of the steps in "5.4.20 CSIv2 Test Setup" on page 91 before you run the CSIv2 tests.

1 **Install and configure basic settings for the Java Platform, Enterprise Edition VI and the Java Platform, Enterprise Edition RI.**

These procedures are described earlier in this section. You can run the Java Platform, Enterprise Edition servers on separate machines or on the same machine. If running both servers on the same machine, be careful to avoid conflicting properties (for example, port settings).

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

- `create.cmp.tables=true`
- `javaee.home=`<*Java Platform, Enterprise Edition VI installation directory (JAVAEE_HOME)*>
- `javaee.home.ri=`<*Java Platform, Enterprise Edition RI installation directory*>
- `mailuser1=`<*valid email address*>
- `mailHost=`<*valid SMTP server*>
- `orb.host=`<*host where the Java Platform, Enterprise Edition VI naming server is running*>
- `orb.port=`<*port where the Java Platform, Enterprise Edition VI naming service is running*>
- `orb.host.ri=`<*host where the Java Platform, Enterprise Edition RI naming service is running*>
- `orb.port.ri=`<*port where the Java Platform, Enterprise Edition RI naming service is running*>

- `webServerHost=`<*host where the Java Platform, Enterprise Edition VI Web server is running*>
- `webServerPort=`<*port where the Java Platform, Enterprise Edition VI Web server is running*>
- `webServerHost.2=`<*host where the Java Platform, Enterprise Edition RI Web server is running*>
- `webServerPort.2=`<*port where the Java Platform, Enterprise Edition RI Web server is running*>
- `securedWebServicePort=`<*port where the Java Platform, Enterprise Edition secure web service is running*>
- `securedWebServicePort.2=`<*port where the Java Platform, Enterprise Edition RI secure web service is running*>
- `porting.ts.deploy2.class.1=`<*vendor-provided deployment porting class*>
- `porting.ts.login.class.1=`<*vendor-provided login porting class*>
- `porting.ts.url.class.1=`<*vendor-provided URL porting class*>
- `porting.ts.jms.class.1=`<*vendor-provided JMS porting class*>
- `porting.ts.tsHttpsURLConnection.class.1=`<*vendor-provided HttpsURLConnection-class*>

Note – ???The `create.interop.tables.only=true` property does not exist in the `ts.jte` file by default but can be added if needed for creating interop tables.

???For the Java Platform, Enterprise Edition RI, you must set `create.cmp.tables=true` when you set `create.interop.tables.only=true`.

Note – As a general rule, `ts.jte` properties ending with the suffix ".2" are RI server properties that rarely need to be changed. Properties ending with the suffix ".1" are VI server properties that are more likely to require modifying.

---

3 Configure both Java Platform, Enterprise Edition servers.

Run the following Ant targets:

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

If you have not implemented the `config.vi` Ant target for your Java Platform, Enterprise Edition server, perform the steps shown in "Configuring Your Application Server as the VI" on page 55.

4 **Configure rebuildable tests, if applicable at this time.**

Java EE 6 CTS Rebuildable Tests are located under `<TS_HOME>/src/com/sun/ts/tests/jaxws` and `<TS_HOME>/src/com/sun/ts/tests/jws`. If you are interested in running only these tests, see Appendix H, "Rebuilding the JAX-WS and JWS Tests," for instructions on how to rebuild, and then continue on to Chapter 7, "Executing Tests," for instructions on executing tests. If you would like to run tests under `<TS_HOME>/src/com/sun/ts/tests/interop`, continue to the next step.

5 **Add `<J2EE_HOME_RI>/lib/j2ee-svc.jar` to your application server's `CLASSPATH`.**

This jar file is part of the EJB interoperability architecture. It contains only Sun's implementations of the required system value classes, and makes it unnecessary for you to put the entire appserv-rt.jar in your `CLASSPATH`.

6 **Initialize the databases using the appropriate Ant targets.**

a. **Log in to the machine running the RI database and execute the following command:**

```
<TS_HOME>/tools/ant/bin/ant init.javadb
```

b. **???Change the `create.interop.tables.only` to false in the `<TS_HOME>/bin/ts.jte` file.**

c. **Log into the machine running the VI database and execute the following command:**

```
<TS_HOME>/tools/ant/bin/ant init.<database>
```

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

7 **Start the standalone deployment server in a separate shell on the same host as the CTS harness.**

The default deployment porting implementation goes through a standalone deployment server with a dedicated classpath. To start this standalone server, change to the `<TS_HOME>/bin` directory and execute the `start.auto.deployment.server` Ant task.

The standalone server is basically an RMI server used to copy archives to the RI server's `autodeploy` directory. A separate VM is necessary to avoid classloading conflicts that could occur when the VI server is also a version of RI server.

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

For example, see "5.4.20 CSIv2 Test Setup" on page 91 for configuration settings required for CSIv2 tests.

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

5.4 Modifying Environment Settings for Specific Technology Tests

Before you can run any of the technology-specific Java EE 6 CTS tests, you must supply certain information that JavaTest needs to run the tests in your particular environment. This information exists in the `<TS_HOME>/bin/ts.jte` environment file. This file contains sets of name/value pairs that are used by the tests. You need to assign a valid value for your environment for all of the properties listed in the sections that follow.

Note – This section only discusses a small subset of the properties you can modify. Refer to Appendix B, "`ts.jte` Modifiable Properties," for information about the many other `ts.jte` properties you may want to modify for your particular test environment.

This section includes the following topics:

5.4.1 Test Harness Setup

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

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

- The `harness.temp.directory` property specifies a temporary directory that the harness creates and to which the CTS harness and tests write temporary files. The default setting should not need to be changed.
- The `harness.log.port` property specifies the port that server components of the tests use to send logging output back to JavaTest. If the default port is not available on the machine running JavaTest, you must edit this property and set it to an available port. The default setting is `2000`.
- The `harness.log.traceflag` property is used to turn on or turn off verbose debugging output for the tests. The value of the property is set to `false` by default. Set the property to `true` to turn debugging on.
- The `deployment_host.1` and `deployment_host.2` properties specify the systems where the vendor's Java Platform, Enterprise Edition server and the Java Platform, Enterprise Edition RI server are running. By default, JavaTest will use the `orb.host` and `orb.host.ri` systems, which are set in the `ts.jte` file.
- The porting class `.1` and `.2` property sets specify the class names of porting class implementations. By default, both property sets point to the Java Platform, Enterprise Edition RI-specific classes. To run the interoperability tests, do not modify the `.2` set. These properties should always point to the Java Platform, Enterprise Edition RI classes. Modify the `.1` set to point to implementations that work in your specific Java Platform, Enterprise Edition environment. See "Configuring a Java Platform, Enterprise Edition RI and VI for Interop/Rebuildable Tests" on page 60 for additional information about setting these properties.
- The `-Dcts.tmp` option for the `testExecute` and `testExecuteAppClient` commands in the `ts.jte` file have been set. This Java option tells the test suite the location to which the test suite will write temporary files.

5.4.2 Windows-Specific Properties

When configuring the Java EE 6 CTS for the Windows environment, set the following properties in `<TS_HOME>/bin/ts.jte`:

- `pathsep` to semicolon (pathsep=;)
- `s1as.applicationRoot` to the drive on which you have installed CTS (for example, `s1as.applicationRoot=C:`)

When installing in the Windows environment, the Java Platform, Enterprise Edition RI, J2SE, and CTS should all be installed on the same drive. If you must install these components on different drives, also change the `ri.applicationRoot` property in addition to the `pathsep` and `s1as.applicationRoot` properties; for example:

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

Note – When configuring the RI and CTS for the Windows environment, never specify drive letters in any path properties in `ts.jte`.

5.4.3 Test Execution Command Setup

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

- `command.testExecute`
- `command.testExecuteAppClient`
- `command.testExecuteAppClient2`

If these commands do not meet your needs, you can define separate entries for the UNIX and Windows environments. Edit either the `ts_unix` or `ts_win32` test execution properties in the `ts.jte` file. For UNIX, these properties are:

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

For Windows, these properties are:

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

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

Note – The default settings are specific to the Java Platform, Enterprise Edition RI. If you are not using the Java Platform, Enterprise Edition RI, adjust these properties accordingly.

5.4.4 Servlet Test Setup

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

```
ServletClientThreads=[2X size of default servlet instance pool]
```

The `ServletClientThreads` property configures the number of threads used by the client for the `SingleThreadModel` servlet test. If your container implementation supports pooling of `SingleThreadModel` servlets, set the value of the `ServletClientThreads` property to twice the value of the default servlet instance pool size. If your container implementation only maintains a single instance of a `ServletClientThreads` servlet, use the default value of `2`.

5.4.5 JDBC Test Setup

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

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

5.4.6 Standalone RMI/IIOP Server Test Setup

The standalone RMI/IIOP server testing verifies that Java Platform, Enterprise Edition application components can access and communicate with an external RMI/IIOP server application.

The `start.rmiiiop.server` Ant target uses the `ts.classpath` property setting from the `ts.jte` file when starting the standalone RMI/IIOP server application. The standalone RMI/IIOP server application must start up using the ORB that comes with the Java Platform, Enterprise Edition RI.

Make sure that `ts.classpath` property contains the Java Platform, Enterprise Edition RI JAR files and classes and that the following properties have been set in the `ts.jte` file:

```
rmi.http.server.host=[hostname]
rmi.http.server.port=[port-number]
```

The `rmi.http.server.host` and `rmi.http.server.port` properties must be set to the host and port where the standalone RMI/IIOP http server is running. The default values for these properties are `localhost` and `10000`, respectively.

▼ To start the standalone RMI/IIOP server

● **Execute the following command:**

```
<TS_HOME>/tools/ant/bin/ant start.rmiiiop.server
```

5.4.7 JavaMail Test Setup

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

```
mailuser1=[user@domain]
transport_protocol=[smtp]
mailFrom=[user@domain]
mailHost=mailserver
```

- Set the `mailuser1` property to a valid mail address. Mail messages generated by the JavaMail tests are sent to the specified address.
- Set the `transport_protocol` property to a valid mail protocol. The default setting is `smtp`.
- Set the `mailFrom` property to a mail address from which mail messages that the JavaMail tests generate will be sent.
- Set the `mailHost` property to the address of a valid mail server where the mail will be sent.

5.4.8 JAXR Test Setup

You will need a Registry to run the JAXR tests. The Java Web Services Developers Pack 1.3 contains an implementation of a UDDI version 2 business registry. You can find the Java Web Services Developers Pack at `http://java.sun.com`. Refer to the installation instructions that accompany the webservices software for additional information.

The JAXR test suite assumes you are using an internal registry (inside a firewall). If you want to use a public registry outside a firewall, you will need to make some updates to the `ts.jte` file in the `testExecuteAppClient` section to set up proxy information.

5.4.8.1 Example JAXR `ts.jte` Property Settings

```
http.proxyHost=myProxy.myCompany.com
http.proxyPort=8080
https.proxyHost=myProxy.myCompany.com
https.proxyPort=8080
command.testExecuteAppClient= \
com.sun.ts.lib.harness.ExecTSTestCmd DISPLAY=${ts.display} HOME="${user.home}" \
LD_LIBRARY_PATH=${javaee.home}/lib \
windir=${windir} \
SYSTEMROOT=${SYSTEMROOT} \
PATH="${javaee.home}/nativelib" \
TZ=${tz} \
${JAVA_HOME}/bin/java \
-Dorg.omg.CORBA.ORBInitialHost=${orb.host} \
-Djava.security.policy=${javaee.home}/lib/appclient/client.policy \
-Dcts.tmp=$harness.temp.directory \
-Dorg.omg.CORBA.ORBInitialPort=${orb.port} \
-Djava.security.auth.login.config=${javaee.home}/lib/appclient/appclientlogin.conf \
-Djava.protocol.handler.pkgs=javax.net.ssl \
-Dcom.sun.enterprise.home=${javaee.home} \
-Djavax.net.ssl.keyStore=${ts.home}/bin/certificates/clientcert.jks \
-Djavax.net.ssl.keyStorePassword=changeit \
-Dcom.sun.aas.installRoot=${javaee.home} \
-Dcom.sun.aas.imqLib=${javaee.home}/imq/lib \
-Djava.util.logging.manager=com.sun.enterprise.server.logging.ACCLogManager \
-Djavax.net.ssl.trustStore=${s1as.domain}/config/cacerts.jks \
-Djava.endorsed.dirs=${s1as.java.endorsed.dirs} \
-Dstartup.login=false \
-Dhttp.proxyHost=${http.proxyHost} \
-Dhttp.proxyPort=${http.proxyPort} \
-Dhttps.proxyHost=${https.proxyHost} \
-Dhttps.proxyPort=${https.proxyPort} \
-Ddeliverable.class=${deliverable.class} -classpath ${ts.run.classpath} \
com.sun.enterprise.appclient.Main $testExecuteArgs -configxml ${s1as.domain}/config/sun-acc.xml
```

The above settings cover the appclient portion of the testing. You will also need to update the proxy information for the server. For the RI, you will need to add this to the `jvm-options` in the `domain.xml` file. You can also use the CTS `config.vi` script to update `domain.xml` instead of directly editing it. To use the `config.vi` script, add the proxy to `s1as.jvm.options` in the `ts.jte` file:

```
s1as.jvm.options=-Dhttp.proxyHost=${http.proxyHost}:
-Dhttp.proxyPort=${http.proxyPort}:
-Dhttps.proxyHost=${https.proxyHost}:
-Dhttps.proxyPort=${https.proxyPort}
```

Note that this is only needed for JAXR when using an external registry. You should not add proxy info when running the rest of the CTS test suite.

5.4.8.2 JAXR `ts.jte` Properties

For JAXR testing, make sure the following properties have been set in the `<TS_HOME>/bin/ts.jte` file:

- `authenticationMethod` — Authentication method for the JAXR provider; UDDI_GET_AUTHTOKEN is the AuthToken protocol defined by [UDDI_API2]
- `registryURL` — Standard connection property for publishing. for the RegistryServer, the UDDI Registry that comes with the Java Web Services Developers Pack: registryURL = http://localhost:8080/RegistryServer/
- `queryManagerURL` — Standard connection property for querying. for the RegistryServer, the UDDI Registry that comes with the Java Web Services Developers Pack: queryManagerURL = http://localhost:8080/RegistryServer/
- `jaxrPassword` — Used for setting connection credentials this must be set up in the UDDI registry

- `jaxrUser` — Used for setting connection credentials this must be set up in the UDDI registry
- `jaxrUser2`, `jaxrPassword2` — Second account for Association tests

  For the `RegistryServer` you can use the predefined username and passwords. These would be set as follows:

  - `jaxrUser=testuser`
  - `jaxrPassword=testuser`
  - `jaxrUser2=jaxr-sqe`
  - `jaxrPassword2=jaxrsqe`

  If using digital certificates set to = "" instead.
- `jaxrSecurityCredentialType` — Tells CTS whether to use username/password or digital certificates for JAXR credentials:
  - `0` — Username/password
  - `1` — Digital certificates
- `jaxrAlias` — Identifies the entry in the keystore for this user.
- `jaxrAlias2` — Identifies the entry in the keystore for the second user
- `jaxrAliasPassword` — Identifies the password in the keystore for this user.
- `jaxrAlias2Password` — Identifies the password in the keystore for the second user
- `jaxrWebContext` — Context root for jaxr html tests pages
- `jaxrConnectionFactoryLookup` — Preferred way for a client to lookup a JAXR ConnectionFactory is to use JNDI; an alternate method is to use the `newInstance` static method on the `ConnectionFactory`
  - `0` = use JNDI lookup
  - `1` = use `newInstance` method.
- `jaxrJNDIResource` — JAXR `ConnectionFactoryReference` if JNDI lookup is being used; for example `java:comp/env/eis/JAXR`; if not using JNDI lookup set to `""`.
- `providerCapabilityLevel` — Provider must set this to the supported capability level `0` or `1`

5.4.9 JAX-RS Test Setup

This section explains how to set up the test environment to run the JAX-RS tests using the Java EE 6 Reference Implementation and/or a Vendor Implementation. This setup also includes steps for packaging/repackaging and publishing the packaged/repackaged WAR files as well.

▼ To Configure Your Environment to Run the JAX-RS Tests Against the Java EE 6 RI

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

1. **Set the `jaxrs_impl_lib` property to point to the Jersey RI.**

   The default setting for this property is `${javaee.home}/modules/jersey-gf-bundle.jar`.

2. **Set the `servlet_adaptor` property to point to the Servlet adaptor class for the JAX-RS implementation.**

   The default setting for this property is `com/sun/jersey/spi/container/servlet/ServletContainer.class`, the servlet adaptor supplied in Jersey.

3. **Set the `jaxrs_impl_name` property to the name of the JAX-RS RI.**

   A file bearing this name has been created under `$TS_HOME/bin/xml/impl/glassfish/jersey.xml` with packaging instructions.

   The default setting for this property is `jersey`.

▼ To Package WAR files for Deployment on the Java EE 6 RI

The Java EE 6 CTS test suite does not come with prebuilt test WAR files for deployment on Java EE 6 RI . The test suite includes a command to generate the test WAR files that will be deployed on the Java EE 6 RI. The WAR files are Jersey-specific, with Jersey's servlet class and Jersey's servlet defined in the `web.xml` deployment descriptor.

Generated Jersey-specific WAR files will be deposited in the *$TS_HOME*`/dist` directory and will include `jersey` as part of their name. For example:

```
$TS_HOME/dist/com/sun/ts/tests/jaxrs/ee/rs/getTest/jaxrs_rs_getTest_web.war.jersey
```

The names of the WAR files include the string `jersey` to make them unique, thereby minimizing the risk that the files will be overwritten by files with the same name or the original files accidentally modified.

To package the JAX-RS WAR files for deployment on the Java EE 6 RI, complete the following steps:

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

2. **Execute the `update.jaxrs.wars` Ant target.**

   This creates a `web.xml.jersey` file for each test based on the Java EE 6 Reference Implementation's servlet class name and packages the tests in WAR files that will be deployed on the Java EE 6 RI.

▼ To Configure Your Environment to Run the JAX-RS Tests Against a Vendor Implementation

Complete the following steps to configure your test environment to run the JAX-RS tests against your vendor implementation. Before you can run the tests, you need to repackage the WAR files that contain the JAX-RS tests and the VI-specific Servlet class that will be deployed on the vendor's Java EE 6–compliant application server.

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

1 **Set the `jaxrs_impl_lib` properties to point to all classes/JARs for the vendor's JAX-RS implementation and all dependent libraries.**

The default setting for these properties is `${javaee.home}/modules/jersey-gf-bundle.jar`.

2 **Set the `servlet_adaptor` property to point to the Servlet adaptor class for the vendor's JAX-RS implementation.**

The class must be located in the JAR file defined by the `jaxrs_impl_lib` property. By default, this property is set to `com/sun/jersey/spi/container/servlet/ServletContainer.class`, the servlet adaptor supplied in Jersey.

3 **Set the `jaxrs_impl_name` property to the name of the JAX-RS vendor implementation to be tested.**

The name of the property must be unique. A file bearing this name will be created under `$TS_HOME/bin/xml/impl/${impl.vi}/${jaxrs_impl_name}.xml` with packaging and/or deployment instructions.

The default setting for this property is `jersey`.

▼ To Repackage WAR files for Deployment on the Vendor Implementation

To run the JAX-RS tests against a vendor's implementation in a Java EE 6–compliant application server, the tests need to be repackaged to include the VI-specific servlet, and the VI-specific servlet must be defined in the deployment descriptor.

A vendor must create VI-specific Java EE 6 –compliant WAR files so the VI-specific Servlet class will be included instead of the Java EE 6 RI-specific Servlet class.

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

Java EE 6 CTS makes it easier for the vendor by including template WAR files that contain all of the necessary files except for the VI-specific servlet adaptor class. The Java EE 6 CTS provides a tool to help with the repackaging task.

Each test that has a JAX–RS resource class to publish comes with a template deployment descriptor file. For example, the file `$TS_HOME/src/com/sun/ts/tests/jaxrs/ee/rs/getTest/web.xml.template` contains the following elements:

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

In this example, the `<servlet-class>` element has a value of `servlet_adaptor`, which is a placeholder for the implementation-specific Servlet class. A Jersey-specific deployment descriptor also comes with the Java EE 6, and has the values for the `com.sun.jersey.spi.container.servlet.ServletContainer`:

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

The Java EE 6 CTS test suite provides a tool, `${ts.home}/bin/xml/impl/glassfish/jersey.xml`, for the Java EE 6 RI that you can use as a model to help you create your own VI-specific Web test application.

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

1 **Create a VI handler file.**

Create a VI-specific handler file `$TS_HOME/bin/xml/impl/${`*impl.vi*`}/${`*jaxrs_impl_name*`}.xml` if one does not already exist.

Ensure that the `jaxrs_impl_name` property is set in the `ts.jte`file and that its name is unique, to prevent another file with the same name from being overwritten.

2 **Set the `servlet_adaptor` property in the `ts.jte` file.**

This property will be used to set the value of the `<servlet-class>` element in the deployment descriptor.

3 **Create VI Ant tasks.**

Create a `update.jaxrs.wars` target in the VI handler file. Reference this `update.jaxrs.wars` target in the `jersey.xml` file.

This target will create a `web.xml.${`*jaxrs_impl_name*`}` for each test that has a deployment descriptor template. The `web.xml.${`*jaxrs_impl_name*`}` will contain the VI-specific Servlet class name. It will also create the test WAR files will be created under `$TS_HOME/dist` directory. For example:

```
ls $TS_HOME/dist/com/sun/ts/tests/jaxrs/ee/rs/getTest/ \
jaxrs_rs_getTest_web.war.jersey
jaxrs_rs_getTest_web.war.${jaxrs_impl_name}
```

4 **Change to the `$TS_HOME/bin` directory and execute the `update.jaxrs.wars` Ant target.**

This creates a `web.xml.${`*jaxrs_impl_name*`}` file for each test based on the VI's servlet class name and repackages the tests.

5.4.10 Deployment Test Setup

Ensure that the following properties in the `ts.jte` file have been set:

- `deployManagerJarFile.1` property to a JAR file that contains manifest entries according to the *Java Platform, Enterprise Edition Deploy API 1.0 Specification*
- `deployManageruri.1` property to a URI to connect to
- `deployManageruname.1` and `deployManagerpasswd.1` properties to the user name and password that are used when connecting to a deployment manager, if needed

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

5.4.11 Connector Test Setup

The Connector tests verify that a Java Platform, Enterprise Edition server correctly implements the Connector V1.6 specification. The Connector compatibility tests ensure that your Java Platform, Enterprise Edition server still supports the Connector V1.0 functionality.

5.4.11.1 Extension Libraries

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

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

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

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

The RAR files that are used withJava EE 6 CTS test suite differ from those that were used in earlier test suites. Java EE 6 CTS no longer bundles the same common classes into every RAR file. Duplicate common classes, such as `whitebox.jar`, have been removed from each RAR file and are now handled as an Installed Library.

This was done to address the following compatibility issues:

- Portable use of Installed Libraries for specifying a resource adapter's shared libraries

  See section EE.8.2.2 of the Java EE 6 platform specification and section 20.2.0.1 in the JCA 1.6 specification, which explicitly state that the resource adapter server may employ the library mechanisms in Java EE 6.
- Support application-based standalone connector accessibility

  Section 20.2.0.4 of the JCA 1.6 Specification uses the classloading requirements that are listed in section 20.3 in the specification.

5.4.11.2 Connector Resource Adapters and Classloading

Java EE 6 CTS has scenarios in which multiple standalone RAR files that use the same shared library (for example, `whitebox.jar`) are referenced from an application component.

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

5.4.11.3 Use Case Problem Scenario

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

- RAR1's classloader has access to RAR1's classes and its copy of `whitebox.jar`. (RAR1's classloader contains RAR1's classes and `whitebox.jar`)
- RAR2's classloader has access to RAR2's classes and its copy of `whitebox.jar`. (RAR2's classloader contains RAR2's classes and `whitebox.jar`)

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

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

The portable way of solving the issues raised by this use case problem scenario is to use installed libraries, as described in section EE.8.2.2 in the Java EE 6 platform specification. If both RAR files (RAR1 and RAR2) reference `whitebox.jar` as an installed library and the application server can use a single classloader to load this common dependency, there will be no type-related issues.

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

5.4.11.4 Required Porting Package

The Java EE 6 CTS test suite treats the `whitebox.jar` dependency as an Installed Library dependency instead of bundling the dependency (or dependencies) with every RAR file. Each RAR file now contains a reference to the `whitebox.jar` file through its Manifest files Extension-List attribute.

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

See section EE.8.2.2 in the Java EE 6 platform specification for information about the reference implementation's support for Installed libraries. However, note that this section does *not* recommend a mechanism that a deployer can use to provide Installed Libraries in a portable manner.

5.4.11.5 Creating Security Mappings for the Connector RAR Files

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

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

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

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

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

5.4.11.6 Creating Required Server-Side JVM Options

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

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

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

5.4.12 XA Test Setup

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

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

The Java EE 6 CTS whitebox adapter is a resource adapter that provides the JDBC API as its client API. This model has two valuable benefits:

- Your database serves as the Enterprise Information System, as defined in the Connector architecture.
- The resource adapter will work with the same JDBC 3.0-compliant driver that you use with the rest of the test suite.

▼ To set up your environment to run the XA tests

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

- `connector_connectionURL` — The URL for your JDBC driver; should be the same as the one used elsewhere in the test suite.
- `xa.properties` — Properties required by the XA driver; the managed connection factory will set these properties via reflection on the class specified by the `xa.xadatasource.class` property. The `xa.properties` value is the set of properties that will be set on your XA JDBC driver (the class denoted by the `xa.xadatasource.class` property). See Section 9.4.1 of the JDBC 3.0 Specification for more details.

Note – When specifying `xa.properties`, be sure to escape the appropriate characters. The `xa.properties` value needs to be treated as a single property even though it is made up of many subproperties. The properties must be separated by colons (`:`), and the names and values separated by equal signs (=). Any values that contain colons or equal signs that do not need to be interpreted to must inside single quotes. All values in single quotes are treated as a string literals.

- `xa.xadatasource.class` — The implementation of the `XADataSource` interface; this class is your XA JDBC driver. Note that this class and any dependent classes must be accessible by the CTS libraries in the application server's extension directory.
- `connector_connectionURL` — The URL for your JDBC driver; should be the same as the one used elsewhere in the test suite.

You need to modify some or all of these properties to plug your own JDBC driver into the CTS whitebox resource adapter.

2 Make sure that the database to which you point has been initialized.

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

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

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

- The embedded RAR files are located in the `<TS_HOME>/src/com/sun/ts/tests/xa/ee/tsr` directory.
- For the JDBC resource adapter, deploy the following RAR files:

```
JDBCwhitebox-tx=java:comp/env/eis/JDBCwhitebox-tx
JDBCwhitebox-notx=java:comp/env/eis/JDBCwhitebox-notx
JDBCwhitebox-xa=java:comp/env/eis/JDBCwhitebox-xa
JDBCwhitebox-tx-param=java:comp/env/eis/JDBCwhitebox-tx-param
JDBCwhitebox-notx-param=java:comp/env/eis/JDBCwhitebox-notx-param
JDBCwhitebox-xa-param=java:comp/env/eis/JDBCwhitebox-xa-param
```

Deploy the RAR files to the appropriate JNDI name, as indicated. For example, you would deploy the `JDBCwhitebox-tx.rar` file to `java:comp/env/eis/JDBCwhitebox-tx`.

- The EIS RAR files are located in the following directory:
  `<TS_HOME>/src/com/sun/ts/tests/common/connector/whitebox`
- The JDBC RAR files are located in the following directory:
  `<TS_HOME>/src/com/sun/ts/tests/common/connector/JDBCwhitebox/whitebox`

RAR files in the `<TS_HOME>/src/com/sun/ts/tests/common/connector` directory must be built and must exist before any dependent tests can pass. Deployment can either be done ahead of time or at runtime, as long as connection pools and resources are established prior to test execution.

5.4.13 EJB 3.0 Test Setup

This section explains special configuration needs to be completed before running the EJB 3.0 DataSource and Stateful Timeout tests.

The EJB 3.0 DataSource tests do not test XA capability and XA support in a database product is typically not required for these tests. However, some Java EE products could be implemented in such a way that XA must be supported by the database. For example, when processing the @DataSourceDefinition annotation or `<data-source>` descriptor elements in tests, a Java EE product infers the datasource type from the interface implemented by the driver class. When the driver class implements multiple interfaces, such as `javax.sql.DataSource`, `javax.sql.ConnectionPoolDataSource`, or `javax.sql.XADataSource`, the vendor must choose which datasource type to use. If `javax.sql.XADataSource` is chosen, the target datasource system must be configured to support XA. Consult the documentation for your database system and JDBC driver for information that explains how to enable XA support.

▼ To Configure the Test Environment to Run the EJB 3.0 DataSource Tests

The EJB 3.0 DataSource tests under the following `tests/ejb30` directories require you to run the Ant task in step 2.

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

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

1 **Set any vendor-specific or less common datasource properties with the `jdbc.datasource.props` property in the `ts.jte` file.**

The value of the property is a comma-separated array of name-value pairs, in which each property pair uses a "`name=value`" format, including the surrounding double quotes.

The value of the property must not contain any extra spaces.

For example:

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

2 **Run the `configure.datasource.tests` Ant target to rebuild the EJB 3.0 DataSource Definition tests using the new database settings specified in the `ts.jte` file.**

This step must be completed for Java EE 6 and Java EE 6 Web Profile testing.

▼ To Configure the Test Environment to Run the EJB 3.0 Stateful Timeout Tests

The EJB 3.0 Stateful Timeout Tests in the following test directories require special setup:

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

1 **Set the `javatest.timeout.factor` property in the `ts.jte` file to a value such that the JavaTest harness does not time out before the test completes.**

A value of 2.0 or greater should be sufficient.

2 **Set the `test.ejb.stateful.timeout.wait.seconds`property, which specifies the minimum amount of time, in seconds, that the test client waits before verifying the status of the target stateful bean, to a value that is appropriate for your server.**

The value of this property must be an integer number. The default value is 480 seconds. This value can be set to a smaller number (for example, 240 seconds) to speed up testing, depending on the stateful timeout implementation strategy in the target server.

5.4.14 EJB Timer Test Setup

Set the following properties in the `ts.jte` file to configure the EJB timer tests:

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

- The `ejb_timeout` property sets the duration of single-event and interval timers. The default setting and recommended minimum value is `30000` milliseconds.
- The `ejb_wait` property sets the period for the test client to wait for results from the `ejbTimeout()` method. The default setting and recommended minimum value is `60000` milliseconds.

Java EE 6 CTS does not have a property that you can set to configure the date for date timers.

The timer tests use the specific `jndi-name` jdbc/DBTimer for the datasource used for container-managed persistence to support the Java Platform, Enterprise Edition RI's use of an XA datasource in its timer implementation. For example:

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

The test directories that use this datasource are:

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

When testing against the Java Platform, Enterprise Edition RI, you must first start JavaDB and initialize it in addition to any other database you may be using, as explained in "Configuring the Java Platform, Enterprise Edition RI as the VI" on page 53.

5.4.15 Entity Bean Container-Managed Persistence Test Setup for EJB V1.1

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

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

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

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

These instance variable names correspond to the following database schema:

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

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

The Java Platform, Enterprise Edition RI provides the container-managed persistence implementation-specific features as part of its runtime XML file. Your Java Platform, Enterprise Edition platform implementation needs to map the container-managed fields to the appropriate backend datastore. The manner in which you do this is implementation-specific. The `DeploymentInfo` class provides all of the runtime XML information as an object that is passed to the `TSDeploymentInterface2` implementation.

For a list of SQL statements used in CMP 1.1 finders, refer to "G.2 SQL Statements for CMP 1.1 Finders" on page 201.

5.4.16 Java Persistence API Test Setup

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

Within a Java Platform, Enterprise Edition environment, support for both container-managed and application-managed entity managers is required. Application-managed entity managers can be JTA or resource-local. Refer to Chapter 5 of the Java Persistence API Specification for additional information regarding entity managers.

▼ To Configure the Test Environment to Run the JPA Pluggability Tests

The JPA Pluggability tests under the `src/com/sun/ts/tests/pluggability` directory ensure that a third-party persistence provider is pluggable, in nature. These tests require additional setup before they can be run with a Vendor's Implementation.

1 **If you are using the Reference Implementation, execute the `enable.ts.persistence.provider` Ant task, which replaces EclipseLink, the default persistence provider, with the TopLink persistence provider.**

One of the things that this task does is to add a JVM option to the Reference Implementation to specify where the log file is written.

Note – If you are *not* using the Reference Implementation, you must copy the TopLink persistence provider JAR file and set the JVM option that specifies the location to which the log file will be written by hand.

---

2 **Set the `log.file.location` property in the `ts.jte` file to specify the location to which the `JPALog.txt` log file will be written.**

Java EE 6 CTS uses TopLink to log persistence API calls. The logs are collected in the `JPALog.txt` file.

3 **If you are using the Reference Implementation, after completing the test run, execute the `disable.ts.persistence.provider` Ant task.**

This task reverts the persistence provider to the default setting (the default persistence provider, EclipseLink).

Note – If you are *not* using the Reference Implementation, you must revert to EclipseLink, the default persistence provider, and unset the JVM option that specifies the location to which the log file will be written by hand.

5.4.16.1 Persistence Test Vehicles

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

- `stateless3` — Bean-managed stateless session bean using JNDI to lookup a JTA `EntityManager`; uses `UserTransaction` methods for transaction demarcation
- `stateful3` — Container-managed stateful session bean using `@PersistenceContext` annotation to inject JTA `EntityManager;` uses container-managed transaction demarcation with a transaction attribute (required)
- `appmanaged` — Container-managed stateful session bean using `@PersistenceUnit` annotation to inject an `EntityManagerFactory`; the `EntityManagerFactory` API is used to create an Application-Managed JTA `EntityManager`, and uses the container to demarcate transactions
- `appmanagedNoTx` — Container-managed stateful session bean using `@PersistenceUnit` annotation to inject an `EntityManagerFactory`; the `EntityManagerFactory` API is used to create an Application-Managed Resource Local `EntityManager`, and uses the `EntityTransaction` APIs to control transactions
- `pmservlet` - Servlet that uses the @PersistenceContext annotation at the class level and then uses jndi lookup to obtain the `EntityManager`; alternative to declaring the persistence context dependency via a `persistence-context-ref` in `web.xml`, as discussed below; tested in the `com/sun/ts/tests/ejb30/persistence/ee` test directory, and uses `UserTransaction` methods for transaction demarcation
- `puservlet` — Servlet that injects an `EntityManagerFactory` using the `@PersistenceUnit` annotation to create a to Resource Local `EntityManager`, and uses `EntityTransaction` APIs for transaction demarcation

Note – For vehicles using a RESOURCE_LOCAL transaction type, be sure to configure a non-transactional resource with the logical name jdbc/DB_no_tx. Refer to Appendix B, "ts.jte Modifiable Properties," for information about the jdbc.db property in ts.jte.

5.4.16.2 GeneratedValue Annotation

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

The GeneratedValue annotation is exercised in the test directory com/sun/ts/tests/ejb30/persistence/types/auto. The table provided for this test to run with Java Platform, Enterprise Edition RI is listed in Example 5–1.

EXAMPLE 5–1 GeneratedValue Annotation Test Table

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

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

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

5.4.17 JMS Test Setup

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

```
jms_timeout=5000
```

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

Note – The client-specified values for `JMSDeliveryMode`, `JMSExpiration`, and `JMSPriority` must not be overridden when running the CTS JMS tests.

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

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

- **Factories**:

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

- **Queues**:

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

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

- **Topics**:

```
MY_TOPIC
MY_TOPIC1
TOPIC_BMT
ejb_ee_bb_localaccess_mdbtaccesstest_MDB_TOPIC
ejb_ee_deploy_mdb_ejblink_casesensT_TestBean
ejb_ee_deploy_mdb_ejblink_scopeT_TestBean
ejb_ee_deploy_mdb_ejblink_singleT_TestBean
```

```
ejb_ee_deploy_mdb_ejblink_singleT_TestBeanBMT
ejb_ee_deploy_mdb_ejbref_casesensT_TestBean
ejb_ee_deploy_mdb_ejbref_scopeT_Cyrano
ejb_ee_deploy_mdb_ejbref_scopeT_Romeo
ejb_ee_deploy_mdb_ejbref_scopeT_Tristan
ejb_ee_deploy_mdb_ejbref_singleT_TestBean
ejb_ee_deploy_mdb_ejbref_singleT_TestBeanBMT
ejb_ee_deploy_mdb_enventry_casesensT_CaseBean
ejb_ee_deploy_mdb_enventry_casesensT_CaseBeanBMT
ejb_ee_deploy_mdb_enventry_scopeT_Bean1_MultiJar
ejb_ee_deploy_mdb_enventry_scopeT_Bean1_SameJar
ejb_ee_deploy_mdb_enventry_scopeT_Bean2_MultiJar
ejb_ee_deploy_mdb_enventry_scopeT_Bean2_SameJar
ejb_ee_deploy_mdb_enventry_singleT_AllBean
ejb_ee_deploy_mdb_enventry_singleT_AllBeanBMT
ejb_ee_deploy_mdb_enventry_singleT_BooleanBean
ejb_ee_deploy_mdb_enventry_singleT_ByteBean
ejb_ee_deploy_mdb_enventry_singleT_DoubleBean
ejb_ee_deploy_mdb_enventry_singleT_FloatBean
ejb_ee_deploy_mdb_enventry_singleT_IntegerBean
ejb_ee_deploy_mdb_enventry_singleT_LongBean
ejb_ee_deploy_mdb_enventry_singleT_ShortBean
ejb_ee_deploy_mdb_enventry_singleT_StringBean
ejb_ee_deploy_mdb_resref_singleT_TestBean
jms_ee_mdb_mdb_exceptT_MDB_DURABLETXNS_CMT
jms_ee_mdb_mdb_exceptT_MDB_DURABLE_BMT
jms_ee_mdb_mdb_exceptT_MDB_DURABLE_CMT
jms_ee_mdb_mdb_msgHdrT_MDB_TOPIC
jms_ee_mdb_mdb_msgPropsT_MDB_TOPIC
jms_ee_mdb_mdb_msgTypesT1_MDB_TOPIC
jms_ee_mdb_mdb_msgTypesT2_MDB_TOPIC
jms_ee_mdb_mdb_msgTypesT3_MDB_TOPIC
jms_ee_mdb_mdb_rec_MDB_TOPIC
jms_ee_mdb_mdb_sndToTopic_MDB_TOPIC
jms_ee_mdb_mdb_sndToTopic_MDB_TOPIC_REPLY
jms_ee_mdb_xa_MDB_DURABLE_BMT
jms_ee_mdb_xa_MDB_DURABLE_CMT
```

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

5.4.18 Transaction Interoperability Testing

Using two Java Platform, Enterprise Edition server implementations, you can test up to four transaction interoperability configurations. However, note that you only need to test and pass configurations that your Java Platform, Enterprise Edition server supports. Table 5–1 shows these configurations.

TABLE 5–1 Transaction Interoperability Testing Configurations

Configuration	Transaction Interoperability Setting for a Java Platform, Enterprise Edition Vendor Implementation	Transaction Interoperability Setting for the Java Platform, Enterprise Edition RI
1	ON	OFF
2	ON	ON
3	OFF	OFF
4	OFF	ON

Modify the interoperability settings for transaction interoperability according to what you need to test:

- If your implementation *supports* transaction interoperability, you must test configurations #1 and #2.
- If your implementation *does not support* transaction interoperability, you must test configurations #3 and #4.

The `ts.jte` file has the following transaction interoperability properties:

```
EJBServer1TxInteropEnabled=[false | true]
EJBServer2TxInteropEnabled=[false | true]
```

To run the required test configurations described in Table 5–1, use the following commands to change the Java Platform, Enterprise Edition SDK settings as necessary.

- To set the Java Platform, Enterprise Edition RI Transaction Interoperability setting to `False`:

  ```
  cd $TS_HOME/bin
  <TS_HOME>/tools/ant/bin/ant disable.ri.tx.interop
  ```
- To set the Java Platform, Enterprise Edition RI Transaction Interoperability setting to `True`:

  ```
  <TS_HOME>/tools/ant/bin/ant enable.ri.tx.interop
  ```

The default Java Platform, Enterprise Edition RI Transaction Interoperability setting is `True`.

5.4.19 JASPIC Test Setup

Java Authentication Service Provider Interface for Containers (JASPIC) 1.0 tests are security tests. The JASPIC Servlet profile is the only required profile for Java EE 6 CTS. There are other optional profile tests, such as SOAP, but you are not required to run these for certification.

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

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

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

Complete the following steps before you run the JASPIC tests:

1. Configure the JASPIC-required properties in the `ts.jte` file:
   a. Set the `provider.configuration.file` property to the location of your implementation's instance `lib` directory, where it can be loaded when your implementation runtime is started.

      This file typically coexists with the `tssv.jar` file and the `provider-configuration.dtd` file.

   b. Set the `vendor.authconfig.factory` property to specify your `AuthConfigFactory` class.

      This property setting will be used by the JASPIC tests to register the test suite's provider in your `AuthConfigFactory`.

   c. Set the `logical.hostname.servlet` property to the logical host that will process Servlet requests.

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

   d. Set the `servlet.is.jsr115.compatible` property based on whether or not you are running the Servlet profile in a JSR 115–compatible container.
2. Ensure that the `config.vi` Ant task has been run before running the `enable.jaspic` Ant task.

   These Ant tasks perform the following JASPIC–required steps:

   - Set up users and passwords for your implementation.

     See Step 8b in "Configuring Your Application Server as the VI" on page 55"Configuring Your Application Server as the VI" on page 55 for more information.

   - Install the client-side certificate in the `trustStore` in your implementation.

     See Step 16 in "Configuring Your Application Server as the VI" on page 55"Configuring Your Application Server as the VI" on page 55 for more information.

- Append the file `<TS_HOME>/bin/server_policy.append` to the Java policy file(s) on your implementation.

  See Step 17 in "Configuring Your Application Server as the VI" on page 55"Configuring Your Application Server as the VI" on page 55 for more information.
- Appends the file `<TS_HOME>/bin/client_policy.append` to the application client's Java policy file, which is referenced in the `TestExecuteAppClient` section of the `ts.jte` file.

  See Step 18 in "Configuring Your Application Server as the VI" on page 55"Configuring Your Application Server as the VI" on page 55 for more information.
- Copies the `TS_HOME/lib/tssv.jar` file to your implementation instance library directory.

  The `tssv.jar` file includes the class files necessary to load `TSAuthConfigFactory` and related classes.
- Copies the TSSV configuration files (`ProviderConfiguration.xml`, `configuration.dtd`) to your implementation instance library directory.

  The `provider-configuration.dtd` file is a DTD file that resides in the same directory as the `ProviderConfiguration.xml` file and describes the `ProviderConfiguration.xml` file. This file should *not* be edited.
- Copies `<TS_HOME>/bin/ts.java.security` to `<JAVAEE_HOME>/domains/domain1/config/ts.java.security`.
- Sets the the following JVM options:
  - `-Djava.security.properties=<JAVAEE_HOME>/domains/domain1/config/ts.java.security`
  - `-Dlog.file.location=${log.file.location}`
  - `-Dprovider.configuration.file=${provider.configuration.file}`

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

   Note – It may be necessary to restart your implementation after completing this step.

   ---

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

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

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

5.4.20 CSIv2 Test Setup

Common Secure Interoperability Version 2 (CSIv2) is security-related interoperability testing. You must complete all of the setup instructions in "Configuring a Java Platform, Enterprise Edition RI and VI for Interop/Rebuildable Tests" on page 60 and the steps provided in this section before you run the CSIv2 tests.

This following sections describe how to set up two Java Platform, Enterprise Edition servers, one running the vendor's Java Platform, Enterprise Edition server and the other running the Java Platform, Enterprise Edition RI. Be sure to complete the steps in "5.3 Configuring a Java Platform, Enterprise Edition Server" on page 51 before proceeding with the instructions below.

▼ Configuring the Vendor's Java Platform, Enterprise Edition Server

- **Make sure the required IORs, based on the values of the fields that are described in "Generating IORs Based on Runtime XML Information" on page 91, Appendix E, "Interoperable Object Reference Definitions," provide additional information about IOR definitions.**

▼ Configuring the Java Platform, Enterprise Edition RI Server

1. **Run the CSIv2 Ant target.**

   ```
   <TS_HOME>/tools/ant/bin/ant enable.csiv2
   ```

2. **Run the CSIv2 tests.**

   See Chapter 7, "Executing Tests," for instructions on executing tests. After the test run concludes, you need to analyze the results. See Appendix C, "Analyzing CSIv2 Test Logs," for information about analyzing the test logs.

3. **Disable CSIv2.**

   ```
   <TS_HOME>/tools/ant/bin/ant disable.csiv2
   ```

▼ Generating IORs Based on Runtime XML Information

The DeploymentInfo class contains public accessor methods that correspond to XML elements within the <ior-security-config> element in the Sun EJB jar runtime XML that is packaged with the test beans. Java Platform, Enterprise Edition vendor implementations are required to generate IORs that are based on the values pulled from this XML document as described in this section. Failure to do so will result in test failures.

The fields are divided into three categories:

- Fields that deal with client authentication at the client authentication layer of the CSIv2 protocol:
  - `asAuthMethod`

    - `asRequired`
    - `asRealmName`
- Fields that deal with the secure transport layer (SSL):
    - `transportIntegrity`
    - `transportConfidentiality`
    - `EstablishTrustInTarget`
    - `EstablishTrustInClient`
- Fields that deal with caller propagation:
    - `sasCallerPropagation`

The values of these fields must be used to construct the CompoundSecMec structure within an IOR. The construction of the IORs is briefly described in this class. Appendix E, "Interoperable Object Reference Definitions," lists of all the IORS that are expected to be generated. For more detailed information about IORS, refer to the CSIv2 specification.

Not all possible combinations of fields are used by CSIv2 tests. For example, the following fields are used to indicate that client authentication is required at the client authentication layer of the CSIv2 protocol:

```
asAuthMethod = "username_password"
asRealmName  = "default"
asRequired   = "true"
```

The following fields are used to indicate that client authentication is not required at the client authentication layer of the CSIv2 protocol:

```
asAuthMethod = "username_passsword"
asRealmName  = "default"
asRequired   = "false"
```

- **The Java example listed below, demonstrates how users can extract the required information from the DeploymentrInfo object:**

```
DeploymentInfo info;
 List ejbs = info.getEnterpriseBeans().getEjb();
foreach ejb in ejbs {
   ejb.getIorSecurityConfig().getAsContext()
.getAuthMethod().getContent();
   ejb.getIorSecurityConfig().getAsContext()
.getRequired().getContent();
  ejb.getIorSecurityConfig().getAsContext()
.getRealm().getContent();
   ejb.getIorSecurityConfig().getTransportConfig()
.getIntegrity().getContent();
   ejb.getIorSecurityConfig().getTransportConfig()
.getConfidentiality().getContent();
   ejb.getIorSecurityConfig().getTransportConfig()
.getEstablishTrustInClient().getContent();
   ejb.getIorSecurityConfig().getTransportConfig()
.getEstablishTrustInTarget().getContent();
} // end loop
```

5.4.20.1 Mapping Roles

The rolemapping element defines role-to-principal, role-to-group, and/or role-to-user-to-group mappings.

- A role is an logical grouping of users that is defined by an application component provider or assembler.
- A group is a set of users, classified by common traits, defined in the Java Platform, Enterprise Edition Application Server.

  Note that a Java Platform, Enterprise Edition group is designated for the entire Java Platform, Enterprise Edition server, whereas a role is associated with a specific application in a Java Platform, Enterprise Edition server only.
- A principal is an individual (or application program) identity that has been defined in the Java Platform, Enterprise Edition Application Server. Principals can be associated with a group.

The security-role-mappings are defined in the following files:

- `applicationName.ear.sun-application.xml`
- `applicationName.jar.sun-ejb-jar.xml`
- `applicationName.war.sun-web.xml`

However, the definitions in the file `applicationName.ear.sun-application.xml` take precedence over the definitions in the other files.

5.4.20.2 Role Mapping Examples

The examples that follow show how `role-name`, `principal-name`, and `group-names` are used for `security-role-mapping`.

EXAMPLE 5–2 `role-name` Administrator

```
<security-role-mapping>
    <role-name>Administrator</role-name>
    <principal-name>javaee</principal-name>
    <principal-name>javajoe</principal-name>
</security-role-mapping>
```

EXAMPLE 5–3 `role-name` Manager

The following example shows how `role-name` Manager is mapped to `principal-name` `javajoe` and `group-name` `MGR`.

```
<security-role-mapping>
    <role-name>Manager</role-name>
    <principal-name>javajoe</principal-name>
    <group-name>MGR</group-name>
</security-role-mapping>
```

5.4.20.3 Sample `ior-security-config` Elements

```
<ior-security-config>
  <transport-config>
    <integrity>supported</integrity>
    <confidentiality>supported</confidentiality>
    <establish-trust-in-target>supported</establish-trust-in-target>
    <establish-trust-in-client>supported</establish-trust-in-client>
  </transport-config>
  <as-context>
    <auth-method>username_password</auth-method>
    <realm>default</realm>
    <required>false</required>
  </as-context>
  <sas-context>
    <caller-propagation>supported</caller-propagation>
  </sas-context>
</ior-security-config>
```

`ior-security-config`, which describes the security configuration information for the IOR, consists of three components:

- "`transport-config`" on page 94
- "`as-context`" on page 96
- "`sas-context`" on page 97

transport-config

`transport-config` contains the following sub-elements:

- `integrity`
- `confidentiality`
- `establish-trust-in-client`
- `establish-trust-in-target`

integrity Field

The `integrity` field is used to indicate the integrity requirements that a target places on the client at the SSL level.

Valid values are:

- `none` — Indicates that the target does not support integrity at the SSL level
- `required` — Indicates that the target supports, but does not require, integrity at the SSL level
- `supported` — Indicates that the target requires the client to establish a secure connection with integrity at the SSL level

confidentiality Field

The `confidentiality` field is used to indicate the confidentiality requirements that a target places on the client at the SSL level.

Valid values are:

- `none` — Indicates that the target does not support confidentiality at the SSL level
- `required` — Indicates that the target requires the client to establish a secure connection with confidentiality at the SSL level
- `supported` — Indicates that the target supports, but does not require, confidentiality at the SSL level

An IOR must be generated as shown below. Although confidentiality is used as an example, the principles of IOR generation apply to all other fields related to security requirements at the SSL level, including `establish-trust-in-client`, `establish-trust-in-target`, and `integrity`.

- If the value for a field is `none`:
  - The bit that corresponds to the field in `transport_mech.target_requires` must be set to `0`.
  - The bit that corresponds to the field in `transport_mech.target_supports` must be set to `0`.
- If the value for a field is `supported`:
  - The bit that corresponds to the field in `transport_mech.target_supports` must be set to `1`.
  - The bit that corresponds to the field in `transport_mech.target_requires` must be set to `0`.

  For example, if the value of the confidentiality field is true, the following setting is necessary:

  ```
  transport_mech.target_supports = {Confidentiality}
  ```
- If the value for a field is `required`:
  - The bit that corresponds to the field in `transport_mech.target_requires` must be set to `1`.
  - The bit that corresponds to the field in `transport_mech.target_supports` must also be set to `1`.
  - The bit that corresponds to the field must also be set in `CompoundSecMec.target_requires`.

  For example, if the value of confidentiality is `required`, the following settings are necessary:

  ```
  transport_mech.target_requires={Confidentiality}
  transport_mech.target_supports={Confidentiality}
  CompoundSecMec.target_requires={Confidentiality}
  ```

establish-trust-in-target Field

The `establish-trust-in-target` field is used to indicate whether a target can authenticate itself to a client at the SSL level.

Valid values are:

- `none` — Indicates that the target cannot authenticate itself to the client
- `supported` — Indicates that the target can authenticate itself to a client

establish-trust-in-client Field

The `establish-trust-in-client` field is used to indicate the authentication requirements that a target places on the client at the SSL level.

Valid values are:

- `none` — Indicates that the target does not support client authentication at the SSL level
- `required` — Indicates the client must authenticate itself to the target at the SSL level
- `supported` — Indicates that the target supports, but does not require, client authentication at the SSL level

as-context

`as-context` (CSIv2 authentication service) describes the authentication mechanism that will be used to authenticate the client. If specified, it will be the username-password mechanism.

`as-context` contains the following sub-elements:

- `auth-method`
- `realm`
- `required`

auth-method Field

The `auth-method` field indicates the authentication mechanism that may be used to authenticate the client to the target at the client authentication layer.

Valid values are:

- `none` — Indicates that the target does not support client authentication at the client authentication layer. The IOR must be generated as follows:

  ```
  as_context_mech.target_supports = {}
  ```

  If the value is `none`, the `realm` and `required` field values are irrelevant.
- `username_password` — Indicates that the authentication mechanism is the `GSSUP` mechanism. This value is relevant and should only be used when `asRequired` is `true`. When set to `true`, the IOR must be generated as described in the `required` field summary.

realm Field

The `realm` field contains the name of the realm in which the user is to be authenticated.

Valid values are:

- `none`
- `default`

This field is relevant and should only be used when the `required` field is set to `true`, in which case the IOR must be generated as described in the `required` field summary.

required Field

The `required` field specifies whether or not a client is required to authenticate at the client authentication layer.

Valid values are:

- `true` — Indicates that the client is required to authenticate at the client authentication layer.

  If the value is `true`, an IOR must be generated as follows:

  - The `as_context_mech` must contain a client authentication mechanism derived from the value of the `auth-method` field. If the value of the `auth-method` field is `username_password`, the client authentication mechanism must be set to `GSSUP_OID`; for example:

    ```
    as_context_mech.client_authentication_mech = GSSUP_OID
    ```
  - The target name must match the value of the `realm` field:

    ```
    as_context_mech.target_name = {realm}
    ```
- The `establish-trust-in-client` bit must be set in the following fields:
  - `as_context_mech.target_supports={EstablishTrustInClient}`
  - `as_context_mech.target_requires={EstablishTrustInClient}`
  - `CompoundSecMec.target_requires={EstablishTrustInClient}`
- `false` — Indicates that client authentication at the client authentication layer is not required.

  The value of the `required` field can be `false`. However, in the CSIv2 tests, whenever the required field is `false`, the `auth-method` field must always be set to `none`. In this case, the IOR must be generated as described in the `auth-method` field summary.

sas-context

`sas-context` describes caller propagation. The `caller-propagation` field indicates whether the target will accept propagated caller identities.

Valid values are:

- `none`
- `supported`

The IOR must be generated as follows:

- If the value is none:
  - The bit that corresponds to the field in `sas_context_mech.target_supports` must be set to zero, as shown:

    ```
    sas_context_mech.target_supports={}
    ```
  - The value in the field `sas_context_mech.supported_naming_mechanisms` must be set to zero, as shown:

    ```
    supported_naming_mechanisms={}
    ```
  - The bit that corresponds to `ITTPrincipalName`, ITTDistinguishedName, `ITTX509CertChain`, and `ITTAnonymous` in the `sas_context_mech.supported_identity_types` field must be set to zero.
- If the value is `supported`:
  - The bit that corresponds to the field in `sas_context_mech.target_supports` must be set as follows:

    ```
    sas_context_mech.target_supports={IdentityAssertion}
    ```
  - The `sas_context_mech.supported_naming_mechanisms` field must contain at least `GSSUPMechOID`, as follows:

    ```
    supported_naming_mechanisms={GSSUPMechOID}
    ```
  - The `ITTPrincipalName` bit must be set in the `sas_context_mech.supported_identity_types`, as shown:

    ```
    sas_context_mech.supported_identity_types= \
    {ITTPrincipalName, ITTDistinguishedName, \
    ITTX509CertChain, ITTAnonymous}
    ```

  For the Java Platform, Enterprise Edition Appplication Server, the `ior-security-config` is defined in the `applicationName.jar.sun-ejb-jar.xml` file.

5.4.21 JACC Test Setup

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

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

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

- `A=javax.security.jacc.PolicyConfigurationFactory.provider`
- `B=javax.security.jacc.policy.provider`
- `C=vendor.javax.security.jacc.PolicyConfigurationFactory.provider`
- `D=vendor.javax.security.jacc.policy.provider`

▼ To Configure the JACC Provider for the Java Platform, Enterprise Edition RI

● **Execute the JACC Ant target:**

```
<TS_HOME>/tools/ant/bin/ant enable.jacc
```

This command does the following:

- Switches the system properties
- Adds `tsprovider.jar` to Java Platform, Enterprise Edition appserver's classpath.
- Adds `log.file.location` system property to the Java Platform, Enterprise Edition Appserver's system properties. This is used for generating log files, which is used for verifying JSR 115 contracts.

Note – When running JACC tests against the Java Platform, Enterprise Edition RI, if you need to restart the RI, be sure to first remove all JACC log files (`jacc_log.*`) from the `JAVAEE_HOME/domains/domain1/logs` directory.

Also note that if you are running CTS JSR 115 tests and you need to restart your appserver, be sure to remove the generated `jacc_log.txt` file before running the JSR 115 tests again.

5.4.22 JAX-WS Test Setup

The following properties, which are used by the Endpoint API tests in the `jaxws/api/javax_xml_ws/Endpoint` directory, must be set in the `ts.jte` file:

- The `http.server.supports.endpoint.publish` property indicates whether or not your HTTP server supports endpoint publishing.
- The `http.server.supports.endpoint.publish.2` property indicates whether or not the Java EE 6 RI HTTP server supports endpoint publishing.

If your server supports endpoint publishing (supports the use of `javax.xml.ws.Endpoint.publish()` methods), set the property value to “true”. In an *unmanaged* environment, such as that in standalone mode, this call is usually allowed. When the property is set to “true”, the test will check that the endpoint publish methods publish the endpoints without throwing an exception. When this result occurs with this scenario, the result is considered a PASS.

If your server does *not* support endpoint publishing (does not support the use of `javax.xml.ws.Endpoint.publish()` methods), set the property value to “false”. In a *managed* environment, such as the Java EE 6 Reference Implementation, this is usually the case and this call would not be allowed. When the property is set to “false”, the tests will check that these endpoint publish methods do not publish the endpoints and an exception will be thrown. When this result occurs with this scenario, the result is considered a PASS.

The `http.server.supports.endpoint.publish.2` property settings are used to specify whether or not endpoint publishing is supported by the Java EE 6 RI HTTP server. The same settings and caveats as the `http.server.supports.endpoint.publish` property apply to this property, based on whether or not the Java EE 6 RI supports endpoint publishing.

5.4.23 WSDL: Webservice Test and Runtime Notes

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

These WSDL tags are also used when performing file URL publishing, as required by JSR109. JSR109 states that http URL and file URL publishing must be supported on a Java Platform, Enterprise Edition platform. In addition, the `<wsdl-override>` is used as a mechanism for satisfying the partial WSDL requirement in the JSR109 specification. This mechanism enables the specification of the location of the final full published WSDL of a deployed webservice endpoint within the client EAR when only a partial WSDL is packaged, which enables client access to the full WSDL and correct SOAP address to communicate with the webservice.

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

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

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

5.4.23.1 WSDL `ts.jte` Properties

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

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

```
wsdlRepository1=$TS_HOME/tmp/wsdlRepository1
wsdlRepository2=$TS_HOME/tmp/wsdlRepository2
```

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

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

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

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

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

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

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

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

Note – In the case of interop webservices tests, the CTS harness substitutes the `webServerHost.2:webServerPort.2` with the settings in the `ts.jte` file. This host and port defines the Sun RI Java Platform, Enterprise Edition implementation used as the interop test machine. See `tests/interop/webservices` for these tests.

5.4.23.2 Webservice Endpoint WSDL Elements

Setting Endpoint Address

`element : endpoint-address-uri`

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

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

- **EJB Example:**

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

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

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

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

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

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

```
            <endpoint-address-uri>/GoogleSearch</endpoint-address-uri>
          </webservice-endpoint>
     </servlet>
</web>
```

In this case, the target endpoint address would be :

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

EJB Endpoint Security

`element : login-config`

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

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

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

Transport Guarantee

`element : transport-guarantee`

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

Publishing Final WSDL During Deployment

- **URL publishing:** no extra information required.

  The final WSDL document for each webservice endpoint is always published to a URL having the following syntax:

  - *EJB endpoints:*

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

- *Servlet endpoints:*

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

  or

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

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

- **File publishing:**

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

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

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

5.4.23.3 Webservice Client WSDL Elements

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

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

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

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

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

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

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

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

Resolving Container-Managed Ports

`element : wsdl-port`

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

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

Setting Stub Properties

`element : stub-property`

These are used to have the container set any of the properties defined in `javax.xml.rpc.Stub` on the stub/dynamic proxy object returned to the application from the Service instance. The property name must match the *value* of the `javax.xml.rpc.Stub` property constants. The stub properties are set per Port within the `service-ref`. Examples are shown below.

- **Setting endpoint address:**

```
<service-ref>
                  <service-ref-name>service/GoogleSearchProxy</service-ref-name>
                  <port-info>
                    <service-endpoint-interface>googleclient.GoogleSearchPort
                    </service-endpoint-interface>
                    <wsdl-port>
                       <namespaceURI>urn:GoogleSearch</namespaceURI>
                       <localpart>GoogleSearchPort</localpart>
                    </wsdl-port>
                    <stub-property>
                       <name>javax.xml.rpc.service.endpoint.address</name>
                       <value>http://localhost:8000/google/GoogleSearch</value>
                    </stub-property>
                   </port-info>
```

- **Setting Basic Auth properties:**

```
<service-ref>
        <service-ref-name>service/GoogleSearchProxy</service-ref-name>
        <port-info>
```

```
                    <service-endpoint-interface>googleclient.GoogleSearchPort
                    </service-endpoint-interface>
                    <wsdl-port>
                       <namespaceURI>urn:GoogleSearch</namespaceURI>
                       <localpart>GoogleSearchPort</localpart>
                    </wsdl-port>
                    <stub-property>
                       <name>javax.xml.rpc.security.auth.username</name>
                       <value>javaee</value>
                    </stub-property>
                    <stub-property>
                       <name>javax.xml.rpc.security.auth.password</name>
                       <value>javaee</value>
                    </stub-property>
                  </port-info>
```

- **Setting Logging property (Implementation-specific):**

  Name: `com.sun.enterprise.webservice.client.transport.log`

  Value: a `file` URL

  This is useful for debugging. When set, all soap/http requests and responses made through the associated stub will be logged to a file.

```
<module-name>appclient.jar</module-name>
      <service-ref>
         <service-ref-name>service/GoogleSearch</service-ref-name>
         <port-info>
           <service-endpoint-interface>google.GoogleSearchPort
           </service-endpoint-interface>
           <wsdl-port>
              <namespaceURI>urn:GoogleSearch</namespaceURI>
              <localpart>GoogleSearchPort</localpart>
           </wsdl-port>
           <stub-property>

   <name>com.sun.enterprise.webservice.client.transport.log</name>
             <value>file:/tmp/jaxrpc.log</value>
           </stub-property>
         </port-info>
      </service-ref>
```

Setting Call Properties

`element : call-property`

Call properties are set on `service-ref` for `Call` objects returned from `javax.xml.rpc.Service.createCall()`. This is the only kind of `Call` object that is not tied to a port.

```
<service-ref>
      <service-ref-name>service/GoogleSearch</service-ref-name>
      <call-property>
        <name>javax.xml.rpc.security.auth.username</name>
        <value>javaee</value>
      </call-property>
      <call-property>
        <name>javax.xml.rpc.security.auth.password</name>
        <value>javaee</value>
      </call-property>
</service-ref>
```

Call properties are set within `port-info` for all `other` `javax.xml.rpc.Service` methods that return `Call` objects.

```
<module-name>appclient.jar</module-name>
      <service-ref>
         <service-ref-name>service/GoogleSearch</service-ref-name>
         <port-info>
           <wsdl-port>
              <namespaceURI>urn:GoogleSearch</namespaceURI>
              <localpart>GoogleSearchPort</localpart>
           </wsdl-port>
           <call-property>
             <name>javax.xml.rpc.security.auth.username</name>
             <value>javaee</value>
           </call-property>
           <call-property>
             <name>javax.xml.rpc.security.auth.password</name>
             <value>javaee</value>
           </call-property>
         </port-info>
      </service-ref>
```

The allowable properties are defined in the javadoc for `javax.xml.rpc.Call`.

Overriding WSDL

`element : wsdl-override`

The `wsdl-override` element forces the deployment process to use a different WSDL than the one associated with a `service-ref` in the standard deployment module. This element is optional if the `service-ref` WSDL is full WSDL, and is required if partial WSDL. In all cases, it must point to a valid URL of a full WSDL document. Some examples are shown below.

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

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

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

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

5.4.24 Signature Test Setup

5.4.24.1 `sigTestClasspath` Property

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

```
sigTestClasspath=<jar_to_test>:<jars_used_by_yours>
```

where:

- *jar_to_test* — The JAR file you are validating when running the signature tests; when running against the Java Platform, Enterprise Edition RI, set to javaee.jar
- *jars_used_by_yours* — The JAR file(s) that are used or referenced by your JAR file; must include any classes that might be extended or implemented by the classes in your *jar_to_test*; include `rt.jar` when running against the Java Platform, Enterprise Edition RI

5.4.24.2 Additional Signature Test Information

The Java EE 6 CTS signature tests perform verifications in two different modes: *static* and *reflection*. The test results list which signature tests pass or fail, and the mode (static or reflection) for that test. As a troubleshooting aid when failures occur, consider the following:

- **All static mode tests fail:**

  The likely cause is that the `sigTestClasspath` needs modification. When running on Windows, be sure to use semicolons (`;`) for `CLASSPATH` separators.
- **For all other signature test failures:**

  Check the report output from the test to determine which tests failed and why.

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

5.4.25 Backend Database Setup

The following sections address special backend database setup considerations:

- "5.4.25.1 Setup Considerations for mySQL" on page 108
- "5.4.25.2 Setup Considerations for MS SQL Server" on page 109

5.4.25.1 Setup Considerations for mySQL

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

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

5.4.25.2 Setup Considerations for MS SQL Server

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

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

5.5 Using the JavaTest Harness Configuration GUI

You can use the JavaTest harness GUI to modify general test settings and to quickly get started with the default CTS test environment. After familiarizing yourself with these basic configuration settings, you will probably want to continue with the instructions in "5.4 Modifying Environment Settings for Specific Technology Tests" on page 63.

5.5.1 Basic Configuration Overview

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

The JavaTest harness requires two types of configuration information:

- **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.

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

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

Once the JavaTest harness GUI is displayed, whenever you choose *Run Tests->Start* to begin a test run, the JavaTest harness determines whether all of the required configuration information has been supplied:

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

5.5.2 The Configuration Interview

To configure the JavaTest harness to run the Java EE 6 CTS tests, complete the following steps. Note that you only need to complete these steps the first time you start the JavaTest harness. After you complete these steps, you can either run all or a subset of the tests, as described in Chapter 7, "Executing Tests."

▼ To Perform a Basic Configuration

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

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

   The Welcome screen displays.

2. **Click *File->Create* Work Directory to create a new work directory.**

   If you already have a working directory you want to use, click *File->Open Work Directory* instead.

   At this point, the JavaTest harness is preconfigured to run the basic CTS tests.

3. **If you want to run the test suite at this time using your current configuration settings, click *Run Tests->Start* from the main menu.**

   The default tests are executed with the default configuration settings.

   If you do not want to run the test suite at this time, continue with the steps below to modify your test configuration.

4. **Click *Configure->Edit Configuration* from the main menu.**

   The Configuration Welcome screen displays.

5. **Click *Next* (right arrow).**

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

6. **Accept the default configuration file and click *Next*.**

   You are prompted to specify a test environment.

7. **Select either *ts_unix* or *ts_win32*, and then click *Next*.**

   Choose *ts_unix* if you are running the tests in a Unix or Linux environment; choose *ts_win32* if you are running the tests under Windows.

   After making your selection and clicking *Next*, you are prompted to specify whether you want to run all or a subset of the test suite.

8. **Specify whether you want to run all or a subset of the tests, and then click *Next*.**

   Select *Yes* to run a subset of the tests; select *No* to run all tests.

   If you select *Yes*, proceed to the next step. If you select *No*, skip to Step 10.

9. **Select the tests you want to run from the displayed test tree, and then click *Next*.**

   You can select entire branches of the test tree, or use Ctrl+Click or Shift+Click to select multiple tests or ranges of tests, respectively.

10. **Specify whether you want to use an exclude list, and then click *Next*.**

    Select *Yes* to use an exclude list; select *No* to not use an exclude list.

    If you select *Yes*, proceed to the next step. If you select *No*, skip to Step 13.

11. **Specify the exclude list you want to use, and then click *Next*.**

    Select *initial* to use the default list; select *custom* to use a custom list.

    If you select *custom*, proceed to the next step. If you select *initial*, skip to Step 13.

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

13. **Click *Done* to accept and save your configuration settings.**

    You are prompted to specify the location in which you want to save your configuration settings.

14. **Specify the file in which you want to save your configuration settings, and then click *Save File*.**

    You are returned to the JavaTest main window.

15. **If you want to run the test suite at this time using your current configuration settings, click *Run Tests->Start* from the main menu.**

    The default tests are executed with the settings you specified.

◆ ◆ ◆ CHAPTER 6

Setup and Configuration for Testing with the Java EE 6 Web Profile

This chapter describes how to configure the Java EE 6 CTS test suite to work with your Java EE 6 Web Profile test environment. It is recommended that you first set up the testing environment using the Java EE 6 Web Profile RI and then with your Java EE 6 Web Profile server.

6.1 Configuring the Java EE 6 Web Profile Test Environment

The instructions in this section and in "Configuring Your Application Server as the VI" on page 55 step you through the configuration process for the Sun Solaris, Microsoft Windows, and Linux platforms.

6.1.1 Running Tests Against a Java EE 6 Web Profile Implementation

The Java EE 6 CTS is the Technology Compatibility Kit (TCK) for the Java Platform, Enterprise Edition as well as the Java EE 6 Web Profile. Implementations of the full Java Platform, Enterprise Edition must pass all of the tests as defined by Java EE 6 CTS Rules in Chapter 2, "Procedure for Java Platform, Enterprise Edition 6 Certification."

Implementations of the Java EE 6 Web Profile must run the tests that verify requirements defined by the Java EE 6 Web Profile Specification. These tests are defined by the Rules in Chapter 3, "Procedure for Java Platform, Enterprise Edition 6 Web Profile Certification." These requirements are a subset of the tests contained in the Java EE 6 CTS test suite. The test suite provides a mechanism whereby only those tests for the Java EE 6 Web Profile will be run. "To Run Tests Against a Java EE 6 Web Profile Implementation" on page 114 explains how to use this mechanism.

▼ To Run Tests Against a Java EE 6 Web Profile Implementation

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

   ```
   javaee.level= web
   ```

   This setting will allow WAR files only (that is, no EAR files) to be passed to the Deployment Porting Package.

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

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

     ```
     <TS_HOME>/tools/ant/bin/ant -Dkeywords=javaee_web_profile
     runclient
     ```

     Only tests that are required by the Java EE 6 Web Profile will be run.

     Note – If you are in a test directory that contains no Java EE 6 Web Profile tests, the test run will abort and report back that no tests were found.

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

     a. **Select View→Filters→CurrentConfiguration.**

     b. **Select Configure→ChangeConfiguration→Keywords.**
        The Keywords dialog will be displayed.

     c. **In the Keywords dialog, check the Select Tests that Match check box, specify the javaee_web_profile keyword in the text field, then click Done.**
        Only those tests that are valid in the Java EE 6 Web Profile will be enabled in the test tree.

6.1.2 Running Optional Subsets of Tests in Addition to the Java EE 6 Web Profile Tests

Keywords can be used to run subsets of tests from additional areas that are not required by the Java EE 6 Web Profile specification. For example, if your server implements the Java EE 6 Web Profile and the Java Connector Architecture 1.6 technology, set the keywords to "javaee_web_profile|connector_web_profile" to enable running tests for both areas. The command below shows how to specify these keywords to run the tests in both areas.

```
<TS_HOME>/tools/ant/bin/ant
-Dkeywords="(javaee_web_profile|connector_web_profile)
runclient
```

If you add "`&(jsp_vehicle|servlet_vehicle|web_vehicle)`" to the keyword expression above, the test harness will run any javaee_web_profile or connector test in the jsp, servlet, or web vehicle(s) only. This instruction would be necessary when running optional areas of tests in a Web Profile environment to limit the containers in which the tests will be run. Keywords for each vehicle type are available in the form of <*vehicle name*>`_vehicle`.

Table 6–1 lists optional subsets of tests that can be run for the Web Profile and provides the technology-to-keyword mappings for each of the optional areas.

TABLE 6–1 Keyword to Technology Mappings for Web Profile Optional Subsets

Technology	Keyword(s)
Connector	`connector_web_profile`
JACC	`jacc_web_profile`
JASPIC	`jaspic_web_profile`
JavaMail	`javamail_web_profile`
JAXR	`jaxr_web_profile`
JAX-RPC	`jaxrpc_web_profile`
JAX-RS	`jaxrs_web_profile`
JAX-WS	`jaxws_web_profile`
JMS	`jms_web_profile`
SAAJ	`saaj_web_profile`
XA	`xa_web_profile`

To add tests for other technologies, select the appropriate keyword(s) from Table 6–1. This table provides a mapping of keywords to *optional* technologies (test directories) in the test suite and indicates optional test areas for the Java EE 6 Web Profile.

◆ ◆ ◆ CHAPTER 7

Executing Tests

The Java EE 6 CTS 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 inChapter 4, "Installation," and Chapter 5, "Setup and Configuration," respectively.

7.1 Java EE 6 CTS Operating Assumptions

The following is assumed in this chapter:

- Java Platform, Enterprise Edition RI is installed and configured as described in this guide
- Java SE 6 SDK is correctly installed and configured on the host machine
- Java EE 6 CTS is installed and configured as described in this guide
- Implementations of the technologies to be tested are properly installed and configured.

7.2 Starting JavaTest

There are two general ways to run the Java EE 6 CTS using the JavaTest harness software:

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

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

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

Note – The Ant build tool, which is included in the Java EE 6 CTS bundle, is found in the `<TS_HOME>/tools/ant/bin` directory. The `build.xml` file in `<TS_HOME>/bin` contains the various Ant targets for the Java EE 6 CTS test suite

---

▼ To Start JavaTest in GUI Mode

1. **Set `TS_HOME` to the directory in which the Java EE 6 CTS is installed.**

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

3. **Ensure that the `ts.jte` file contains information relevant to your setup.**

   Refer to Chapter 5, "Setup and Configuration," for detailed configuration instructions.

4. **Execute the `<TS_HOME>/tools/ant/bin/ant gui` target to start the JavaTest GUI:**

   ```
   ./ant gui
   ```

   Using JavaTest GUI to run CTS tests is described later in this guide. Detailed information about the JavaTest interface is provided in the JavaTest User's Guide.

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

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

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

To set one of these keywords in the Javatest GUI, use the *Configure->Change Configuration->Keywords* menu item, and set the appropriate keyword.

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

Note – If you are running Interop or JWS/JAX-WS reverse tests, which run against the Java EE 6 Reference Implementation, you must start the standalone deployment server in a separate shell on the same host as the CTS harness. The default deployment porting implementation goes through a standalone deployment server with a dedicated classpath. To start the standalone deployment server, change to the `<TS_HOME>/bin` directory and execute the `start.auto.deployment.server` Ant task.

▼ To Start JavaTest in Command-Line Mode

1. **Set <TS_HOME> to the directory in which Java EE 6 CTS was installed.**

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

3. **Ensure that the `ts.jte` file contains information relevant to your setup.**

   Refer to Chapter 5, "Setup and Configuration," for detailed configuration instructions.

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

   ```
   <TS_HOME>/tools/ant/bin/ant runclient
   ```

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

Note – The `forward` and `reverse` keywords are available to filter the interop and/or rebuildable tests during a selected test run when running tests in one of the following directories only:

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

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

To set one of these keywords when running in command-line mode, set the appropriate keyword using the keyword system property; for example:

```
<TS_HOME>/tools/ant/bin/ant -Dkeywords=forward runclient
```

or

```
<TS_HOME>/tools/ant/bin/ant -Dkeywords=reverse runclient
```

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

Note – If you are running Interop or JWS/JAX-WS reverse tests, which run against the Java EE 6 Reference Implementation, you must start the standalone deployment server in a separate shell on the same host as the CTS harness. The default deployment porting implementation goes through a standalone deployment server with a dedicated classpath. To start the standalone deployment server, change to the `<TS_HOME>/bin` directory and execute the `start.auto.deployment.server` Ant task.

Example 7–1 Running the Java EE 6 CTS Signature Tests

To run the Java EE 6 CTS signature tests, enter the following commands:

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

Example 7–2 Running a Single Test Directory

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

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

Example 7–3 Running a Subset of Test Directories

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

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

7.3 Validating Your Test Configuration

▼ To Validate Your Configuration in GUI Mode

1. **Start the JavaTest GUI and step through the basic configuration steps, if required, as described in "5.5.2 The Configuration Interview" on page 110.**

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

3. **Highlight the `samples` directory, right-click, and choose *Execute These Tests*.**

   If a work directory has not been specified, you are prompted to specify or create a new one.

4. **Click *File->Create Work Directory* on the JavaTest main menu. The Create Work Directory dialog is displayed.**

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

6. **Click *Run Tests->Start* from the JavaTest main menu to run the default tests.**

   If your configuration information is incomplete, you are prompted to supply the missing parameters.

   The JavaTest status bar grows while JavaTest tracks statistics relative to the files done, tests found, and tests done.

7. **Check the results.**

   Test progress and results are displayed by the JavaTest harness.

▼ To Validate Your Configuration in Command-Line Mode

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

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

```
<TS_HOME>/tools/ant/bin/ant runclient
```

All sample tests will be run, and should pass.

3 **Generate test reports by executing the following commands:**

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

```
<TS_HOME>/bin
```

b. **Run the `report` Ant target:**

```
./ant report
```

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

Note – Although you can run the Ant report target from any test directory, its support is not guaranteed in the lower level directories. It is recommended that you always run the report target from `<TS_HOME>/bin`, from which reports are generated containing information about which tests were or were not run.

7.4 Running a Subset of the Tests

▼ To Run a Subset of Tests in GUI Mode

1 **Click** *Configure->Edit Configuration* **from the JavaTest main menu.**

The Configuration Editor is displayed.

2 **Click** *Specify Tests to Run?* **from the option list on the left.**

You are asked whether you want to run all or a subset of the test suite.

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

4 **Select the tests you want to run from the displayed test tree, and then click *Done*.**

You can select entire branches of the test tree, or use Ctrl+Click or Shift+Click to select multiple tests or ranges of tests, respectively.

After clicking *Done*, you are returned to the JavaTest main window.

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

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

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

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

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

```
<TS_HOME>/tools/ant/bin/ant runclient
```

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

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

You can run certain tests in batch mode based on the test's prior run status by specifying the `priorStatus` system property when invoking `<TS_HOME>/tools/ant/bin/ant`.

● **Invoke `<TS_HOME>/tools/ant/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 Java EE 6 tests with a status of failed and error by invoking the following commands:

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

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

7.5 Test Reports

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

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

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

7.5.1 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.

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:**

   `<TS_HOME>/tools/ant/bin/ant report -Dreport.dir="`*<report_dir>*`"`

   Reports are written for the last test run to the directory you specify.

   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:

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

Troubleshooting Although you can run the `report` Ant target from any test directory, its support is not guaranteed in the lower level directories. It is recommended that you always run the `report` target from `<TS_HOME)/bin`, from which reports are generated containing information about which tests were or were not run.cc

7.5.2 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`.**

The current report directory is displayed when you run the `report` target.

◆ ◆ ◆ CHAPTER 8

Debugging Test Problems

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

This chapter includes the following topics:

8.1 Overview

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

- **Errors** — Tests with errors could not be executed by the JavaTest harness. These errors usually occur because the test environment is not properly configured.
- **Failures** — Tests that fail were executed but had failing results.

The Test Manager GUI provides you with a number of tools for effectively troubleshooting a test run. See the *JavaTest User's Guide* and JavaTest online help for detailed descriptions of the tools described in this chapter.

8.2 Test Tree

Use the test tree in the JavaTest GUI to identify specific folders and tests that had errors or failing results. Color codes are used to indicate status as follows:

- **Green** — Passed
- **Blue** — Test Error
- **Red** — Failed to pass test
- **White** — Test not run
- **Gray** — Test filtered out (not run)

8.3 Folder Information

Click a folder in the test tree in the JavaTest GUI to display its 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.

8.4 Test Information

To display information about a test in the JavaTest GUI, click its icon in the test tree or double-click its name in a folder status 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.

8.5 Report Files

Report files are another good source of troubleshooting information. You may view the individual test results of a batch run in the JavaTest Summary window, but there are also a wide range of HTML report files that you can view in the JavaTest ReportBrowser or in the external browser or your choice following a test run. See "7.5 Test Reports" on page 124 for more information.

8.6 Configuration Failures

Configuration failures are easily recognized because many tests fail the same way. When all your tests begin to fail, you may want to stop the run immediately and start viewing individual test output. However, in the case of full-scale launching problems where no tests are actually processed, report files are usually not created (though sometimes a small `harness.trace` file in the report directory is written).

When aborting a test run, consider the following:

- If you abort a test run when running the JavaTest harness in GUI mode, the GUI tools automatically cleans up your environment for the next test run. This cleanup includes undeploying any components or applications that may deployed or registered with the Application Server.
- If you abort a test run in command-line mode (by pressing Ctrl+C), your environment might not be left in a clean state, causing potential failures in subsequent test runs. In such cases, you may need to perform the following procedure to restore your environment to a clean state.

▼ To Restore Your Environment After Aborting a Test Run in Command-line Mode

1. **Log in to the Application Server with the `asadmin` command.**
2. **List all registered components with the `asadmin list-components` command.**
3. **Undeploy any listed components related to your test run with the `asadmin undeploy` *<listed_component>* command.**

CHAPTER 9

Troubleshooting

This chapter explains how to debug test failures that you could encounter as you run the Java Platform, Enterprise Edition Compatibility Test Suite.

9.1 Common CTS Problems and Resolutions

This section lists common problems that you may encounter as you run the Java Platform, Enterprise Edition Compatibility Test Suite software on the Java Platform, Enterprise Edition RI. It also proposes resolutions for the problems, where applicable.

- **Problem:**

  The following exception may occur when a Java EE 6 CTS test tries to write a very long tracelog:

  ```
  java.lang.StringIndexOutOfBoundsException: String
  index out of range:
  -13493
  at java.lang.String.substring(String.java:1525)
  at java.lang.String.substring(String.java:1492)
  at javasoft.sqe.javatest.TestResult$Section
  $WritableOutputBuffer.write(TestResult.java:650)
  at java.io.Writer.write(Writer.java:153)
  at java.io.PrintWriter.write(PrintWriter.java: 213)
  at java.io.PrintWriter.write(PrintWriter.java: 229)
  at java.io.PrintWriter.print(PrintWriter.java: 360)
  at java.io.PrintWriter.println(PrintWriter.java:497)
  at javasoft.sqe.javatest.lib.ProcessCommand
  $StreamCopier.run(ProcessCommand.java:331)
  ```

  The execution of the test will either fail or hang.

  Resolution:

  Set the `-Djavatest.maxOutputSize=`*nnn* system parameter in the `runclient` and/or `gui` targets in the `<TS_HOME>/bin/build.xml` file to a value that is higher than the default setting of `100,000` on the JavaTest VM.

- **Problem:**

  When you start the Java Platform, Enterprise Edition RI on Windows XP/2000 via the `javaee -verbose` command line, the system may not find the specified path and could display one of the following errors:

  ```
  "Verify that JAVA_HOME is set correctly"
  "Verify that JAVAEE_HOME is set correctly"
  ```

  Resolution:

  Set `JAVA_HOME` to the path where the version of Java being used was installed and set `JAVAEE_HOME` to the location of the Java Platform, Enterprise Edition installation directory.

- **Problem:**

  If the `cts.jar` and the `tsharness.jar` files are not loadable by the extension classloader of your Java Platform, Enterprise Edition server, the following exception will be displayed in the window where the server was started when you attempt to run the tests:

  ```
  java.lang.NoClassDefFoundError: com/sun/cts/util
  RemoteLoggingInitException
  ```

  Resolution:

  Ensure that the `cts.jar` and `tsharness.jar` files can be loaded by the extension class loader of your Java Platform, Enterprise Edition server.

9.2 Support

After completing the troubleshooting process explained in this chapter, if you still cannot resolve your test problems, please contact the Java Partner Engineering representative assigned to your account.

CHAPTER 10

Building and Debugging Tests

For final certification and branding, all tests must be run through the JavaTest test harness. However, you can execute different Ant targets during your build and debug cycle. The following sections describe how to use Ant with the following targets to rebuild, list, and run tests:

- `runclient`
- `clean`
- `build`
- `ld`, `lld`, `lc`, `llc`, `pd`, `pc`

Licensees can only run the Sun-built version of the tests for certification, except in the case of rebuildable tests. See Appendix H, "Rebuilding the JAX-WS and JWS Tests," for more information about rebuildable tests.

This chapter includes the following topics:

10.1 Configuring Your Build Environment

▼ To Configure Your CTS Build Environment

Complete the following steps to set up your environment to build, deploy, and run the CTS tests using Ant. The following example is for the Solaris platform:

1 **Set the following environment variables in your shell environment to use the build infrastructure that comes with the TCK:**

a. **`TS_HOME` to the directory in which the Java EE 6 CTS software is installed.**

b. **`TS_HOME/bin` to your `PATH` in your command shell.**

C Shell:

```
setenv PATH ${TS_HOME}/bin:${PATH}
```

Bourne Shell:

```
PATH=${TS_HOME}/bin:${PATH}
export PATH
```

c. **`JAVA_HOME` to the directory in which the J2SE 5 software is installed.**

d. **`JAVAEE_HOME` to the directory in which the Java Platform, Enterprise Edition RI is installed.**

e. **Unset `ANT_HOME`, if it is currently set in your environment.**

2 **Change to the `<TS_HOME>/bin` directory and verify that the `ts.jte` file has the following properties set:**

a. **`webserver.home` the directory in which the Java Web Server is installed.**

b. **`webserver.host` to the host on which the Web server is running.**

c. **`webserver.port` to the port on which the Web server is running.**

d. **`javaee.home.ri` to the directory in which the Java Platform, Enterprise Edition RI is installed for reference to the packager tool used by the build infrastructure.**

e. **`ts.classpath` required classes needed for building/running the TCK.**

10.2 Building the Test

▼ To Build the CTS Tests

To build the Java EE 6 CTS tests using Ant, complete the following steps:

1 **To build a single test directory, type the following:**

```
cd <TS_HOME>/src/com/sun/ts/tests/test_dir
<TS_HOME>/tools/ant/bin/ant clean build
```

This cleans and builds the tests in the directory specified for *test_dir*.

2 **To list the classes directory for this test that was built, type the following:**

```
<TS_HOME>/tools/ant/bin/ant lc
```

or

```
<TS_HOME>/tools/ant/bin/ant llc
```

3 **To list the distribution directory of archives for this test that was built, type the following:**

```
<TS_HOME>/tools/ant/bin/ant pd
```

or

```
<TS_HOME>/tools/ant/bin/ant pc
```

10.3 Running the Tests

To run the Java EE 6 CTS tests using Ant, use one of the following procedures.

▼ To Run a Single Test Directory

● **To run a single test directory, type the following:**

```
cd <TS_HOME>/src/com/sun/ts/tests/test_dir
<TS_HOME>/tools/ant/bin/ant runclient
```

This runs all tests in *test_dir*.

▼ To Run a Single Test Within a Test Directory

● **To run a single test within a test directory, type the following:**

```
cd <TS_HOME>/src/com/sun/ts/tests/test_dir
<TS_HOME>/tools/ant/bin/ant runclient -Dtest=test_name
```

This runs only the *test_name* in the *test_dir* test directory. To show all the tests that can be run from a particular test directory, change to the directory and execute the `list.tests` Ant task. The actual test name displays to the right of the pound sign (#), which follows the fully qualified name of the client class.

10.4 Listing the Contents of `dist/classes` Directories

You can use various Ant targets to list the contents of corresponding `dist/classes` directories from the `src` directory without leaving the `src` directory. All listings are sorted by modification time, with the most recent modification listed first. Output is redirected to `more`. The format may vary on Windows and Unix. Ant does not support changing directory into the `dist/classes` directories, but you can copy and paste the first line of the output, which is the target path.

The Ant list targets are as follows:

- `ld` — Lists the contents of the current test's dist directory
- `lld` — Provides a long listing of the contents of the current test's dist directory
- `lc` — Lists the contents of the current test's classes directory
- `llc` — Provides a long listing of the contents of the current test's classes directory
- `pd` — Starts a new shell placed into the current test's dist directory
- `pc` — Starts a new shell placed into the current test's classes directory

If you run these targets in a directory that is not under the `src` directory, they will list the contents of the current directory.

Note – `pc`, `lc`, and `llc` also support the `-Dbuild.vi` property for listing the rebuildable tests. The rebuildable tests are located under `<TS_HOME>/classes_vi_built` instead of `<TS_HOME>/classes`.

The following listing shows sample output for the Ant `lc` target.

```
cd $TS_HOME/src/com/sun/ts/tests/samples/ejb/ee/simpleHello
$TS_HOME/tools/ant/bin/ant lc
<TS_HOME>/classes/com/sun/ts/tests/samples/ejb/ee/simpleHello
----------------------------------------------------------------
Hello.class
HelloClient.class
HelloEJB.class
HelloHome.class

$TS_HOME/tools/ant/bin/ant -Dbuild.vi=true lc
<TS_HOME>/classes_vi_built/com/sun/ts/tests/samples/ejb/ee/simpleHello
----------------------------------------------------------------
Hello.class
HelloClient.class
HelloEJB.class
HelloHome.class
```

10.5 Debugging Service Tests

The Java EE 6 CTS service tests test the compatibility of the Java Platform, Enterprise Edition Service APIs: JavaMail, JDBC, JMS, JTA, JAX-WS, JWS, CAJ, and RMI over IIOP. The test suite contains sets of tests that the JavaTest harness, in conjunction with the Java EE 6 CTS harness extensions, runs from different Java Platform, Enterprise Edition containers (enterprise bean, JSP, Servlet, and application client). The test suite wraps each of these tests inside generic components, called vehicles. Each Java EE 6 CTS service test has been set up to run in a default set of vehicles. Each technology's specification determines this set. When run as part of the certification process, all service API tests must pass in their default vehicle set.

Refer to the `<TS_HOME>/src/vehicle.properties` file to for a list the default vehicle sets for the Java EE 6 CTS service API tests.

To help you debug service API tests, the test suite provides a mechanism that allows for fine-grained control over which tests you can run in specific vehicles. When you override the default vehicle set for a particular set of service tests, the new set of vehicles must be a subset of the valid vehicle set for that set of tests. If the new set is not a subset of the default set, the test suite will use the default set. Example 10–1 illustrates this mechanism.

Note – You can only use this mechanism for debugging purposes. For certification, you must run using the default set of vehicles.

10.5.1 Examples

EXAMPLE 10–1 Running RMI/IIOP Enterprise Edition Tests

To run the RMI/IIOP enterprise edition tests in the application client vehicle only, set the following system property in the `<TS_HOME>/bin/build.xml` file for the Ant `gui` or `runclient` targets:

```
<sysproperty key="tests_rmiiiop_ee.service_eetest.vehicles"
  value="appclient"/>
```

This property overrides the default vehicle set for all tests under the specified directory (and in every subdirectory of that directory). Note that the first part of the property name matches the tests directory structure as it appears under `<TS_HOME>` (with the underscore character (_) replacing any file separator, such as the slash character (/) or the backslash character (\).

Before you run the test(s), you should temporarily rename the file `<TS_HOME>/src/testsuite.jtd`.

EXAMPLE 10–2 Restricting the JDBC Test Run

To restrict the JDBC test run to the servlet and JSP vehicles only, set the following system property in the `<TS_HOME>/bin/build.xml` file for the Ant `gui` or `runclient` targets:

EXAMPLE 10–2 Restricting the JDBC Test Run *(Continued)*

```
<sysproperty key="tests_jdbc_ee.service_eetest.vehicles"
  value="servlet jsp"/>
```

Before you run the test(s), you should temporarily rename the file `<TS_HOME>/src/testsuite.jtd`.

Note that you must remove these properties before you run the Java EE 6 CTS test suite for certification.

10.5.2 Obtaining Additional Debugging Information

When running the JavaTest harness in command-line mode, you can obtain additional debugging information by setting the `HARNESS_DEBUG` environment variable, as follows:

```
setenv HARNESS_DEBUG=true
```

Subsequent runs with the Ant `runclient` command generate additional debugging information.

You can also generate additional test run information by seting the `<TS_HOME>/bin/ts.jte harness.log.traceflag` property as follows:

```
harness.log.traceflag=true
```

CHAPTER 11

Implementing the Porting Package

Some functionality in the Java Platform, Enterprise Edition platform is not completely specified by an API. To handle this situation, the Java EE 6 CTS test suite defines a set of interfaces in the `com.sun.cts.porting` package, which serve to abstract any implementation-specific code. The CTS also provides implementations of these interfaces to work with the Java Platform, Enterprise Edition RI.

You must create your own implementations of the porting package interfaces to work with your particular Java Platform, Enterprise Edition server environment. You also need to create a deployment plan for each deployable component (EAR, EJB JAR, WAR, and RAR files) in the test suite as defined by the Java Platform, Enterprise Edition platform and JSR-88. There is a new `getDeploymentPlan()` method on the `TSDeploymentInterface2` interface, which returns an input stream to your deployment plan.

Note – Vendors are required to intrepret the `ior-security-config` specified in Sun-specific EJB runtime XML file and configure the EJB according to the specified values. For more information, see "5.4.20.1 Mapping Roles" on page 93.

11.1 Overview

The Java Platform, Enterprise Edition RI uses a set of <*module-name-with-extension*>`.sun-`<*standard-deployment-desc-component-prefix*>`.xml` files that are associated with each deployable component. A CTS `DeploymentInfo` object parses the contents of several runtime XML files: `sun-application_1_4-0.xml`, `sun-application-client_1_4-0.xml`, `sun-ejb-jar_2_1-0.xml`, and `sun-web-app_2_4-0.xml`, and makes their content available to create deployment plans by means of the `getDeploymentPlan()` method.

To use specific implementations of these classes, you simply modify the following entries in the `porting class .1` section of the `ts.jte` environment file to identify the fully-qualified class names:

```
porting.ts.deploy2.class.1=[vendor-deployment-class]
porting.ts.login.class.1=[vendor-login-class]
porting.ts.url.class.1=[vendor-url-class]
porting.ts.jaxrpc.class.1=[vendor-jaxrpc-class]
porting.ts.jms.class.1=[vendor-jms-class]
porting.ts.HttpsURLConnection.class.1=[vendor-httpsURLConnection-class]
```

The `<TS_HOME>/src/com/sun/ts/lib/porting` directory contains the interfaces and `Factory` classes for the porting package.

Note – You must not modify any of the CTS V6 source code. Create your own implementations of these interfaces and point to them in the appropriate section of the `ts.jte` file.

Note the change to the deployment porting property above. It has changed to be `deploy2`. This is because there is a new deployment porting interface because of the standardization of a deployment API in Java Platform, Enterprise Edition. Any functionality that is still not addressed by this API is part of the new interface com.sun.ts.lib.porting.TSDeploymentInterface2.

Make sure your porting class implementations meet the following requirements:

- Implement the following porting interfaces:
  - `TSDeploymentInterface2`
  - `TSLoginContextInterface`
  - `TSURLInterface`
  - `TSJMSAdminInterface`
  - `TSHttpsURLConnectionInterface`
  - `TSJAXRPCInterface`
- Include the implementation of the previous interfaces in the classpaths of JavaTest, the test clients, and the test server components:
  - In the `ts.harness.classpath` property in the `<TS_HOME>/bin/ts.jte` file
  - In the `CLASSPATH` variable of the `command.testExecute` and `command.testExecuteAppClient` properties in the `ts.jte` file
  - In the classpath of your Java Platform, Enterprise Edition server

Note that because the JavaTest VM calls certain classes in the CTS porting package directly, porting class implementations are not permitted to exit the VM (for example, via the `System.exit` call).

11.2 Porting Package APIs

The following sections describe the API in the Java EE 6 CTS porting package. Sun provides the implementation classes that are used with the Java Platform, Enterprise Edition RI. These classes are located in the `<TS_HOME>/src/com/sun/ts/lib/implementation/sun/javaee` directory. You are encouraged to examine these implementations before you create your own.

Detailed API documentation for the porting package interfaces is available in the `<TS_HOME>/docs/api` directory. The API included in this section are:

- "11.2.1 `TSDeploymentInterface2`" on page 141
- "11.2.3 `TSJMSAdminInterface`" on page 143
- "11.2.4 `TSLoginContextInterface`" on page 143
- "11.2.5 `TSURLInterface`" on page 143
- "11.2.6 `TSHttpsURLConnectionInterface`" on page 144
- "11.2.7 `TSJAXRPCInterface`" on page 144

11.2.1 `TSDeploymentInterface2`

The Java EE 6 CTS test suite provides a new version of the Deployment porting interface. With the introduction of a standard deployment API in the J2EE 1.4 platform (via JSR-88), many of the porting methods in the original interface `TSDeploymentInterface` no longer require implementation-specific functionality. The Java EE 6 CTS test suite provides an implementation of the interface `TSDeploymentInterface`, which uses only the standard Deployment APIs defined by the Java Platform, Enterprise Edition platform. The following properties are still in the `ts.jte` file to reflect this and should not be changed:

- `porting.ts.deploy2.class.1=com.sun.ts.lib.deliverable.cts.deploy.StandardDeploymen`
- `porting.ts.deploy2.class.2=com.sun.ts.lib.deliverable.cts.deploy.StandardDeploymen`

The class `StandardDeployment14` also requires the following properties to be set in the `ts.jte` file:

- `deployManagerJarFile.1=${JAVAEE_HOME}/lib/deployment/sun-deploy.jar`
- `deployManageruri.1=deployer:Sun:AppServer:RI::localhost`
- `deployManageruname.1=foo`
- `deployManagerpasswd.1=bar`

These properties are necessary in order to get an instance of and interact with the `DeploymentManager` for your Java Platform, Enterprise Edition implementation.

The `deployManagerJarFile` property must point to the JAR file that contains the manifest entries necessary to get your `DeploymentManager` instance. The `deployManageruri` property represents the URI that is used to locate your `DeploymentManager`.

The `deployManageruname` and `deployManagerpasswd` properties are used when calling `DeploymentFactoryManager.getDeploymentManager`.

`StandardDeployment14` calls into the new deployment porting interface (`TSDeploymentInterface2`). Licensees must implement this new interface and set the following property in the `ts.jte` file to point to their implementation:

```
porting.ts.deploy2.class.1=com.sun.ts.lib.implementation.sun.JavaEE.SunRIDeployment2
```

The `TSDeployment2` class acts as a `Factory` object for creating concrete implementations of `TSDeploymentInterface2`. The concrete implementations are specified by the `porting.ts.deploy2.class.1` and `porting.ts.deploy2.class.2` properties in the `ts.jte` file. Each Java Platform, Enterprise Edition implementation must provide an implementation of the interface `TSDeploymentInterface2` to support the automatic deployment and undeployment of test applications by the JavaTest test harness. Providing this functionality enables the entire test suite to be run without having to manually deploy/undeploy the Java Platform, Enterprise Edition test applications prior to running the tests. The implementation provided with this release uses the semantics of the Java Platform, Enterprise Edition RI.

11.2.2 Ant-Based Deployment Interface

In addition to the Java-based deployment porting interfaces, Java EE 6 CTS introduces an Ant-based porting interface as well. The Java-based interface is still used for deployment/undeployment during test runs. The Ant-based interface is used when you want to only deploy/undeploy archives associated with a subdirectory of tests. The Ant-based deployment interface is used by the following:

- The `build.special.webservices.clients` target in the `${ts.home}/bin/build.xml` file

  This target deploys archives to your server implementation and then builds the client classes that use those archives. You must run this target before you run the tests under the `${ts.home}/src/com/sun/ts/tests/webservices12/specialcases` directory.
- The `deploy` and `undeploy` targets in each test subdirectory under the `${ts.home}/src/com/sun/ts/tests` directory

  To use these targets, which are useful for debugging, you must provide an Ant-based deployment implementation.

11.2.2.1 Creating Your Own Ant-based Deployment Implementation

The Ant-based deployment implementation for the Java EE 6 RI is under `${ts.home}/bin/xml/impl/glassfish` directory. To create your own implementation, create a `deploy.xml` file under the `${ts.home}/bin/xml/impl/<vendor-name>` directory. Within the file, create and implement the -deploy and -undeploy targets.

See `${ts.home}/bin/xml/impl/glassfish/deploy.xml` to see how these targets are implemented for the Java EE 6 RI .

Note – There is also a Java-based implementation of TSDeploymentInterface (`com.sun.ts.lib.implementation.sun.javaee.glassfish.AutoDeployment`). This implementation, which leverages the Java EE 6 RI implementation of the Ant-based deployment interface, calls the Ant targets programmatically.

11.2.3 `TSJMSAdminInterface`

JMS-administered objects are implementation-specific. For this reason, the creation of connection factories and destination objects have been set up as part of the porting package. Each Java Platform, Enterprise Edition implementation must provide an implementation of the `TSJMSAdminInterface` to support their own connection factory, topic/queue creation/deletion semantics.

The `TSJMSAdmin` class acts as a `Factory` object for creating concrete implementations of `TSJMSAdminInterface`. The concrete implementations are specified by the `porting.ts.jms.class.1` and `porting.ts.jms.class.2` properties in the `ts.jte` file.

If you wish to create the JMS-administered objects prior to executing any tests, you may use the default implementation of `TSJMSAdminInterface`, `SunRIJMSAdmin.java`, which provides a null implementation. In the case of the Java Platform, Enterprise Edition RI, the JMS administered objects are created during the execution of the `config.vi` Ant target.

There are two types of JMS-administered objects:

- A `ConnectionFactory`, which a client uses to create a connection with a JMS provider
- A `Destination`, which a client uses to specify the destination of messages it sends and the source of messages it receives.

11.2.4 `TSLoginContextInterface`

The `TSLoginContext` class acts as a `Factory` object for creating concrete implementations of `TSLoginContextInterface`. The concrete implementations are specified by the `porting.ts.login.class.1` property in the `ts.jte` file. This class is used to enable a program to login as a specific user, using the semantics of the Java Platform, Enterprise Edition RI. The certificate necessary for certificate-based login is retrieved. The keystore file and keystore password from the properties that are specified in the `ts.jte` file are used.

11.2.5 `TSURLInterface`

The `TSURL` class acts as a `Factory` object for creating concrete implementations of `TSURLInterface`. The concrete implementations are specified by the `porting.ts.url.class.1` property in the `ts.jte` file. Each Java Platform, Enterprise Edition implementation must provide an implementation of the `TSURLInterface` to support obtaining URL strings that are used to access a selected Web component. This implementation can be replaced if a Java Platform, Enterprise Edition server implementation requires URLs to be created in a different manner. In most Java Platform, Enterprise Edition environments, the default implementation of this class can be used.

11.2.6 TSHttpsURLConnectionInterface

The `TSHttpsURLConnection` class acts as a `Factory` object for creating concrete implementations of `TSHttpsURLConnectionInterface`. The concrete implementations are specified by the `porting.ts.HttpsURLConnection.class.1` and `.2` properties in the `ts.jte` file.

You must provide an implementation of `TSHttpsURLConnectionInterface` to support the class `HttpsURLConnection`.

Note – The `SunRIHttpsURLConnection` implementation class uses `HttpsURLConnection` from J2SE 6.

11.2.7 TSJAXRPCInterface

The `TSJAXRPC` class acts as a `Factory` object for creating concrete implementations of `TSJAXRPCInterface`. The concrete implementations are specified by the `porting.ts.jaxrpc.class.1` and `.2` properties in the `ts.jte` file.

You must provide an implementation of `TSJAXRPCInterface` to support the class `TSJAXRPC`. This class is used to provide as name/value pairs the URL value of the deployed webservice endpoints for those sets of tests which use DII, direct HTTP, or direct SAAJ to communicate to the endpoints.

APPENDIX A

Common Applications Deployment

Some tests in the test suite require the deployment of additional applications, components, or resource archives that are located in directories other than the test's directory. When the test harness encounters a test that requires these additional applications, components, or resource archives, they are passed to the `TSDeploymentInterface2` implementation to be deployed. Because these applications can be shared by tests in different test directories, they are called *common applications*.

Table A–1 lists the test directories and the directories that contain the common applications that are required by the test directories.

TABLE A–1 Required Common Applications

Directory Under `com/sun/ts/tests`	Directory Under `com/sun/ts/tests` With Associated Common Applications
`ejb/ee/tx/session`	`ejb/ee/tx/txbean`
`ejb/ee/tx/entity/bmp`	`ejb/ee/tx/txEbean`
`ejb/ee/tx/entity/cmp`	`ejb/ee/tx/txECMPbean`
`ejb/ee/tx/entity/pm`	`ejb/ee/tx/txEPMbean`
`connector/ee/localTx/msginflow`	`common/connector/whitebox`
`connector/ee/mdb`	`connector/ee/localTx`
`common/connector/whitebox`	`connector/ee/noTx`
`common/connector/whitebox`	`connector/ee/xa`
`common/connector/whitebox`	`connector/ee/connManager`
`common/connector/whitebox`	`xa/ee`
`common/connector/JDBCwhitebox/whitebox`	`common/connector/whitebox`

TABLE A–1 Required Common Applications *(Continued)*

Directory Under `com/sun/ts/tests`	Directory Under `com/sun/ts/tests` With Associated Common Applications
`compat13/connector/localTx`	`compat13/connector/whitebox`
`compat13/connector/noTx`	`compat13/connector/whitebox`
`compat13/connector/xa`	`compat13/connector/whitebox`
`interop/tx/session`	`interop/tx/txbean`
`interop/tx/entity`	`interop/tx/txEbean`
`interop/tx/webclient`	`interop/tx/txbean`
`tests/interop/csiv2`	`interop/csiv2/rionly`
`ejb/ee/pm/ejbql`	`ejb/ee/pm/ejbql/schema`
`ejb/ee/tx/session/stateful/bm/TxMDBMS_Direct`	`ejb/ee/tx/session/stateful/bm/TxMDBMSBeans/BeanA`
`ejb/ee/tx/session/stateful/bm/TxMDBMSBeans/BeanB`	`ejb/ee/tx/session/stateful/bm/TxMDBMSBeans/BeanC`
`tests/ejb/ee/tx/session/stateful/bm/TxMDBMS_Indirect`	`ejb/ee/tx/session/stateful/bm/TxMDBMSBeans/BeanA`
`ejb/ee/tx/session/stateful/bm/TxMDBMSBeans/BeanB`	`ejb/ee/tx/session/stateful/bm/TxMDBMSBeans/BeanC`
`ejb/ee/tx/session/stateful/bm/TxMDBSS_Direct`	`ejb/ee/tx/session/stateful/bm/TxMDBSSBeans/BeanA`
`ejb/ee/tx/session/stateful/bm/TxMDBSSBeans/BeanB`	`tests/ejb/ee/tx/session/stateful/bm/TxMDBSSBeans/BeanC`
`ejb/ee/tx/session/stateful/bm/TxMDBSS_Indirect`	`ejb/ee/tx/session/stateful/bm/TxMDBSSBeans/BeanA`
`ejb/ee/tx/session/stateful/bm/TxMDBSSBeans/BeanB`	`ejb/ee/tx/session/stateful/bm/TxMDBSSBeans/BeanC`

APPENDIX B

ts.jte Modifiable Properties

This appendix provides an alphabetical listing of all modifiable properties and their default values in the main Java EE 6 CTS configuration file, <TS_HOME>/bin/ts.jte. Although, technically speaking, all properties in ts.jte can be modified, it is recommended that you modify only the properties listed here. Properties not listed here either do not need to be changed under normal circumstances or should not be changed under any circumstance.

Refer to Chapter 5, "Setup and Configuration," for detailed CTS configuration instructions, and to the ts.jte file itself for complete comments and syntax examples.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values

Property	Default Value	Description
alt.dtd.dir	${ts.home}/lib/dtds	DTD location for Java Platform, Enterprise Edition and RI XML files. Used for XML validation when building tests. If javaee.home.ri is set, javaee.home.rilibdtds will be used and alt.dtd.dir is ignored.
alt.schema.dir	${ts.home}/lib/schemas	Schema location for Java Platform, Enterprise Edition and RI XML files. Used for XML validation when building tests. If javaee.home.ri is set, javaee.home.rilibschemas will be used and alt.schema.dir is ignored.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description
appclient.log.output	TRUE	The client logging level for the Application Client container depends on the logging level that is specified in the sun acc.xml file. This logging level directly affects the output of TSLogger, which logs the JSR196 SPI calls made in the Application Client container. This property is used to enable the Application Client container's logging level to INFO.
authenticationMethod	"UDDI_GET_AUTHTOKEN"	Set this to the authentication method for the JAXR provider. UDDI_GET_AUTHTOKEN is the get_AuthToken protocol defined by UDDI_API2.
authpassword	javajoe	Associated password for the authuser.
authuser	javajoe	Used to exercise the role mapping security feature.
certLoginUserAlias	cts	User alias for certificate-based login. This property is used in mutual authentication to pick up the certificate, based on the user alias.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description
command.testExecute	com.sun.ts.lib.harness.ExecTSTestCmd \ CLASSPATH=${ts.harness.classpath}: \ ${ts.home}/classes: \ ${JAVA_HOME}/../lib/tools.jar: \ ${ts.home}/lib/commons-httpclient-3.0.jar: \ ${ts.home}/lib/commons-logging-1.0.4.jar: \ ${ts.home}/lib/commons-codec-1.3.jar: \ ${ts.home}/lib/htmlunit-1.7.jar: \ ${ts.home}/lib/commons-collections-3.1.jar: \ ${ts.home}/lib/commons-io-1.0.jar: \ ${ts.home}/lib/commons-lang-2.0.jar: \ ${ts.home}/lib/jaxen-1.1-beta-6.jar: \ ${ts.home}/lib/js-1.6R1.jar: \ ${ts.home}/lib/nekohtml-0.9.5.jar: \ ${ts.home}/lib/saxpath.jar: \ ${ts.home}/lib/xercesImpl-2.6.2.jar \ DISPLAY=${ts.display} \ HOME="${user.home}" \ windir=${windir} \ SYSTEMROOT=${SYSTEMROOT} \ PATH="${javaee.home}/nativelib" \ ${JAVA_HOME}/bin/java \ -Dcts.tmp=$harness.temp.directory \ -Djava.security.policy=${ts.home}/bin/ \ harness.policy \ -Djava.security.manager \ -Djava.protocol.handler.pkgs=javax.net.ssl \ -Djavax.net.ssl.keyStore=${ts.home}/bin/ \ certificates/clientcert.jks \ -Djavax.net.ssl.keyStorePassword=changeit \ -Djavax.net.ssl.trustStore=${s1as.domain}/${sjsas.instance.config.di -Djava.endorsed.dirs=${s1as.java.endorsed.dirs} \ -Dcom.sun.aas.installRoot=${javaee.home} \ -Dcom.sun.aas.configRoot=${javaee.home}/config \ -Ddeliverable.class=${deliverable.class} \ $testExecuteClass $testExecuteArgs	This command is used to execute any test clients which are not run inside an application client container. For example, any URL clients or standalone Java clients would be executed with this command. Some test directories which make use of this command are servlet and jsp.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description

`command.testExecuteAppClient`	`com.sun.ts.lib.harness.ExecTSTestCmd \`
	`DISPLAY=${ts.display} HOME="${user.home}" \`
	`LD_LIBRARY_PATH=${javaee.home}/lib \`
	`windir=${windir} SYSTEMROOT=${SYSTEMROOT} \`
	`PATH="${javaee.home}/nativelib" TZ=${tz} \`
	`${JAVA_HOME}/bin/java \`
	`-Djava.security.policy=${javaee.home}/lib/\`
	`appclient/client.policy \`
	`-Dcts.tmp=$harness.temp.directory \`
	`-Djava.security.auth.login.config=${javaee.home}/\`
	`lib/appclient/appclientlogin.conf \`
	`-Djava.protocol.handler.pkgs=javax.net.ssl \`
	`-Dcom.sun.enterprise.home=${javaee.home} \`
	`-Djavax.net.ssl.keyStore=${ts.home}/bin/ \`
	`certificates/clientcert.jks \`
	`-Djavax.net.ssl.keyStorePassword=changeit \`
	`-Dcom.sun.aas.installRoot=${javaee.home} \`
	`-Dcom.sun.aas.imqLib=${javaee.home}/imq/lib \`
	`-Djava.util.logging.manager=com.sun.enterprise. \`
	`server.logging.ACCLogManager \`
	`-Djavax.net.ssl.trustStore=${s1as.domain}/ \`
	`${sjsas.instance.config.dir}/cacerts.jks \`
	`-Djava.endorsed.dirs=${s1as.java.endorsed.dirs} \`
	`-Djavax.xml.parsers.SAXParserFactory=com.sun.org.\`
	`apache.xerces.internal.jaxp.SAXParserFactoryImpl \`
	`-Djavax.xml.parsers.DocumentBuilderFactory=com. \`
	`sun.org.apache.xerces.internal.jaxp. \`
	`DocumentBuilderFactoryImpl \`
	`-Djavax.xml.transform.TransformerFactory=com. \`
	`sun.org.apache.xalan.internal.xsltc.trax. \`
	`TransformerFactoryImpl \`
	`-Dorg.xml.sax.driver=com.sun.org.apache.xerces. \`
	`internal.parsers.SAXParser \`
	`-Dorg.xml.sax.parser=org.xml.sax.helpers. \`
	`XMLReaderAdapter \`
	`-Doracle.jdbc.J2EE13Compliant=true \`
	`-Dstartup.login=false -Dauth.gui=false \`
	`-Djava.ext.dirs=${s1as.lib}/jdbcdrivers$ \`
	`{pathsep}${JAVA_HOME}/lib/ext${pathsep}$ \`
	`{javaee.home}/lib/jdbcdrivers${pathsep} \`
	`${javaee.home}/javadb/lib \`
	`-Dcom.sun.aas.configRoot=${javaee.home}/config \`
	`-Ddeliverable.class=${deliverable.class} \`
	`-classpath ${ts.run.classpath} com.sun.enterprise.\`
	`appclient.Main $testExecuteArgs -configxml \`
	`${ts.home}/tmp/appclient/s1as.sun-acc.xml`
		This command is used to execute the application client container for the vendor implementation VI. Note that `TS_HOME/classes` should not be in the `classpath` for this command since all client classes are self-contained in the application archive or referenced via the manifest.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description
`create.cmp.tables`	`TRUE`	When set to `FALSE`, the App Server is responsible for creating CMP tables at deployment of the EJB EAR When set to `TRUE`, `init.datbaseName` will create the tables used by CMP EJBs. The SQL for the CMP tables are contained in: `$TS_HOME/datbaseNamesqldatabaseName.d` `$TS_HOME/datbaseNamesqldatabaseName.d`
`db.dml.file`	`${derby.dml.file}`	DML file for VI.
`db.dml.file.ri`	`${derby.dml.file}`	DML file for RI.
`deploy.delay.in.minutes`	`5`	This property can be used to specify the amount of time in minutes that the test harness will wait for the JSR 88 `ProgressObject` to return either failed or completed from a `DeploymentManager` API call. After the time has elapsed, the harness will report failure for the given action.
`deployManagerJarFile.1`	`${javaee.home}/lib/deployment/sun-as-jsr88-dm.jar`	Location of the JAR file that contains the JSR 88 deployment manager factory class name for the Vendors Implementation VI.
`deployManagerJarFile.2`	`${javaee.home.ri}/lib/deployment/sun-as-jsr88-dm.jar`	Location of the JAR file that contains the JSR 88 deployment manager factory class name for the Reference Implementation RI
`deployManagerpasswd.1`	`${s1as.admin.passwd}`	Password for deployment process for the VI.
`deployManagerpasswd.2`	`${ri.admin.passwd}`	Password for deployment process for the RI.
`deployManageruname.1`	`${s1as.admin.user}`	User name for deployment process for the VI.
`deployManageruname.2`	`${ri.admin.user}`	User name for deployment process for the RI.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description
deployManageruri.1	deployer:Sun:AppServer::${deployment_host.1}:${s1as.admin.port}	JSR 88 management URL location of the deployment process for the VI.
deployManageruri.2	deployer:Sun:AppServer::${deployment_host.2}:${ri.admin.port}	JSR 88 management URL location of the deployment process for the RI.
deployment_host.1	${orb.host}	Name of machine running the JSR 88 deployment process for the vendor's Java Platform, Enterprise Edition implementation.
deployment_host.2	${orb.host.ri}	Name of machine running the JSR 88 deployment process for the Java Platform, Enterprise Edition RI.
derby.classes	${javaee.home}/javadb/lib/derbyclient.jar	CLASSPATH to JDBC driver classes.
derby.dbName	derbyDB	Database Name.
derby.dml.file	derby/derby.dml.sql	DML file used for CTS test cases.
derby.driver	org.apache.derby.jdbc.ClientDriver	DriverManager driver.
derby.passwd	cts1	User password configured.
derby.poolName	cts-derby-pool	Name of pool configured in the Java Platform, Enterprise Edition RI; do not change.
derby.port	1527	Database Server port.
derby.server	${orb.host}	Database Server.
derby.url	jdbc:derby://${derby.server}:${derby.port}/${derby.dbName};create=true	URL to the CTS database.
derby.user	cts1	User ID configured
ejb_timeout	30000	Amount of time, in milliseconds, that duration will be set for an ejbtimeout callback method
ejb_wait	60000	Amount of time, in milliseconds, that the client will wait for ejbtimeout callback method results
EJBServer1TxInteropEnabled	true	Transaction interoperability settings for Vendor Java Platform, Enterprise Edition EJB Server

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description
`EJBServer2TxInteropEnabled`	`true`	Transaction interoperability settings for Java Platform, Enterprise Edition RI EJB Server
`extension.dir`	`${s1as.domain}/lib/ext`	Extension directory for App Server under test (not RI) App Server vendors will need to set this to their App Server's extension directory. `config.vi` target copies the CTS library JAR files to this location.
`harness.executeMode`	`0`	Used to run the harness in the following modes of execution: ■ `0` — Default (deploy, run, undeploy) ■ `1` — Deploy only ■ `2` — Run only ■ `3` — Undeploy only ■ `4` — Deploy and run only
`harness.log.delayseconds`	`1`	Number of seconds to delay to allow reporting from remote clients to finish.
`harness.log.port`	`2000`	The port the harness listens on for log messages from remote clients.
`harness.log.traceflag`	`false`	Used to turn on or off verbose debugging output for the tests.
`harness.socket.retry.count`	`10`	Denotes the number of times we should attempt to create a server socket when initializing a test client. The socket is used for logging purposes.
`harness.temp.directory`	`${ts.home}/tmp`	Directory location used by the harness to store temporary files.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description

`if.existing.work.report.dirs`	`auto`	Specifies how existing `work.dir` and `report.dir` will be handled, and it must be one of the following values:
		■ `overwrite` — Overwrites all content in `work.dir` and `report.dir`
		■ `backup` — Moves all content in `work.dir` and `report.dir` to `work.dir_time_day_bak` and `report.dir_time_day_bak`, respectively
		■ `append` — Reuses and preserves the existing `work.dir` and `report.dir`
		■ `auto` — Lets the build files decide which mode to use (`overwrite`, `backup`, or `append`). The value is determined like this:
		`if.existing.work.report.dirs == auto`
		`  if in CTS workspace`
		`    if.existing.work.report.dirs = overw`
		`  else we are in a distribution bundle`
		`    if.existing.work.report.dirs = appen`
		`  end if`
		`else`
		`  if.existing.work.report.dirs = value i`
		`end if`

`impl.vi`	`glassfish`	This property corresponds to the vendor under test. The name represents the directory that contains vendor-specific Ant scripts that are located in the `$TS_HOME/bin/xml/impl/{imp.vi}` directory.
`impl.vi.deploy.dir`	`${s1as.domain}/autodeploy`	Auto deployment directory of the vendor's application server under test.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description
`impl.vi.host`	`${s1as.admin.host}`	Admin host where the Vendor Implementation is running.
`impl.vi.port`	`${s1as.admin.port}`	Admin port where the Vendor Implementation is running.
`instance.listenerName`	`orb-listener-1`	Default value for the IIOP listener for your instance. Users will most likely not need to change this.

`javaee.classes.ri`	`${ri.lib}/mail.jar:${ri.lib}/javaee.jar: \`
	`${ri.lib}/appserv-rt.jar: \`
	`${ri.lib}/appserv-ext.jar: \`
	`${ri.lib}/appserv-cmp.jar: \`
	`${ri.lib}/appserv-admin.jar: \`
	`${ri.imq.share.lib}/imqadmin.jar: \`
	`${ri.lib}/install/applications/jmsra/imqjmsra.jar: \`
	`${ri.imq.share.lib}/imq.jar: \`
	`${ri.imq.share.lib}/fscontext.jar: \`
	`${ri.lib}/jsf-api.jar:${ri.lib}/appserv-jstl.jar:\`
	`${ri.lib}/xercesImpl.jar:${ri.lib}/activation.jar: \`
	`${ri.lib}/appserv-ws.jar:${ri.lib}/dtds: \`
	`${ri.lib}/schemas: \`
	`${ri.lib}/toplink-essentials.jar: \`
	`${ri.lib}/dbschema.jar`
		Classes required by Java Platform, Enterprise Edition RI.

`javaee.home`	`/sun/appserver9`	The location of the vendor's Java Platform, Enterprise Edition platform implementation.
`javaee.home.ri`	`/sun/appserver9`	The location of the RI.

`javaee.home.ri.classpathsuffix`	`${javaee.home.ri}/lib/riinterceptors.jar: \`
	`${pathsep}${javaee.home.ri}/javadb/lib/ \`
	`derbyclient.jar`
		The classpath suffix of the RI used in interop tests. Must contain the RI JDBC driver JARs and the RMI interceptor classes.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description
javatest.timeout.factor	1.0	This property specifies the scale factor used by JavaTest to adjust the time JavaTest will wait for a given test to complete before returning failure. For instance if the default test timeout is 5 minutes, this value will be multiplied by 5 minutes to determine the total timeout delay. Note this value only works with JavaTest's batch mode `runclient`. When using the JavaTest GUI users must change this timeout factor in the GUI.
jaxrConnectionFactoryLookup	0	The preferred way for a client to lookup a JAXR `ConnectionFactory` is to use JNDI. An alternate method to lookup a JAXR `ConnectionFactory` is to use the `newInstance` static method on the `ConnectionFactory`. ■ `0` — Use JNDI lookup ■ `1` — Use `newInstance` method
jaxrJNDIResource	"java:comp/env/eis/JAXR"	JAXR `ConnectionFactoryReference` if JNDI lookup is being used. For example, `java:comp/env/eis/JAXR`. If not using JNDI lookup set to "".
jaxrPassword	""	Is used for setting connection credentials this must be set up in the UDDI registry.
jaxrPassword2	""	A user name for Association tests.
jaxrUser	""	Is used for setting connection credentials this must be set up in the UDDI registry.
jaxrUser2	""	A user name for Association tests.
jaxrWebContext	"web_content"	The context root for JAXR HTML tests pages.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description
`jdbc.db`	`derby`	The name of the currently configured DB. This value is the prefix of the DB properties currently being used. Some valid values are derby and `derbyEmbedded`. See the other DB property names for other valid values.
`jdbc.lib.class.path`	`${ts.home}/internal/lib`	This property is used by the `database.classes` properties to point to where the JDBC drivers live.
`jdbc.poolName`	`${derby.poolName}`	Configure the connection pool that will be tested in this CTS test run.
`jms_timeout`	`5000`	Amount of time, in milliseconds, that synchronous receives will wait for a message
`jstl.db.driver`	`${derby.driver}`	JDBC driver
`jstl.db.url`	`${derby.url}`	DB URL
`log.file.location`	`${s1as.domain}/logs`	This property is used by JACC tests to create and analyze provider logs. Specify the log directory in which your App Server generates logs.
`login`	`default`	This property must be set to run app client security tests.
`mailFrom`	`xyz@sun.com`	Reply to address set in the email messages generated by the JavaMail tests.
`mailHost`	`bur-mail2.east.sun.com`	Hostname of the SMTP server.
`mailuser1`	`some-valid-email-user-name@sun.com`	Must be set to a valid email address where test mails will be sent.
`namingServiceHost1`	`${orb.host}`	Naming Service host name for the Vendor's Implementation VI.
`namingServiceHost2`	`${orb.host.ri}`	Naming Service host name for the Reference Implementation RI.
`namingServicePort1`	`${orb.port}`	Naming Service port for the VI.
`namingServicePort2`	`${orb.port.ri}`	Naming Service port for the RI.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description
`nobodyuser`	`guest`	This value must be the same value returned by a call to `getCallerPrincipal.getName` from the EJB tier when an unauthenticated caller in the web tier invokes an EJB method.
`orb.host`	`localhost`	Hostname of the machine running the RI
`orb.port`	`3699`	The port number the RI is listening to for service requests.
`password`	`j2ee`	Associated password for the user.
`password1`	`${derby.passwd}`	Set this to the password for the `jdbcDB1` resource.
`password2`	`${derby.passwd}`	Set this to the password for the `jdbcDB2` resource.
`password3`	`${derby.passwd}`	Set this to the password for the `jdbcDBTimer` resource.
`password_ri`	`j2ee_ri`	Associated password for the RI user.
`password_vi`	`j2ee_vi`	Associated password for the VI user.
`pathsep`	`:`	Users must set this property when running on Windows. The appropriate value on windows is a semicolon (`;`). If you are not running on Windows leave this property set to its default value of colon (`:`).
`porting.ts.deploy2.class.1`	`com.sun.ts.lib.implementation.sun.javaee.SunRIDeployment2`	Vendor's Implementation VI of `com.sun.ts.lib.porting.TSDeploymentInterface`
`porting.ts.HttpsURLConnection.class.1`	`com.sun.ts.lib.implementation.sun.javaee.SunRIHttpsURLConnection`	VI of `com.sun.ts.lib.porting.TSHttpsURLConnectionI`
`porting.ts.jms.class.1`	`com.sun.ts.lib.implementation.sun.javaee.SunRIJMSAdmin`	VI of `com.sun.ts.lib.porting.TSJMSAdminInterface.`
`porting.ts.login.class.1`	`com.sun.ts.lib.implementation.sun.javaee.SunRILoginContext`	VI of `com.sun.ts.lib.porting.TSLoginContextInterfa`

TABLE B–1 Modifiable `<TS_HOME>/bin/ts.jte` Properties and Default Values *(Continued)*

Property	Default Value	Description
`porting.ts.url.class.1`	`com.sun.ts.lib.implementation.sun.common.SunRIURL`	VI of `com.sun.ts.lib.porting.TSURLInterface.`
`queryManagerURL`	`""`	Standard connection property for querying. For `RegistryServer` it is: `queryManagerURL = http localhost 8080R`
`registryURL`	`""`	Standard connection property for publishing. For `RegistryServer` it is: `registryURL = http localhost 8080Regis`
`report.dir`	`/tmp/JTreport`	The directory used to store JavaTest summary reports of test results.
`ri.admin`	`${javaee.home.ri}/bin/asadmin`	The Java Platform, Enterprise Edition RI `asadmin` command.
`ri.admin.host`	`${orb.host.ri}`	The Java Platform, Enterprise Edition RI host.
`ri.admin.passwd`	There is no default setting for this property.	The Java Platform, Enterprise Edition RI `asadmin` user password.
`ri.admin.port`	`4848`	The Java Platform, Enterprise Edition RI port.
`ri.admin.user`	`admin`	The Java Platform, Enterprise Edition RI `asadmin` user ID.
`ri.asenv.loc`	`${javaee.home.ri}/config`	Location of `asenv.conf` or `asenv.bat`.
`ri.domain`	`${ri.domain.dir}/${ri.domain.name}`	The Java Platform, Enterprise Edition RI domain path being used.
`ri.domain.dir`	`${javaee.home.ri}/domains`	Points to where your domains are installed.
`ri.domain.name`	`domain1`	The Java Platform, Enterprise Edition RI domain being used.
`ri.imq.share.lib`	`${javaee.home.ri}/imq/lib`	Shared library directory for IMQ.
`ri.imqbin.loc`	`${javaee.home.ri}/imq/bin`	Location of the IMQ `bin` directory.
`ri.java.endorsed.dirs`	`${ri.lib}/endorsed`	Endorsed directory used by RI.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description
ri.jvm.options		Java options needed by the Java Platform, Enterprise Edition RI.
ri.lib	${javaee.home.ri}/lib	Library directory for other Java Platform, Enterprise Edition RI JARs.
ri.server	server	The Java Platform, Enterprise Edition RI server instance being used.
rmiiiop.http.server.host	${orb.host}	RMI server host name.
rmiiiop.http.server.port	10000	RMI server port.
s1as.admin	${javaee.home}/bin/asadmin	The SJSAS asadmin command.
s1as.admin.host	${orb.host}	The SJSAS host.
s1as.admin.passwd	There is no default setting for this property.	The SJSAS asadmin user password.
s1as.admin.port	4848	The SJSAS port.
s1as.admin.user	admin	The SJSAS asadmin user ID.
s1as.asenv.loc	${javaee.home}/config	Location of asenv.conf or asenv.bat.
s1as.classpathsuffix	${javaee.home}/lib/tsprovider.jar	The classpath suffix of the RI when being used as the App Server under test.
s1as.db.ext.dirs		The extension directory for DB Type 2 drivers. Currently, this only needs to be set when using DB2 else it should be empty.
s1as.domain	${s1as.domain.dir}/${s1as.domain.name}	The SJSAS domain path being used.
s1as.domain.dir	${javaee.home}/domains	Points to where your domains are installed.
s1as.domain.name	domain1	The SJSAS domain being used.
s1as.imq.share.lib	${javaee.home}/imq/lib	Shared library directory for IMQ.
s1as.imqbin.loc	${javaee.home}/imq/bin	Location of the IMQ bin directory.
s1as.java.endorsed.dirs	${s1as.lib}/endorsed	Endorsed directory used by SJSAS.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description
`s1as.jvm.options`	`-XX\:MaxPermSize=128m: \ -Doracle.jdbc.J2EE13Compliant=true`	Java options needed by SJSAS.
`s1as.lib`	`${javaee.home}/lib`	Library directory for other Java Platform, Enterprise Edition RI JARs.
`s1as.pe.jmsServer`	`imqbroker`	Name of the JMS server the RI/PE.
`s1as.se.jmsServer`	`imqbroker`	Name of the JMS server for SE/EE
`s1as.server`	`server`	The SJSAS server instance being used.
`s1as.targets`	`${s1as.server}`	Instances to deploy tests Supports multiple instances. For example: `s1as.targetsserver=server-1`
`securedWebServicePort`	`1044`	Must be set to run `secbasicssl` and `csiv2` tests. Set this property with your application server's secured `webservice` port.
`securedWebServicePort.2`	`1045`	Points to the secured `webservice` port in Sun's Reference Implementation (RI).
`ServletClientThreads`	`2`	The `ServletClientThreads` property configures the number of threads used by the client for the `SingleThreadModel` servlet test. If the container implementation supports pooling of `SingleThreadModel` serlvets, set the value of `ServletClientThreads` to twice the value of the default servlet instance pool size. If the container implementation only maintains a single instance of a `SingleTheadModel` servlet, leave the default value of `2`.
`sjsas.cmp.backend`	`derby`	Backend to use when we are using `java2db` with the CMP tests.
`sjsas.cts.timer.resource`	`derby`	Backend to use when we are using `java2db` with the CMP tests for the `jdbcDBTimer` resource.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description
sjsas.das.orb.host	${orb.host}	ORB host name for the DAS.
sjsas.das.orb.port	3700	ORB port number for the DAS.
sjsas.das.securedWebServicePort	1043	HTTPS listener port for the DAS.
sjsas.das.webServerPort	8000	HTTP listener port for the DAS.
sjsas.env.type	das	CTS test configuration. Possible values are: ■ das (for PE or DAS) ■ remote (for remote instance) ■ cluster (for cluster configuration not yet supported)
sjsas.instance.config.dir	config	Configuration directory used by the instance being tested. ■ For PE/DAS = config ■ For remote instance = config
sjsas.master.password	changeit	Used to create a node agent only applicable to EE. Defaults to changeit. This can be changed at EE install time.
sjsas.node.agent.dir.name	n	The name of the node agent directory to use. This value will be used on Windows only and ignored on non-Windows platforms. The default is n meaning the create node agent command will pass the agentdir argument with a value of s1as.applicationRootsjsas.node.agent.dir.nam
sjsas.nodeagent.name	node-agent-1	Name of node agent used by the remote instance.
sjsas.nodeinstance.name	server-1	Name of the remote instance.
transport_protocol	smtp	Must be set to a valid protocol, i.e, SMTP.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description
ts.classpath	${ts.home}/lib/tspackager.jar: \ ${javaee.home.ri}/javadb/lib/derby.jar: \ ${ts.home}/lib/tsharness.jar: \ ${ts.home}/lib/cts.jar: \ ${ts.home}/lib/sigtest.jar: \ ${ts.run.classpath}:${ts.run.classpath.ri}: \ ${ts.home}/lib/javatest.jar: \ ${ts.home}/lib/jdom.jar:${ts.home}/lib/dom4j.jar:\ ${ts.home}/tools/ant/lib/ant.jar: \ ${ts.home}/lib/commons-httpclient-3.0.jar: \ ${ts.home}/lib/commons-logging-1.0.4.jar: \ ${ts.home}/lib/htmlunit-1.7.jar	Classes used to build the CTS tests.
ts.display	:0.0	Location to display CTS output on UNIX.
ts.harness.classpath	${ts.home}/lib/jaxb-api.jar: \ ${ts.home}/lib/jaxb-libs.jar: \ ${ts.home}/lib/jaxb-impl.jar: \ ${ts.home}/lib/jaxb-xjc.jar: \ ${ts.home}/lib/tsharness.jar: \ ${ts.home}/lib/cts.jar: \ ${ts.home}/lib/sigtest.jar: \ ${ts.run.classpath}:${ts.run.classpath.ri}: \ ${ts.home}/lib/javatest.jar: \ ${ts.home}/lib/jdom.jar: \ ${ts.home}/lib/dom4j.jar: \ ${ts.home}/tools/ant/lib/ant.jar	Classes required by JavaTest.
ts.lib.classpath	${ts.home}/lib/tspackager.jar: \ ${javaee.home.ri}/javadb/lib/derby.jar: \ ${ts.harness.classpath}	Classes used to build cts.jar.
ts.run.classpath	${ts.home}/lib/oasis-regrep.jar: \ ${ts.home}/lib/omar-common.jar: \ ${s1as.lib}/dom.jar:${s1as.lib}/xalan.jar: \ ${javaee.classes}:${ts.home}/lib/tsharness.jar: \ ${ts.home}/lib/cts.jar	Classpath required by the vendor appclient container.
ts.run.classpath.ri	${javaee.classes.ri}: \ ${ri.lib}/riinterceptors.jar: \ ${ts.home}/lib/tsharness.jar: \ ${ts.home}/lib/cts.jar	Classpath required by Java Platform, Enterprise Edition RI appclient container.
tz	US/Eastern	Your local time zone. For valid values, consult your operating system documentation.

TABLE B–1 Modifiable <TS_HOME>/bin/ts.jte Properties and Default Values *(Continued)*

Property	Default Value	Description
user	j2ee	User defined to exercise role mapping feature.
user1	${derby.user}	Set this to the user for the jdbcDB1 resource.
user2	${derby.user}	Set this to the user for the jdbcDB2 resource.
user3	${derby.user}	Set this to the user for the jdbcDBTimer resource.
user_ri	j2ee_ri	Username for the reference implementation used in interop tests.
user_vi	j2ee_vi	Username for the vendor implementation used in interop tests.
webServerHost	${orb.host}	Hostname for the Vendor's Java Platform, Enterprise Edition Web Server.
webServerHost.2	${orb.host.ri}	Hostname for the Java Platform, Enterprise Edition RI Web Server.
webServerPort	8001	Port number of the Vendor's Java Platform, Enterprise Edition Web Server.
webServerPort.2	8002	Port number of the Java Platform, Enterprise Edition RI Web Server.
work.dir	/tmp/JTwork	The directory used to store JavaTest test results and test information.
wsdlRepository1	${harness.temp.directory}/wsdlRepository1	Location to publish final WSDL files when using file URL publishing for Vendor Java Platform, Enterprise Edition implementation.
wsdlRepository2	${harness.temp.directory}/wsdlRepository2	Location to publish final WSDL files when using file URL publishing for Sun RI.

APPENDIX C

Analyzing CSIv2 Test Logs

This appendix explains how to analyze the CSIv2 test logs. The topics included in this appendix are as follows:

C.1 Overview

This section describes the deployment scenarios under which the CSIv2 interoperability tests are executed. Each scenario uses two servers. The label "Licensee Java Platform, Enterprise Edition Server" refers to the server under test. The label "Java EE 6 RI Server" refers to the reference server against which compatibility will be verified.

There are three primary deployment scenarios, each of which is described in greater detail in sections that follow:

- "C.1.1 Application Client-to-EJB Scenarios" on page 166
- "C.1.2 EJB-to-EJB Test Scenarios" on page 167
- "C.1.3 Web Client-to-EJB Scenarios" on page 168

Although these scenarios often involve more than one invocation, the invocation of primary interest is the final invocation, which is always between servers.

In every deployment scenario, an EAR file, `rionly.ear`, is deployed on the server that is running the Java EE 6 RI. This EAR file contains a Log Bean and a Control Bean, which are used to facilitate the collection of results.

When the server that is running the Java EE 6 RI is started, a set of logging interceptors are registered, allowing the test code to collect protocol-level information. Note that this interceptor is only installed on the server that is running the Java EE 6 RI. The Java EE 6 CTS does not require the Licensee Java Platform, Enterprise Edition Server to support CORBA Portable Interceptors.

Each scenario is executed in both a "`forward`" and a "`reverse`" direction:

- In the `forward` direction, the client component of the invocation of primary interest is on the Licensee Java Platform, Enterprise Edition Server, and the server component is on the server that is running the Java EE 6 RI. In this direction, information about the CSIv2 request message is collected and analyzed.
- In the `reverse` direction, the client component of the invocation of primary interest is on the server that is running the Java EE 6 RI, and the server component is on the Java Platform, Enterprise Edition Licensee server. In this direction, information about the published IOR and the CSIv2 response message is collected and analyzed.

"C.1.1 Application Client-to-EJB Scenarios" on page 166, "C.1.2 EJB-to-EJB Test Scenarios" on page 167, and "C.1.3 Web Client-to-EJB Scenarios" on page 168 show the components that are present in both the `forward` and `reverse` directions.

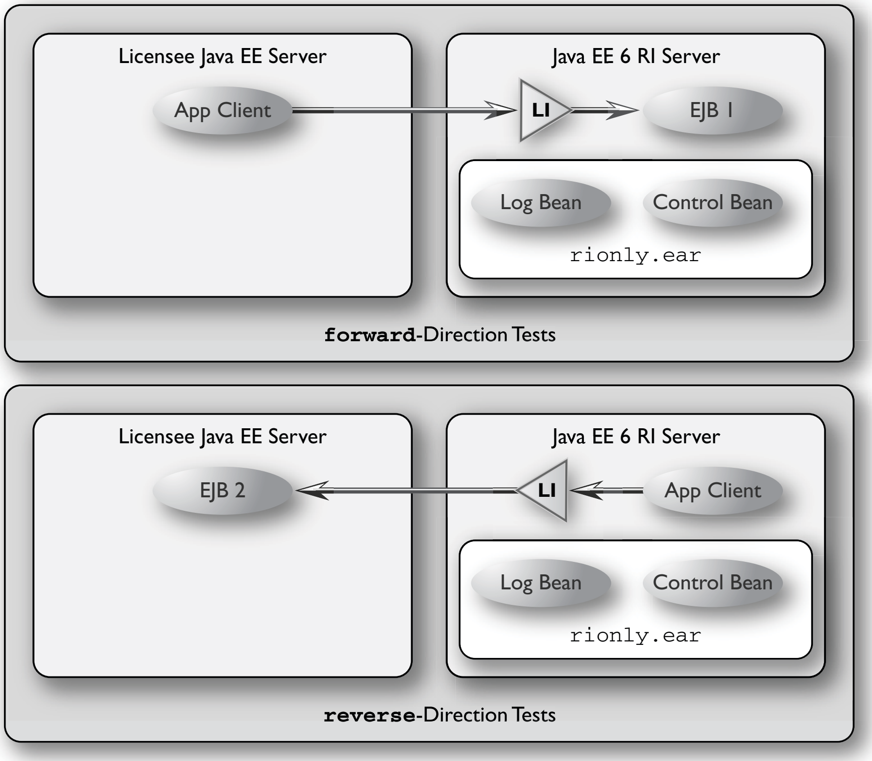
Whenever an EJB is mentioned in the following scenarios, it is important to realize that all possible combinations of Stateful Session Bean and CMP Entity Bean are used. For example, on an EJB-to-EJB invocation, we test Session Bean-to-Session Bean, Session Bean-to-Entity Bean, Entity Bean-to-Session Bean, and Entity Bean-to-Entity Bean. Each combination forms a new test.

By understanding these deployment scenarios, you can have a deeper understanding of how the tests work, which will make it easier to determine why a particular test is not passing.

C.1.1 Application Client-to-EJB Scenarios

Each application client-to-EJB scenario includes an application client and an EJB, as shown in Figure C–1. In the `forward` direction, the application client is deployed on the Licensee Java Platform, Enterprise Edition server and the EJB is deployed on the server that is running the Java EE 6 RI. In the `reverse` direction, the application client is deployed on the server that is running the Java EE 6 RI, and the EJB is deployed on the Licensee Java Platform, Enterprise Edition Server. The Logging Interceptor (LI) is deployed on the server that is running the Java EE 6 RI.

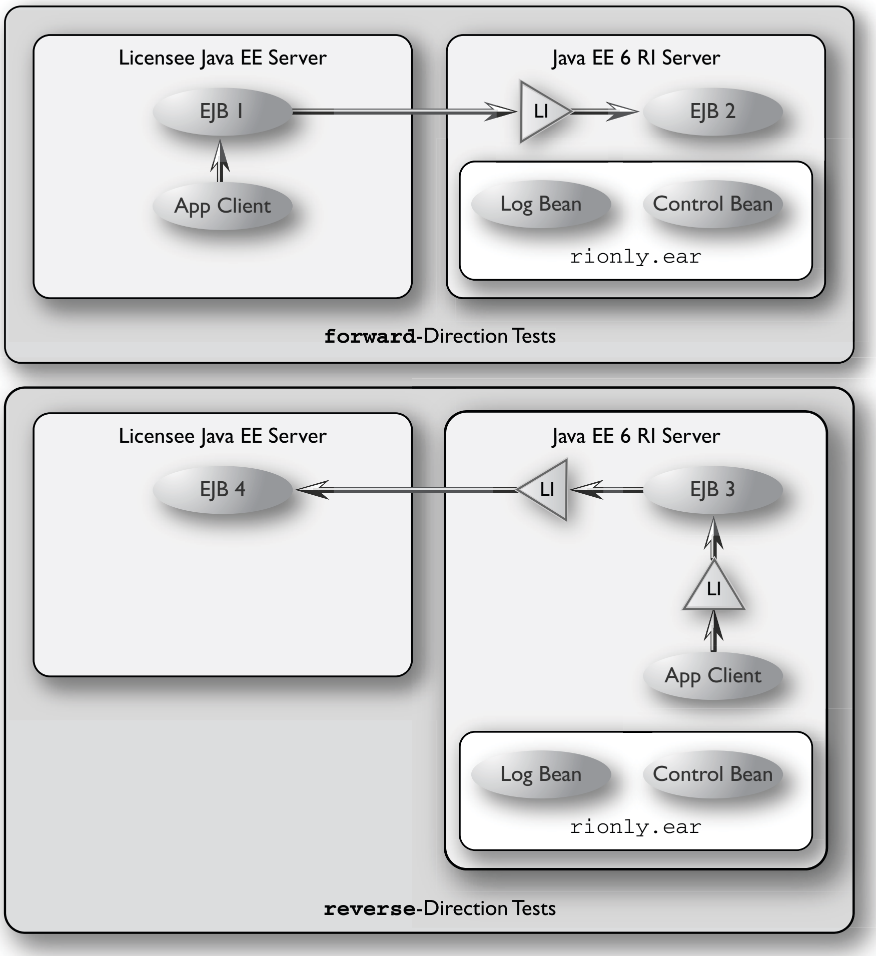
FIGURE C–1 Application Client-to-EJB Test Scenario



C.1.2 EJB-to-EJB Test Scenarios

Each of the EJB-to-EJB scenarios includes an application client and two enterprise beans, as shown in Figure C–2. In the `forward` direction, the application client and EJB1 are deployed on the Licensee Java Platform, Enterprise Edition Server. EJB2 is deployed on the server that is running the Java EE 6 RI. In the `reverse` direction, the Application Client and EJB3 are deployed on the server that is running the Java EE 6 RI. EJB4 is deployed on the Licensee Java Platform, Enterprise Edition Server. The Logging Interceptor (LI) is deployed on the server that is running the Java EE 6 RI.

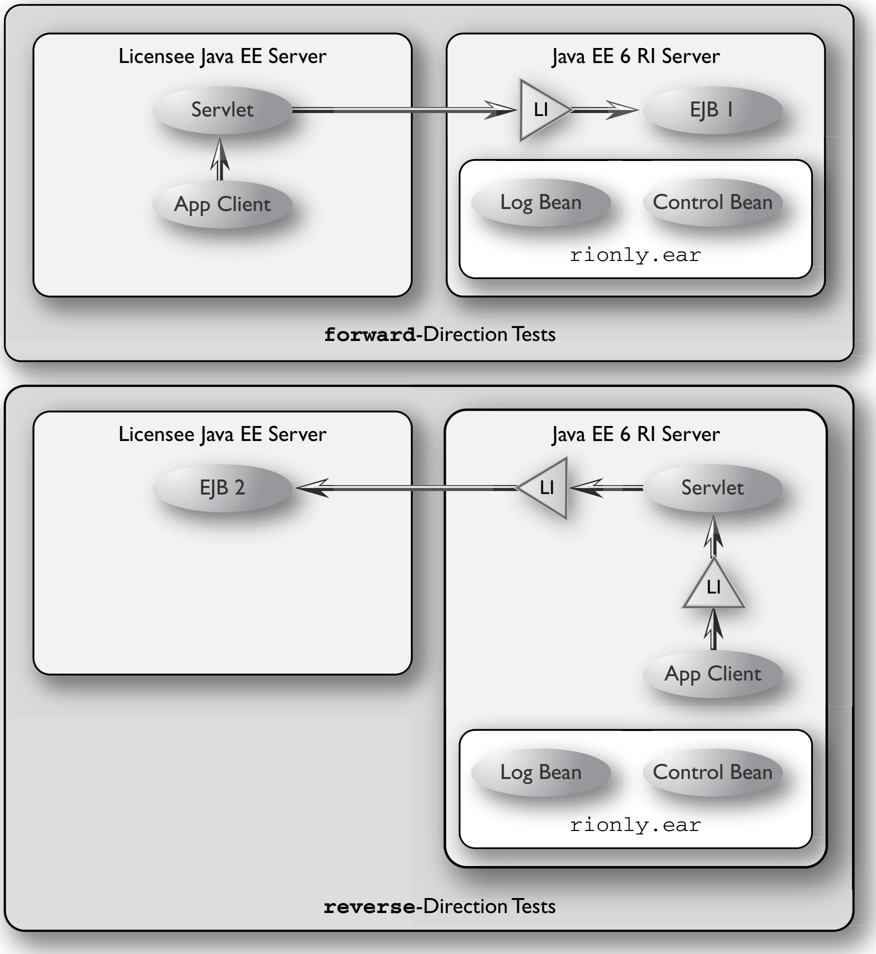
FIGURE C–2 EJB-to-EJB Test Scenario



C.1.3 Web Client-to-EJB Scenarios

Each Web client-to-EJB scenarios includes an application client, a servlet, and an EJB, as shown in Figure C–3. In the forward direction, the application client and the Web client are deployed on the Licensee Java Platform, Enterprise Edition Server. The EJB is deployed on the server that is running the Java EE 6 RI. In the reverse direction, the application client and the Web client are deployed on the server that is running the Java EE 6 RI. The EJB is deployed on the Licensee Java Platform, Enterprise Edition Server.

FIGURE C–3 Web-to-EJB Test Scenario



C.2 CSIv2 Test Directory Naming Conventions

The CSIv2 test directories are named according to the configuration that they represent. All tests are located in subdirectories of the `<TS_HOME>/src/com/sun/ts/tests/interop/csiv2` directory.

The CSIv2 test directories use the following naming conventions:

<orig>_<prot>_<auth>_<assertion>

Where:

- *<orig>* is the origin of the invocations:
  - `ac` — Application client
  - `ew` — EJB or Web client
- *<prot>* is the transport protection for the invocation:
  - `ssln` — No SSL transport protection
  - `ssl` — SSL transport protection
- *<auth>* is the authentication settings for the deployed beans:
  - `sslr_upn` — SSL authentication Required, No Username/Password authentication
  - `ssln_upr` — No SSL authentication, Username/Password authentication required
  - `ssln_upn` — Neither SSL authentication nor Username/Password authentication
- *<assertion>* is the type of identity assertion:
  - `noid` — No identity assertion
  - `noid_a` — Negative test case for no identity assertion
  - `ccid` — Certificate chain assertion
  - `upid` — Username/Password assertion
  - `anonid` — Assertion of anonymous identity

C.3 CSIv2 Test Directory Structure

The directory structure for the CSIv2 tests begins at the `tests/interop/csiv2` directory. The `/common` subdirectory contains code that is common to all CSIv2 tests. The other subdirectories each indicate different deployment settings. Each subdirectory has a `/forward` and a `/reverse` subdirectory.

Tests in the `/forward` subdirectory are run in the forward direction only (for example, the application client runs in the Licensee Java Platform, Enterprise Edition server, and a call is made to the Java EE 6 RI server). Tests in the `/reverse` subdirectory are run in the reverse direction only (for example, the application client runs in the server that is running the Java EE 6 RI, and a call is made to the Licensee Java Platform, Enterprise Edition server); for example:

```
/tests
   /interop
      /csiv2
         /common
         /ac_ssl_sslr_upn_noid
            /forward
            /reverse
         /ac_ssl_ssln_upr_noid
            /forward
            /reverse
            ...
```

C.4 Naming Conventions for CSIv2 Test Names

The CSIv2 test names are structured as follows:

*<dirname>*_*<client-component>*_*<server-component>*_*<testid>*`[_`*<direction>*`]`

Where:

- *<dirname>* is the directory name of the test, under `/tests/interop/csiv2`; for example:

  `ew_ssl_ssln_upn_anonid`
- *<client-component>* is the type of client component:
  - `sb` — session bean
  - `eb` — entity bean
  - `wb` — servlet
- *<server-component>* is the type of server component:
  - `sb` — session bean
  - `eb` — entity bean
- *<testid>* is the test ID; for example, `testid3`.
- *<direction>* is the direction of the test. The direction is omitted if forward, or reverse if in the reverse direction. For these tests, forward means from licensee server to the Java EE 6 RI server, and reverse means from the Java EE 6 RI server to the licensee server. In other words, the application client runs in the licensee's container in the forward direction.

Sample test application names for EJB-to-EJB tests include the following:

- `ew_ssl_ssln_upn_anonid_sb_sb_testid3`
- `ew_ssl_ssln_upn_anonid_sb_eb_testid3`
- `ew_ssl_ssln_upn_anonid_eb_sb_testid3_reverse`
- `ew_ssl_ssln_upn_anonid_eb_eb_testid3_reverse`

C.5 Debugging CSIv2 Test Failures

The CSIv2 test infrastructure provides two areas from which to obtain debugging output:

- Java EE 6 CTS clients, beans, and servlets
- Java EE 6 CTS CSIv2 interceptors

The sections that follow explain how to enable/disable logging to help you debug CSIv2 test failures.

C.5.1 Debugging CTS Clients, Beans, and Servlets

The first area in which you can enable logging is Java EE 6 CTS clients, beans, and servlets. If you have done any debugging in other technology areas in the Java EE 6 CTS test suite, you are likely to be familiar with enabling and using logging to obtain additional information with which you can debug test problems. This kind of debugging output is enabled by setting the `harness.log.traceflag` property to "true" in the `<TS_HOME>/bin/ts.jte` file. See Appendix B, "`ts.jte` Modifiable Properties," for a description of this property.

C.5.2 Debugging CTS CSIv2 Interceptors

The second area in which you can enable logging is Java EE 6 CTS CSIv2 interceptors, including Logging Interceptor Factory, Server Interceptor, and Client Interceptor. These three entities are configured during the CSIv2 test setup, which is described in "5.4.20 CSIv2 Test Setup" on page 91, by executing the `enable.csiv2` Ant task. During that configuration step, the following lines are added to the `<javaee.home.ri>/domains/domain1/config/logging.properties` file:

```
com.sun.ts.tests.interop.csiv2.common.LoggingSecRequestInterceptorFactory.level=INFO
com.sun.ts.tests.interop.csiv2.common.LoggingSecClientRequestInterceptor.level=INFO
com.sun.ts.tests.interop.csiv2.common.LoggingSecServerRequestInterceptor.level=INFO
```

These properties control the logging levels of the CSIv2 interceptors. By default, "INFO" level logging is enabled, which yields only minimal output in the `server.log` file.

▼ To Increase the Logging Level

1 **Stop the Java EE 6 RI.**

2 **Edit the file `<javaee.home.ri>/domains/domain1/config/logging.properties` and set the logging level to "FINE" for the three CSIv2 interceptors.**

3 **Restart the Java EE 6 RI.**

From this point on, an increased amount of logging output from the Java EE 6 CTS CSIv2 logging interceptors will be written to the `server.log` file.

▼ To Reset the Logging Level

1 **Stop the Java EE 6 RI.**

2 **Edit the file `<javaee.home.ri>/domains/domain1/config/logging.properties` and set the logging level back to "INFO" for the three CSIv2 interceptors.**

3 **Restart the Java EE 6 RI.**

Note – Execution of the `disable.csiv2` Ant target, which is explained in "5.4.20 CSIv2 Test Setup" on page 91, will remove the three properities from the `<javaee.home.ri>/domains/domain1/config/logging.properties` file.

C.6 Examining Test Logs

▼ To Examine the Test Logs

1. **The first thing you will notice is the direction in which the test is running:**

   ```
   LocalSessionBean (VI) ====> RemoteSessionBean (RI)
   ```

   VI-to-RI indicates that the test is running in the forward direction; RI-to-VI indicates that the test is running in the reverse direction.

2. **The test direction is followed by a trace that outlines the path of execution (for example, from a local session bean to a remote session bean, etc.)**

   ```
   INVOKING java:comp/env/ejb/LocalSession...
      SVR: Initialize remote logging
      SVR: CSIv2SessionBean.ejbCreateInvoke()
      SVR: Initialize remote logging
      SVR: CSIv2SessionBean.invoke()
      SVR: Entering CSIv2TestLogicImpl.invoke()
      SVR: INVOKING java:comp/env/ejb/RemoteSession...
      SVR: Initialize remote logging
      SVR: CSIv2SessionBean.ejbCreateInvoke()
      SVR: Initialize remote logging
      SVR: CSIv2SessionBean.invoke()
      SVR: Entering CSIv2TestLogicImpl.invoke()
      SVR: Exiting CSIv2TestLogicImpl.invoke()
      SVR: Exiting CSIv2TestLogicImpl.invoke()
   ```

   The CSIv2 tests maintain a log during the invocation. The log is in XML format, and is organized to match the flow of test execution.

   By examining the contents of the log, you can trace the test execution and see what happened in the test. See "C.7 CSIv2 Log Elements" on page 176 for a detailed description of the DTD elements that make up the CSIv2 log.

3. **Output validation results follow the log.**

   In the forward direction, the tests validate the request (see the EstablishContext message). In the reverse direction, the tests validate the IOR that the Licensee's Java EE 6 server publishes for the EJB and the response (see the `CompleteEstablishContext` message or the `ContextError` message). See Appendix E, "Interoperable Object Reference Definitions," for a list of the IORs that the test validation code and the test strategy descriptions reference.

The test output shows you what it is being validated for each test, and indicates the exact section of the log that is being analyzed. See the Example C–1 example, below.

4 **Next, the test output indicates what matched and what mismatched.**

Lines that start with the plus sign (+) indicate matches. Lines that start with "`MISMATCH:`" indicate mismatches. Lines that start with neither are informational messages.

`MISMATCH` messages indicate what was expected. The log tells you what was received. See the Example C–2example, below.

The reverse direction tests validate that the IOR that is published by the Licensee Java Platform, Enterprise Edition Server matches the expected result. The CSIv2 log will represent the values collected for `target_supports`, `target_requires`, and other CSIv2 IOR structures as decimal integers. In accordance with the CSIv2 specification, these values represent a bitmask. To determine the meaning of the bits that have been set in the bitmask, refer to page 16-64 of the CSIv2 specification, which can be found at the following location:

`http://cgi.omg.org/pub/csiv2-ftf/csiv2-031401.pdf`

Example C–1 Sample Validation Log

```
-------------------------------------------
Validating EJBRemote IOR...
   Validating the following IOR against IOR.4:
-------------------------------------------
<ior>
   <port>44139</port>
   <stateful>false</stateful>
   <compound-sec-mech>
   <target-requires>0</target-requires>
   <ior-transport-mech>
   <null-trans/>
   </ior-transport-mech>
   <ior-as-context>
   <target-supports>0</target-supports>
   <target-requires>0</target-requires>
   <client-authentication-mech></client-authentication-mech>
<target-name></target-name>
   </ior-as-context>
   <ior-sas-context>
   <target-supports>1024</target-supports>
   <target-requires>0</target-requires>
   <supported-naming-mechanism>0606678102010101</supported-
   naming-mechanism>
   <supported-identity-types>15</supported-identity-types>
   </ior-sas-context>
   </compound-sec-mech>
   </ior>
-------------------------------------------
Testing CompoundSecMech 1 of 1...
   Testing Naming Mechanisms 1 of 1...
      + This naming mechanism matches IOR.4
      + At least one naming mechanism matched IOR.4.
      + This CompoundSecMech matches IOR.4
      + At least one compound sec mech matched IOR.4.
EJBRemote IOR Valid.
```

Example C–2 Sample Mismatch Log

```
-------------------------------------------
Validating EJBHome IOR...
   Validating the following IOR against IOR.3:
   -------------------------------------------
<ior>
   <port>0</port>
   <stateful>false</stateful>
   <compound-sec-mech>
   <target-requires>70</target-requires>
   <ior-transport-mech>
   <tls-trans>
   <target-supports>38</target-supports>
   <target-requires>6</target-requires>
   <trans-addr>
   <host-name>129.148.71.198</host-name>
   <port>0</port>
   </trans-addr>
   </tls-trans>
   </ior-transport-mech>
   <ior-as-context>
   <target-supports>64</target-supports>
   <target-requires>64</target-requires>
   <client-authentication-mech></client-authentication-mech>
   <target-name></target-name>
   </ior-as-context>
   <ior-sas-context>
   <target-supports>1024</target-supports>
   <target-requires>0</target-requires>
   <supported-identity-types>15</supported-identity-types>
   </ior-sas-context>
   </compound-sec-mech>
   </ior>
-------------------------------------------
Testing CompoundSecMech 1 of 1...
   MISMATCH: Mismatch on target requires.
   Testing Transport Address 1 of 1...
   MISMATCH: Mismatch on port.
   MISMATCH: This transport address does not match IOR.3.
   MISMATCH: None of the transport address matched IOR.3.
   MISMATCH: Mismatch on IOR Transport Mech
   MISMATCH: Mismatch on AS Context
   MISMATCH: None of the naming mechs matched IOR.3.
   MISMATCH: Mismatch on SAS Context
   MISMATCH: This CompoundSecMech does not match IOR.3
   MISMATCH: None of the compound sec mechs matched IOR.3.
EJBHome IOR Invalid.
The following test output contains both matches and mismatches.
-------------------------------------------
Skipping IOR validation.
   Validating EJBHome and EJBRemote invocation request...
   Validating EJBHome Invocation Request...
   Validating the following invocation:
```

```
-------------------------------------------
<client>
   <server-interceptor>
   <operation>createInvoke</operation>
   <req-svc-context present="true">
   <establish-context>
   <client-context-id>0</client-context-id>
   <identity-token>
   <anonymous/>
   </identity-token>
   <client-auth-token></client-auth-token>
   <authz-token-count>0</authz-token-count>
   </establish-context>
   </req-svc-context>
   <ssl-used>false</ssl-used>
   <transport-client-principals>
   </transport-client-principals>
   <server>
   <invocation-principal>guest</invocation-principal>
   </server>
   <reply-svc-context present="true">
   <complete-establish-context>
   <client-context-id>0</client-context-id>
   <context-stateful>false</context-stateful>
   <final-context-token></final-context-token>
   </complete-establish-context>
   </reply-svc-context>
   </server-interceptor>
   </client>
-------------------------------------------
+ Match: Transport client principals absent, as expected.
+ Match: SAS Client principal present.
MISMATCH: Identity Token Type is invalid.  Expecting one of
the following:
   * ITTX509CertChain
   * ITTDistinguishedName
Found:
   * ITTAnonymous
MISMATCH: Mismatched SAS Identity Token Type.
EJBHome Invocation Request Invalid.
```

C.7 CSIv2 Log Elements

The CSIv2 log is stored in an XML format. This section describes the CSIv2 log elements. By understanding what these elements mean, you can use the log contents that are output from each test as a debugging aid. The CSIv2 log can be found in the CTS test source code, in the following location:

```
src/com/sun/ts/tests/interop/csiv2/common/parser/csiv2log.dtd
```

This section includes the following topics:

- "C.7.1 Key Elements in the CSIv2 Log" on page 177
- "C.7.2 Key Elements in the Server Interceptor Log" on page 177

- "C.7.3 Key Elements in the Client Interceptor Log" on page 178
- "C.7.4 Key Elements in an IOR Log" on page 179
- "C.7.5 Comprehensive List of All CSIv2 Log Elements" on page 180

C.7.1 Key Elements in the CSIv2 Log

The key elements of a CSIv2 log include the `<ejb-home>` and `<ejb-remote>` elements. These elements, in turn, contain the log information for the EJB home and remote interfaces.

EXAMPLE C–3 CSIv2 Log Elements

```
<csiv2log>
   <ejb-home>
      <client>
         <client_interceptor> | <server_interceptor>
         </client_interceptor> | </server_interceptor>
      </client>
   </ejb-home>
   <ejb-remote>
      <client>
         <client_interceptor> | <server_interceptor>
         </client_interceptor> | </server_interceptor>
      </client>
   </ejb-remote>
</csiv2log>
```

- The `<ejb-home>` element contains the `<client>` tag, which indicates that the test component is the client in an invocation and `<client_interceptor>` or `<server_interceptor>`, based on reverse or forward tests.
- The `<ejb-remote>` element contains a similar set of elements as the `<ejb-home>` element.

During forward testing (from VI to RI) using a simple scenario, such as an application client directly invoking an EJB, only the `<client_interceptor>` is logged. Conversely, during reverse testing (from RI to VI) using a simple scenario, the `<server_interceptor>` is logged.

For a complex scenario, such as an application client invoking an EJB, which in turn invokes another EJB, both client and server interceptor elements are logged. Other complex scenarios could log multiple client and server interceptors.

C.7.2 Key Elements in the Server Interceptor Log

The server interceptor element includes the `<req_svc_context>`, `<ssl_used>`, `<transport_client_principals>`, `<server>`, and `<reply_svc_context>` elements.

EXAMPLE C–4 Server Interceptor Log Elements

```
<server_interceptor>
   <req_svc_context> ... </req_svc_context>
   <ssl_used> true | false </ssl_used>
```

EXAMPLE C–4 Server Interceptor Log Elements *(Continued)*

```
      <transport_client_principals> ... </transport_client_principals>
     <server> ... </server>
     <reply_svc_context> ... </reply_svc_context>
</server_interceptor>
```

Service contexts provide a means of passing service-specific information as part of IIOP message headers.

These elements contain the following information:

- The `<req_svc_context>` element contains the service context information for the request.
- The `<ssl_used>` element indicates whether the transport is protected with SSL or not.
- The `<transport_client_principals>` element contains the principal used by the container for authentication at the SSL level.
- The `<server>` element logs the invocation principal if the request reaches the other end.
- The `<reply_svc_context>` element contains the service context information for the reply.

C.7.3 Key Elements in the Client Interceptor Log

The client interceptor element includes the `<req_svc_context>`, `<ssl_used>`, `<ior>`, `<server>`, `<location_forward>`, and `<reply_svc_context>` elements.

EXAMPLE C–5 Client Interceptor Log Elements

```
<client_interceptor>
     <req_svc_context> ... </req_svc_context>
     <ssl_used> true | false </ssl_used>
     <ior> ... </ior>
     <server> ... </server>
     <location_forward> ... </location_forward>
     <reply_svc_context> ... </reply_svc_context>
</client_interceptor>
```

Service contexts provide a means of passing service-specific information as part of IIOP message headers.

The client interceptor elements contain the following information:

- The `<req_svc_context>` element contains the service context information for the request.
- The `<ssl_used>` element indicates whether the transport is protected with SSL or not.
- The `<ior>` element contains the Interoperable Object Reference, which describes security policies of an EJB component.
- The `<location_forward>` element is "`true`" if the Client Security Service (CSS) received a location forward in response to a request. If this is the case, the client will establish a confidential connection with the new address.

EXAMPLE C–5 Client Interceptor Log Elements *(Continued)*

A `true` value also indicates that the log will contain another client interceptor element. The test validation mechanism will ignore client interceptor elements that end in a location forward.

- The `<reply_svc_context>` element contains the service context information for the reply.

C.7.4 Key Elements in an IOR Log

An IOR includes the `<port>` and `<stateful>` elements and the `<compound_sec_mech>` structure. The `<compound_sec_mech>` structure contains the `<target_requires>`, `<ior_transport_mech>`, `<ior_as_context>`, and `<ior_sas_context>` elements.

EXAMPLE C–6 IOR Log Elements

```
<ior>
   <port> ... </port>
   <stateful> true | false </stateful>
   <compound_sec_mech>
      <target_requires> ... </target_requires>
      <ior_transport_mech> ... </ior_transport_mech>
      <ior_as_context> ... </ior_as_context>
      <ior_sas_context> ... </ior_sas_context>
   <compound_sec_mech>
</ior>
```

These elements contain the following information:

- The `<port>` element can contain a zero or a nonzero number.

  A nonzero port number indicates that the target supports unprotected IIOP invocations at the specified port number.

  A zero port number indicates that the target supports protected IIOP invocations only.
- The `<stateful>` element is `true` if the target supports the establishment of stateful or reusable contexts.
- The `<compound_sec_mech>` structure describes support in the target for a compound security mechanism that may include security functionality that is realized in the transport layer and/or security functionality above the transport layer.
- The `<target_requires>` element designates a required outcome that shall be satisfied by one or more supporting (but not requiring) layers.
- The `<ior_transport_mech>` element describes the security functionality that is implemented in the transport layer.
- The `<ior_as_context>` element describes the client authentication functionality that the target expects to be layered above the transport layer in the service context.
- The `<ior_sas_context>` element describes the target's identity assertion support or support of authorization attributes that are delivered in the service context.

C.7.5 Comprehensive List of All CSIv2 Log Elements

Table C–1 provides a comprehensive list of all CSIv2 log elements.

TABLE C–1 CSIv2 Log Elements

Element	Description
`<csiv2-log>`	Root XML node. Contains 1 or more `<assertion>` elements.
`<assertion>`	Contains information relevant to a single test assertion. Contains a "name" attribute and an `<invocation>` element.
`<invocation>`	Indicates an invocation was started from a client component to a server component. We analyze invocation information for both an EJB Home (`<ejb-home>`) and an EJB Remote (`<ejb-remote>`) invocation.
`<ejb-home>` / `<ejb-remote>`	Separates the EJB Home from the EJB Remote invocation information. Both elements contain a single `<client>` element.
`<client>`	Indicates that this component is a client in an invocation. Contains a `<reply>` element and either a `<client-interceptor>`, `<server-interceptor>`, or a `<server>` element. The invocation determines which interceptor is to be invoked based on the whether the invoking component is acting as a client or as a server. For example, if an EJB acts as a server to an invocation, then the server-interceptor will be invoked.
`<reply>`	Indicates a reply in an invocation. Can be either `<create-exception>`, or `<other-exception>`.

TABLE C–1 CSIv2 Log Elements *(Continued)*

Element	Description
`<client-interceptor>`	Indicates that the client interceptor was invoked. This will happen when the Java EE 6 Reference Implementation is a client of some invocation. The following information is collected by the client interceptor: 1. `<operation>` 2. `<req-svc-context>` 3. `<ssl-used>` 4. `<ior>` 5. Either `<server-interceptor>` or `<server>` 6. `<location-forward>` 7. `<reply-svc-context>` 8. Possibly another `<client-interceptor>` element
`<operation>`	The name of the operation just invoked.
`<req-svc-context>`	Request service context information. This will contain either an `<establish-context>` message, or an `<invalid-message>`.
`<establish-context>`	Information collected from the CSIv2 `EstablishContext` message. Collects the following information: 1. `<client-context-id>` 2. `<identity-token>`, one of: `<absent>`, `<anonymous>`, `<principal-name>`, `<certificate-chain>`, `<distinguished-name>`, `<unknown-type>` 3. `<client-auth-token>` - Client authentication token 4. `<authz-token-count>` - Number of authorization tokens sent
`<invalid-message>`	Indicates that an invalid message (one that was not expected) was sent in either the request or the reply. A details attribute will indicate why the message was invalid, or the type of message that was received.
`<ssl-used>`	True if SSL will be or was used for this invocation.

TABLE C–1 CSIv2 Log Elements *(Continued)*

Element	Description

`<ior>`	IOR information. This is a description of the IOR that the server published for the EJB, from the client's perspective. Collects the following information:
	1. `<port>`
	2. `<stateful>`
	3. `<compound-sec-mech>`
	`<target-requires>`
	`<ior-transport-mech>` — one of the following:
	a. `<tls-trans>`
	`<target-supports>`
	`<target-requires>`
	`<trans-addr>`
	`<host-name>`
	`<port>`
	b. `<null-trans>`
	c. `<other-trans>`
	`<ior-as-context>`
	`<target-supports>`
	`<target-requires>`
	`<client-authentication-mech>`
	`<target-name>`
	`<ior-sas-context>`
	`<target-supports>`
	`<target-requires>`
	`<supported-naming-mechanism>`
	`<supported-identity-types>`

`<location-forward>`	If `true`, this request ended in a location forward, in which case we will expect to see another client interception later down the road. The test validation will ignore all client interceptor elements that end in a location forward, in case target servers do load balancing, or something of the sort.
`<reply-svc-context>`	Reply service context information. This will contain either a `<complete-establish-context>`, a `<context-error>`, or an `<invalid-message>` element.

TABLE C–1 CSIv2 Log Elements *(Continued)*

Element	Description
`<complete-establish-context>`	Information collected from the CSIv2 `CompleteEstablishContext` message. Collects the following information: 1. `<client-context-id>` 2. `<context-stateful>` 3. `<final-context-token>`
`<context-error>`	Information collected from the CSIv2 `ContextError` message. Collects the following information: 1. `<client-context-id>` 2. `<major-status>` 3. `<minor-status>` 4. `<error-token>`
`<server-interceptor>`	Indicates that the server interceptor was invoked. This will happen when the Java EE 6 Reference Implementation is a server of some invocation. The following information is collected by the server interceptor: 1. `<operation>` 2. `<req-svc-context>` 3. `<ssl-used>` 4. `<transport-client-principals>` 5. `<server>`, if the request makes it to the server. 6. `<reply-svc-context>`
`<transport-client-principals>`	Collection of all transport client principals for this invocation, if it was an SSL invocation.
`<server>`	Indicates that the server bean was invoked. This will happen on every successful invocation. The following information is collected on the server bean: 1. `<invocation-principal>` 2. `<invocation>`, if another invocation is made.
`<invocation-principal>`	The value of `EJBContext.getCallerPrincipal().getName()`.

C.8 IORs and Associated CSIv2 Tests

The Table C–2 table, below, provides additional information about the CSIv2 tests:

- The test ids that are associated with each IOR
- The identity assertion type that is tested by each test
- The name of the directory in which the tests reside

TABLE C–2 IORs and Associated CSIv2 Tests

IOR	Test ID	Identity Assertion Type	Directory Name
0	0	NA	`ac_ssl_sslr_upn_noid`
1	2	NA	`ac_ssl_ssln_upr_noid`
	2a	NA	`ac_ssl_ssln_upr_noid_a`
3	3	`anonymous`	`ew_ssl_ssln_upn_anonid`
	3a	`upid`	`ew_ssl_ssln_upn_upid`
	3b	`ccid`	`ew_ssl_ssln_upn_ccid`
4	4	`anonymous`	`ew_ssln_ssln_upn_anonid`
	4a	`ccid`	`ew_ssln_ssln_upn_ccid`
	6	`upid`	`ew_ssln_ssln_upn_upid`
7	7	`upid`	`ew_ssl_sslr_upn_upid`
	7a	`ccid`	`ew_ssl_sslr_upn_ccid`
	8	`anonymous`	`ew_ssl_sslr_upn_anonid`

APPENDIX D

JASPIC Technology Notes and Files

The JASPIC technology tests are used to verify the compatibility of a licensee's implementation of the JASIC 1.0 specification. This appendix provides information about the following topics:

- "D.1 JASPIC 1.0 Technology Overview" on page 185
- "D.2 JASPIC TSSV Files" on page 186

D.1 JASPIC 1.0 Technology Overview

The JASPIC 1.0 specification (JSR 196) defines a service provider interface (SPI) by which authentication providers implementing message authentication mechanisms can be integrated in client and server message processing runtimes (or containers).

The Java EE 6 CTS uses a Test Suite SPI Verifier (TSSV) to verify whether the vendor's message processing runtimes invoke the right SPI in the right order.

TSSV includes test suite implementations of:

- `AuthConfigFactory`
- `AuthConfigProvider`
- `AuthConfig(Client & Server)`
- `AuthContext(client & Server)`
- `AuthenticationModules(Client & Server)`

TSSV gets loaded into vendor's message processing runtime using one of the following two ways as defined by the JSR196 specification:

- By defining a property in `JAVA_HOME/jre/lib/security/java.security` as follows: `authconfigprovider.factory=com.sun.ts.tests.jaspic.tssv.config.TSAuthConfigFactory`
- By calling `registerConfigProvider()` method in vendor's `AuthConfigFactory` with the following values:
  - Test Suite Provider ClassName

- Map of properties
- Message Layer (such as `SOAP` or `HttpServlet`)
- Application Context Identifier
- A description of the provider

Note – For the Java EE 6 CTS, we register more than one provider in vendor's message processing runtime.

In a typical test scenario (for each profile of Servlet or SOAP), an application is deployed into a vendor's runtime, and a client invokes the service. The message policies required for the secure invocations are built into TSSV implementations, and the runtime is analyzed to see whether it invokes the right SPIs at the right time.

TSSV uses Java logging APIs to log the client and server invocation into a log file (`TSSVLog.txt`), this log file is used by the TCK tests to validate actual logged runtime information against expected results to ensure that the runtime is compliant. The `jaspic_util_web.war` file contains the JASPIC log file processor, which writes output to the `TSSVLog.txt` file. The `TSSVLog.txt` file is put into the location defined by the `log.file.location` property in the `ts.jte` file.

D.2 JASPIC TSSV Files

The following sections describe the `tssv.jar`, `ProviderConfiguration.xml`, and `provider-configuration.dtd` files that are used by the JASPIC TCK tests.

D.2.1 tssv.jar file

The `tssv.jar` file contains classes necessary for populating your implementation with a CTS AuthConfigFactory (ACF) as well as information used to register CTS providers. The tssv.jar file contains the class files for the Test Suite SPI Verifier. The `tssv.jar` file classes need to be loaded by your implementation's runtime during startup.

D.2.2 `ProviderConfiguration.xml` file

The format of the `ProviderConfiguration.xml` file is a test suite-specific format. The file was designed to contain test provider information the test suite uses to populate the ACF with a list of providers for testing. The file needs to be copied to the location specified in the ts.jte file by the `provider.configuration.file` property. An edit to the `ProviderConfiguration.xml` file may be required for your implementation. The current application context Ids are generic and should work as is, but there could be some scenarios in which the application Context Ids may need to be altered.

The value of the `<app-context-id>` element in the `ProviderConfiguration.xml` file should reflect what your implementation will use for its internal representation of the application context identifier for a registered provider. Said differently, the test suite registers its providers with information from the `ProviderConfiguration.xml` file but every implementation is not guaranteed to use the application context identifier that is used in the call to register the configuration provider. This value of the `<app-context-id>` element corresponds to the `appContext` argument in the `AuthConfigFactory.registerConfigProvider()` API. The API documentation for this method indicates that the `appContext` argument may be used but is not guaranteed to be used.

The default `ProviderConfiguration.xml` file should work without modification but you may need to alter the value of the `<app-context-id>` element as previously described to accommodate the implementation under test. You need to find the correct application context identifier for your implementation.

You should enable two levels of logging output to get finer levels of debugging and tracing information than is turned on by default. This is done by setting the `traceflag` property in the `ts.jte` file and the HARNESS_DEBUG environment variable to "true". If both of these are set, application context identifier information should appear in the debug output(s).

D.2.3 provider-configuration.dtd file

The `provider-configuration.dtd` file is a DTD file that resides in the same directory as the `ProviderConfiguration.xml` file and describes the `ProviderConfiguration.xml` file. This file should *not* be edited.

APPENDIX E

Interoperable Object Reference Definitions

This appendix provides the expected published Interoperable Object References (IORs) for the CSIv2 interoperability tests. If, at test time, an IOR does not match the expected result, the test output will refer to one of these IORs by number. The test strategy descriptions attached to each reverse-direction CSIv2 test also reference these IORs.

This appendix contains listings for the following IORs:

E.1 IOR.0

EXAMPLE E–1 IOR.0

```
port=0
CompoundSecMechList {
   stateful = FALSE;
   mechanism_list = {
       CompoundSecMec {
           target_requires={Integrity, Confidentiality,
               EstablishTrustInClient};
           transport_mech = TAG_SSL_SEC_TRANS {
               target_supports = {Integrity, Confidentiality,
                   EstablishTrustInClient,
                   EstablishTrustInTarget};
               target_requires = {Integrity, Confidentiality,
                   EstablishTrustInClient};
               addresses = {
                   TransportAddress = {
                       host_name = x;
                       port = y;
                   };
```

EXAMPLE E–1 IOR.0 *(Continued)*

```
                };
            };
            as_context_mech = {
                target_supports = {};
                ...
            };
            sas_context_mech = {
                target_supports = {};
                ...
            };
        };
    };
};
```

E.2 IOR.1

EXAMPLE E–2 IOR.1

```
port=0
CompoundSecMechList {
    stateful = FALSE;
    mechanism_list = {
        CompoundSecMec {
            target_requires = {Integrity, Confidentiality,
                EstablishTrustInClient};
            transport_mech = TAG_SSL_SEC_TRANS {
                target_supports = {Integrity, Confidentiality,
                   EstablishTrustInTarget};
                target_requires = {Integrity, Confidentiality};
               addresses = {
                    TransportAddress {
                        host_name = x;
                        port = y;
                    };
                };
            };
            as_context_mech = {
                target_supports = {EstablishTrustInClient};
                target_requires = {EstablishTrustInClient};
                client_authentication_mech = GSSUP_OID;
                target_name = {GSSUP,"default"};
                ...
            };
            sas_context_mech = {
                target_supports = {};
                ...
            };
        };
    };
 };
```

E.3 IOR.3

EXAMPLE E–3 IOR.3

```
port=0
CompoundSecMechList {
    stateful = FALSE;
    mechanism_list = {
       CompoundSecMec {
           target_requires = {Integrity, Confidentiality};
           transport_mech = TAG_SSL_SEC_TRANS {
               target_supports = {Integrity, Confidentiality,
                  EstablishTrustInTarget};
               target_requires = {Integrity,
                  Confidentiality};
               addresses = {
                    TransportAddress {
                        host_name = x;
                        port = y;
                    };
                };
           };
           as_context_mech = {
               target_supports = {};
               ...
           };
           sas_context_mech = {
               target_requires = {};
               target_supports = {IdentityAssertion};
               ...
               supported_naming_mechanisms = {GSSUPMechOID};
               supported_identity_types = {ITTPrincipalName};
           };
       };
   };
};
```

E.4 IOR.4

EXAMPLE E–4 IOR.4

```
port=<nonzero-port-number>
CompoundSecMechList {
   stateful = FALSE;
   mechanism_list = {
       CompoundSecMec {
           target_requires = {};
           transport_mech = TAG_NULL_TAG;
           as_context_mech = {
               target_supports = {};
               ...
           };
           sas_context_mech = {
               target_requires = {};
```

EXAMPLE E–4 IOR.4 *(Continued)*

```
               target_supports = {IdentityAssertion};
               ...
               supported_naming_mechanisms = {GSSUPMechOID};
               supported_identity_types = {ITTPrincipalName};
           };
       };
   };
};
```

E.5 IOR.7

EXAMPLE E–5 IOR.7

```
port=0
CompoundSecMechList {
   stateful = FALSE;
   mechanism_list = {
       CompoundSecMec {
           target_requires = {Integrity, Confidentiality,
               EstablishTrustInClient};
           transport_mech = TAG_SSL_SEC_TRANS {
               target_supports = {Integrity, Confidentiality,
                   EstablishTrustInClient,
                   EstablishTrustInTarget};
                target_requires = {Integrity, Confidentiality,
                   EstablishTrustInClient};
                addresses = {
                     TransportAddress {
                         host_name = x;
                         port = y;
                     };
                 };
            };
            as_context_mech = {
                target_supports = {};
                ...
            };
            sas_context_mech = {
               target_requires = {};
               target_supports = {IdentityAssertion};
                ...
                supported_naming_mechanisms = {GSSUPMechOID};
                    supported_identity_types = {ITTPrincipalName};
            };
        };
    };
};
```

◆ ◆ ◆ APPENDIX F

Configuring Your Backend Database

This appendix explains how to configure a backend database to use with a Java Platform, Enterprise Edition server being tested against the Java EE 6 CTS.

The topics included in this appendix are as follows:

F.1 Overview

All Java Platform, Enterprise Edition servers tested against the Java EE 6 CTS must be configured with a database and JDBC 3.0-compliant drivers. Note that the Java Platform, Enterprise Edition RI includes Sun's implementation of the JavaDB database.

To perform interoperability testing, you need to configure two Java Platform, Enterprise Edition servers and two databases, one of which must be the Java Platform, Enterprise Edition RI with the bundled JavaDB database. See "5.3.1 Java Platform, Enterprise Edition Server Configuration Scenarios" on page 52 for more information.

For the purposes of Java EE 6 CTS testing, all database configuration properties required by the CTS are made in the `<TS_HOME>/bin/ts.jte` file. The CTS `init.`*<database>* Ant target uses the properties you set in `ts.jte` to generate one or more SQL statement files that are in turn used create and populate database tables and configure procedures required by the CTS.

The database configuration process comprises four general steps:

1. Set database-related properties in the `<TS_HOME>/bin/ts.jte` file.
2. Configure your Java Platform, Enterprise Edition server implementation for your database and for CTS.

3. Start your database.
4. Run the `init.`*<database>* Ant target to initialize your database for CTS.

The procedure for configuring your Java Platform, Enterprise Edition server for your database is described in "5.3 Configuring a Java Platform, Enterprise Edition Server" on page 51. The final step, initializing your database for CTS by running `init.`*<database>* target, is explained more in the next section.

F.2 The `init.`*<database>* Ant Target

Before your Java Platform, Enterprise Edition server database can be tested against the Java EE 6 CTS, the database must be initialized for CTS by means of the Ant `init.`*<database>* target. For example, the `init.javadb` Ant task is used to initialize the JavaDB database for CTS.

This Ant target references database properties in `ts.jte` and database-specific DDL and DML files to generate SQL statement files that are read by the Java EE 6 CTS when you start the test suite. The DDL and DML files are described later in this appendix, in "F.4 Database DDL and DML Files" on page 197.

The Java EE 6 CTS 5 includes the following database-specific Ant targets:

- `init.cloudscape`
- `init.db2`
- `init.oracle`
- `init.oracleDD`
- `init.oracleInet`
- `init.javadb`
- `init.sybase`
- `init.sybaseInet`
- `init.mssqlserver`
- `init.mssqlserverInet`
- `init.mssqlserverDD`

Each Ant target uses a database-specific JDBC driver to configure a backend for a specific database; for example, OracleInet/Oracle Inet driver; OracleDD/Oracle DataDirect driver. All of these targets are configured in the `<TS_HOME>/bin/setup.xml` file.

F.3 Database Properties in `ts.jte`

Listed below are the names and descriptions for the database properties you need to set for CTS testing. Refer to Appendix B, "`ts.jte` Modifiable Properties," for a complete listing of the `ts.jte` file.

Note that some properties take the form <*property*>`.ri`. In all cases, properties with an `.ri` suffix are used for interoperability testing only. In such cases, the <*property*> value applies to the Java Platform, Enterprise Edition VI server (the server you want to test) and the <*property*>`.ri` value applies to the Sun Java Platform, Enterprise Edition RI server. For example:

```
db.dml.file=<VI_DML_filename>
db.dml.file.ri=<RI_DML_filename>
```

The <*property*>`.ri` properties are only used in two-server configurations; that is, when you are performing interoperability tests.

TABLE F–1 `ts.jte` Database Properties

Property	Description
<*database*>`.classes`	`CLASSPATH` to JDBC driver classes.
<*database*>`.dataSource`	DataSource driver.
<*database*>`.dbName`	Database Name.
<*database*>`.driver`	DriverManager driver.
<*database*>`.password`	User password configured.
<*database*>`.poolName`	Name of pool configured in SunOne Appserver (do not change!).
<*database*>`.port`	Database Server port.
<*database*>`.properties`	Additional properties required by the defined data source for each driver configuration in `ts.jte`. You should not need to modify this property.
<*database*>`.server`	Database Server.
<*database*>`.url`	URL for the CTS database; the `dbName`, `server`, and `port` properties are automatically substituted in to build the correct URL. You should never need to modify this property.
<*database*>`.user`	User ID configured.

TABLE F–1 `ts.jte` Database Properties *(Continued)*

Property	Description
`create.cmp.tables`	When set to `false`, the application server is responsible for creating CMP tables at deployment of the EJB/EAR. When set to `true`, `init.`<*datbase*> creates the tables used by CMP EJBs. The SQL for the CMP tables are contained in <*TS_HOME*>`/`<*datbase*>`/sql/`<*database*>`.ddl.cmp.sql` and <*TS_HOME*>`/`<*datbase*>`/sql/`<*database*>`.ddl.interop.sql`.
`create.interop.tables.only`	This option is only needed when you are configuring the database used by the Java Platform, Enterprise Edition RI for interoperability testing. When set to `true`, only the BMP and CMP tables required for interoperability testing are created when `init.`<*datbase*> is invoked. Note that for the current version of the Java Platform, Enterprise Edition RI, you must set `create.cmp.tables=true` when you set `create.interop.tables.only=true`.
`db.dml.file`	Tells `init.`<*datbase*> which DML file to use for the VI database; for example, `db.dml.file=${javadb.dml.file}`.
`db.dml.file.ri`	Tells `init.`<*datbase*> which DML file to use for the RI database; for example, `db.dml.file=${javadb.dml.file}`.
`jdbc.lib.class.path`	Used by the <*database*>`.classes` properties to point to the location of the JDBC drivers.
`jdbc.poolName`	Configures the connection pool that will be tested in this CTS test run; for example, `jdbc.poolName=${javadb.poolName}`. This needs to set this when running against the S1AS/RI if using a database other than JavaDB.
`password1`	Password for the JDBC/DB1 resource; for example, `password1=${javadb.passwd}`.
`password2`	Password for the JDBC/DB2 resource; for example, `password2=${javadb.passwd}`.
`password3`	Password for the JDBC/DBTimer resource; for example, `password3=${javadb.passwd}`.
`user1`	User name for the JDBC/DB1 resource; for example, `user1=${javadb.user}`.

TABLE F–1 `ts.jte` Database Properties *(Continued)*

Property	Description
`user2`	User name for the JDBC/DB2 resource; for example, `user2=${javadb.user}`.
`user3`	User name for the JDBC/DBTimer resource; for example, `user3=${javadb.user}`.

F.4 Database DDL and DML Files

For each supported database type, the Java EE 6 CTS includes a set of DDL and DML files in subdirectories off the `<TS_HOME>/sql` directory. The `config.vi` and `config.ri` targets use two `ts.jte` properties, `db.dml.file` and `db.dml.file.ri` (interop only), to determine the database type, and hence which database-specific DML files to copy as `<TS_HOME>/bin/tssql.stmt` and `tssql.stmt.ri` (for interop) files.

The `tssql.stmt` and `tssql.stmt.ri` files contain directives for configuring and populating database tables as required by the CTS tests, and for defining any required primary or foreign key constraints and database-specific conmand line terminators.

In addition to the database-specific DML files, the Java EE 6 CTS includes database-specific DDL files, also in subdirectories off `<TS_HOME>/sql`. These DDL files are used by the `init.`<*database*> target to create and drop database tables and procedures required by the CTS.

The SQL statements in the `tssql.stmt` and `tssql.stmt.ri` files are read as requested by individual CTS tests, which use the statements to locate required DML files.

The DDL and DML files are as follows:

- <*database*>`.ddl.sql` — DDL for BMP, Session Beans
- <*database*>`.ddl.sprocs.sql` — DDL for creating stored procedures
- <*database*>`.ddl.cmp.sql` — DDL for CMP Entity Beans
- <*database*>`.ddl.interop.sql` — DDL for interop tests
- <*database*>`.dml.sql` — DML used during test runs

Each DDL command in each `<TS_HOME>/sql/<database>` is terminated with an ending delimiter. The delimiter for each <*database*> is defined in the `<TS_HOME>/bin/xml/initdb.xml` file. If your configuration requires the use of a database other than the databases that `initdb.xml` currently supports, you may modify `initdb.xml` to include a target to configure the database that you are using.

An example of the syntax for a database target in `initdb.xml` is shown below:

```
<target name="init.sybase" >
<antcall target="configure.backend" >
      <param name="db.driver" value="${sybase.driver}"/>
      <param name="db.url" value="${sybase.url}"/>
      <param name="db.user" value="${sybase.user}"/>
      <param name="db.password" value="${sybase.passwd}"/>
      <param name="db.classpath" value="${sybase.classes}"/>
      <param name="db.delimiter" value="!"/>
      <param name="db.name" value="sybase" />
    </antcall>
</target>
```

The <*database*>`.name` property should be added to your `ts.jte` file. The `db.name` property is the name of a subdirectory in `<TS_HOME>/sql`. After updating `initdb.xml`, you invoke the new target with:

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

F.5 CMP Table Creation

If the application server under test does not provide an option to automatically create tables used by CMP Entity EJBs, the needed SQL is provided in `<TS_HOME>/sql/`<*database*>`/`<*database*>`.cmp.sql`.

Setting the `ts.jte` property `create.cmp.tables=true` instructs the `init.`<*database*> target to create the tables defined in the `<TS_HOME>/sql/`<*database*>`/`<*database*>`.cmp.sql` file.

If you set `create.cmp.tables=false` in the `ts.jte` file, it is expected that you will create the necessary CMP tables at deployment time.

◆ ◆ ◆ APPENDIX G

EJBQL Schema

The EJB-QL tests perform queries against a CMP 2.0 abstract persistence model that you deploy before you start the test runs.

Section 11.3.5, "EJB QL and SQL," in the EJB 2.0 Specification contains a sample mapping that shows how the Java Platform, Enterprise Edition RI translates EJB QL to SQL, which helps to clarify the EJB QL semantics.

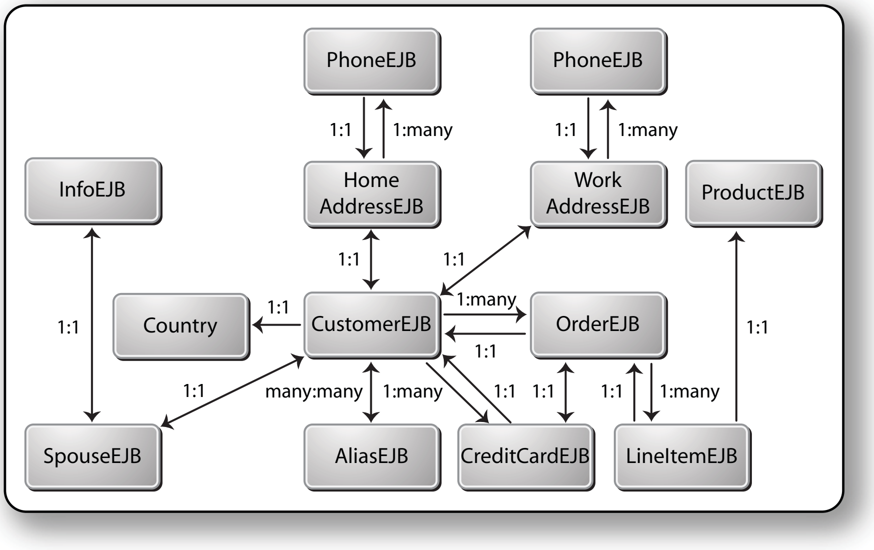
This appendix includes information about the following topics:

- "G.1 Persistence Schema Relationships" on page 199
- "G.2 SQL Statements for CMP 1.1 Finders" on page 201

G.1 Persistence Schema Relationships

The figure, Figure G–1, below, contains detailed information about the persistence schema relationships.

FIGURE G–1 Persistence Schema Relationships



AliasEJB

```
String id;(pk)(cmp)
String alias;(cmp)
Customer customerNoop;(cmr)
Collection customersNoop;(cmr)
Collection customers;(cmr)
```

Country — DVC

```
String name;(cmp)
String code;(cmp)
```

AddressEJB

```
String id;(pk)(cmp)
String street;(cmp)
String city;(cmp)
String state;(cmp)
String zip;(cmp)
Collection phones;(cmr)
```

ProductEJB

```
String id;(pk)(cmp)
String name;(cmp)
float price;(cmp)
int quantity;(cmp)
long partNumber;(cmp)
```

CustomerEJB

```
String id;(pk)(cmp)
String name;(cmp)
Country country;(cmp)
AddressLocal home;(cmr)
AddressLocal work;(cmr)
Collection creditCards;(cmr)
Collection orders;(cmr)
Collection aliases;(cmr)
SpouseLocal spouse;(cmr)
```

CreditCardEJB

```
String id;(pk)(cmp)
String type;(cmp)
String expires;(cmp)
boolean approved;(cmp)
String number;(cmp)
OrderLocal order;(cmr)
CustomerLocal customer;(cmr)
double balance;(cmp)
```

InfoEJB

```
String id;(pk)(cmp)
String street;(cmp)
String city;(cmp)
String state;(cmp)
String zip;(cmp)
SpouseLocal spouse;(cmr)
```

OrderEJB

```
String id;(pk)(cmp)
float totalPrice;(cmp)
CustomerLocal customer;(cmr)
LineItemLocalsampleLineItem;(cmr)
Collection lineItems;(cmr)
CreditCardLocal creditCard;(cmr)
```

LineItemEJB

```
String id;(pk)(cmp)
int quantity;(cmp)
OrderLocal order;(cmr)
ProductLocal product;(cmr)
```

PhoneEJB

```
String id;(pk)(cmp)
String area:(cmp)
String number;(cmp)
AddressLocal  address;(cmr)
```

SpouseEJB

```
String id;(pk)(cmp)
String firstName;(cmp)
String maidenName;(cmp)
String lastName;(cmp)
String SocialSecurityNumber(cmp);
InfoLocal info;(cmr)
CustomerLocal customer;(cmr)
```

G.2 SQL Statements for CMP 1.1 Finders

Listed below are the SQL statements used for CMP 1.1 finders in:

- `ejb/ee/bb/entity/cmp/clientviewtest`
- `interop/ejb/entity/cmp/clientviewtest`
- `ejb/ee/bb/entity/cmp/complexpktest`
- `ejb/ee/tx/txECMPbean`

G.2.1 `ejb/ee/bb/entity/cmp/clientviewtest, interop/ejb/entity/cmp/clientviewtest`

```
<method-name>findWithinPrimaryKeyRange</method-name>
<sql>SELECT "KEY_ID" FROM "TestBeanEJBTable" WHERE "KEY_ID" BETWEEN ?1 AND ?2</sql>
<method-name>findWithinPriceRange</method-name>
```

```
<sql>SELECT "KEY_ID" FROM "TestBeanEJBTable" WHERE "PRICE" BETWEEN ?1 AND ?2</sql>
<method-name>findByName</method-name>
<sql>SELECT "KEY_ID" FROM "TestBeanEJBTable" WHERE "BRAND_NAME" = ?1</sql>
<method-name>findAllBeans</method-name>
<sql>SELECT "KEY_ID" FROM "TestBeanEJBTable"</sql>
<method-name>findByPrice</method-name>
<sql>SELECT "KEY_ID" FROM "TestBeanEJBTable" WHERE "PRICE" = ?1</sql>
<method-name>findByNameSingle</method-name>
<sql>SELECT "KEY_ID" FROM "TestBeanEJBTable" WHERE "BRAND_NAME" = ?1</sql>
```

G.2.2 ejb/ee/bb/entity/cmp/complexpktest

```
<method-name>findByPrice</method-name>
<sql>SELECT "BRAND_NAME", "ID" FROM "TestBeanEJBTable" WHERE "PRICE" = ?1</sql>
<method-name>findById</method-name>
<sql>SELECT "BRAND_NAME", "ID" FROM "TestBeanEJBTable" WHERE "ID" = ?1</sql>
<method-name>findByName</method-name>
<sql>SELECT  "BRAND_NAME", "ID" FROM "TestBeanEJBTable" WHERE "BRAND_NAME" = ?1</sql>
```

G.2.3 ejb/ee/tx/txECMPbean

```
<method-name>findByName</method-name>
<sql>SELECT "KEY_ID" FROM "TxECMPBeanEJBTable" WHERE "BRAND_NAME" = ?1</sql>
<method-name>findWithinPrimaryKeyRange</method-name>
<sql>SELECT "KEY_ID" FROM "TxECMPBeanEJBTable" WHERE "PRICE" BETWEEN ?1 AND ?2</sql>
<method-name>findByPrice</method-name>
<sql>SELECT "KEY_ID" FROM "TxECMPBeanEJBTable" WHERE "PRICE" = ?1</sql>
<method-name>findWithinPrimaryKeyRange</method-name>
<sql>SELECT "KEY_ID" FROM "TxECMPBeanEJBTable" WHERE "KEY_ID" BETWEEN ?1 AND ?2</sql>
```

APPENDIX H

Rebuilding the JAX-WS and JWS Tests

The JAX-WS 2.2 and JWS 2.0 specifications require 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 and JWS portions of the CTS require that you first rebuild them using your generation tools.

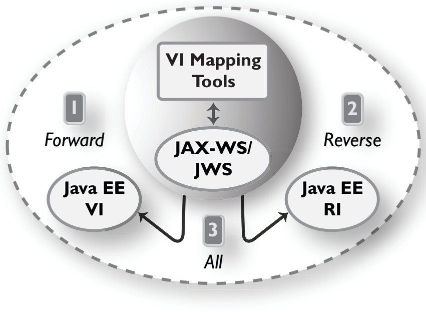
This appendix contains the following sections:

H.1 Overview

The set of prebuilt JAX-WS and JWS archives and classes that ship with the CTS were built using Sun Reference Implementation tools (`wsgen` and `wsimport`), and must be deployed and run against your Java Platform, Enterprise Edition implementation. 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 implementation.

FIGURE H–1 JAX-WS and JWS 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 Java Platform, Enterprise Edition implementation and the Sun Reference Implementation configured simultaneously. See "Configuring a Java Platform, Enterprise Edition RI and VI for Interop/Rebuildable Tests" on page 60 for more information.

H.2 Rebuilding the Tests Using the CTS Infrastructure

Instead of rebuilding and overwriting the prebuilt classes and archives for each test directory, the JAX-WS and JWS components of the CTS provide 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 CTS.

▼ To Rebuild the Tests Using the CTS Infrastructure

1 **Create your own version of the Ant tasks `WsImport` and `WsGen`.**

Documentation for these tasks can be found later in this appendix:

- "H.4 `wsgen` Reference" on page 208
- "H.5 `wsimport` Reference" on page 211

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 **Change to the appropriate test directory (`jaxws` or `jws`) and execute the following build target:**

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

You must do this separately for both the `jaxws` and `jws` test directories.

The clean and build targets may be executed in any subdirectory under `<TS_HOME>/src/com/sun/ts/tests/jaxws` or `<TS_HOME>/src/com/sun/ts/tests/jws` 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, the 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 original test client source code or the service endpoint implementation test source code may be altered in any way by a Vendor as part of the rebuild process.

▼ Example: To Rebuild a Single Test Directory

1 **Change to the `<TS_HOME>/src/com/sun/ts/tests/jaxws/ee/w2j/document/literal/httptest` directory.**

2 **Run `<TS_HOME>/tools/ant/bin/ant llc`.**

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

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

3 **Run `<TS_HOME>/tools/ant/bin/ant lld`.**

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

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

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

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

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

5 **Run `<TS_HOME>/tools/ant/bin/ant -Dbuild.vi=true llc`.**

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

```
$TS_HOME/tools/ant/bin/ant -Dbuild.vi=true llc

/var/tmp/jaxwstck/classes_vi_built/com/sun/ts/tests/jaxws/ee/w2j/
document/literal/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 `<TS_HOME>/tools/ant/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_`.

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

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

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

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

Once you have successfully built the archives using your implementation, proceed to "Configuring a Java Platform, Enterprise Edition RI and VI for Interop/Rebuildable Tests" on page 60 to learn how to deploy these archives and how to run the reverse tests.

H.3 Rebuilding the Tests Manually

When rebuilding the JAX-WS and JWS test classes, it is strongly recommended that you use the procedure described in "H.2 Rebuilding the Tests Using the CTS Infrastructure" on page 204. However, if you choose not to use the CTS infrastructure to rebuild the tests, you can use the following procedure to rebuild the classes manually.

▼ To Rebuild the Tests Manually

1. **Run your tools in each of the `jaxws` and `jws` test directories, under `<TS_HOME>/src/com/sun/ts/tests/jaxws` and `<TS_HOME>/src/com/sun/ts/tests/jws`, respectively.**

   Be 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 EAR files, prepeded with the string `vi_built_`.**

   These EAR files must follow all naming conventions as those found in the EARs built with the CTS infrastructure, and should reside in the same directory with the prebuilt EAR files under `<TS_HOME>/dist` directory.

4. **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 tests not being able to be built or run the prebuilt archives shipped with the CTS. The files you may need to modify are:**

   - XML files in `<TS_HOME>/bin/xml`; these files are used to generate the various EAR files.
   - Any `build.xml` file in `<TS_HOME>/src/com/sun/ts/tests/jaxws` or `<TS_HOME>/src/com/sun/ts/tests/jws`.

- The `<TS_HOME>/src/com/sun/ts/tests/jaxws/common/common.xml` and `<TS_HOME>/src/com/sun/ts/tests/jws/common/common.xml` files, which are the main build files used for the `jaxws` and `jws` build processes. These `common.xml` files contain all the Ant tasks specific to invoking the `jaxws` and `jws` tools.

Note – None of the original 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.

H.4 `wsgen` Reference

The `wsgen` tool generates Java EE 6 portable artifacts used in Java EE 6 Web services. The tool reads a Web service endpoint class and generates all the required artifacts for Web service deployment and invocation.

This section contains the following topics:

- "H.4.1 wsgen Syntax" on page 208
- "H.4.2 wsgen Ant Task" on page 209
- "H.4.3 wsgen Example" on page 211

H.4.1 `wsgen` Syntax

```
wsgen [options] <SEI>
```

TABLE H–1 `wsgen` Command Syntax

Option	Description
`-classpath <path>`	Specify where to find input class files.
`-cp <path>`	Same as `-classpath <path>`.
`-d <directory>`	Specify where to place generated output files.
`-extension`	Allow vendor extensions (functionality not specified by the specification). Use of extensions may result in applications that are not portable or may not interoperate with other implementations.
`-help`	Display help.
`-keep`	Keep generated files.
`-r <directory>`	Used only in conjunction with the `-wsdl` option. Specify where to place generated resource files such as WSDLs.

TABLE H–1 wsgen Command Syntax *(Continued)*

Option	Description
`-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"`

H.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 H–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
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 soap1.1or Xsoap1.2, default is soap1.1. Xsoap1.2is 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

TABLE H–2 wsgen Attributes and Elements *(Continued)*

Attribute	Description	Command Line
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 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, defined elsewhere in the build environment, and contains the list of classes required by the Java EE 6 tools.

H.4.3 wsgen Example

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

H.5 wsimport Reference

The `wsimport` tool generates Java EE 6 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.

This section contains the following topics:

- "H.5.1 wsimport Syntax" on page 212
- "H.5.2 wsimport Ant Task" on page 213
- "H.5.3 wsimport Examples" on page 216

H.5.1 wsimport Syntax

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

TABLE H–3 `wsimport` Command Syntax

Option	Description
`-d <directory>`	Specify where to place generated output files.
`-b <path>`	Specify external Java EE 6 or JAXB binding files (Each `<file>` must have its own `-b`).
`-B <jaxbOption>`	Pass this option to JAXB schema compiler
`-catalog`	Specify catalog file to resolve external entity references, it supports `TR9401`, `XCatalog`, and OASIS XML Catalog format. Please read the Catalog Support document (`http://jax-ws.java.net/2.2/docs/catalog-support.html`) 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.

TABLE H–3 wsimport Command Syntax *(Continued)*

Option	Description
`-verbose`	Output messages about what the compiler is doing.
`-version`	Print version information.
`-wsdllocation <location>`	`@WebService.wsdlLocation` and `@WebServiceClient.wsdlLocation` value.
`-target`	Generate code for the specified version of the Java EE 6 specification. For example, a value of 2.0 generates code that is compliant with the Java EE 6 2.0 Specification. The default value is 2.2.
`-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.
`-XdisableSSLHostnameVerification`	Disbales the SSL Hostname verification while fetching the wsdls.

Multiple Java EE 6 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 Java EE 6 and JAXB binding files can be found in the customization documentation (`https://jax-ws.dev.java.net/nonav/2.2/docs/customizations.html`).

H.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 H–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 Java EE 6 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

TABLE H–4 wsimport Attributes and Elements (Continued)

Attribute	Description	Command Line
wsdllocation	The WSDL URI passed through this option is used to set the value of @WebService.wsdlLocation and @WebServiceClient.wsdlLocation annotation elements on the generated SEI and Service interface	-wsdllocation
catalog	Specify catalog file to resolve external entity references, it supports TR9401, XCatalog, and OASIS XML Catalog format. Additionally, the Ant xmlcatalog type can be used to resolve entities. See the wsimport_catalog sample for more information.	-catalog
package	Specifies the target package	-p
target	Generate code for the specified version of the Java EE 6 specification. For example, a value of 2.0 generates code that is compliant with the Java EE 6 2.0 Specification. The default value is 2.2.	-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 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, defined elsewhere in the build environment, and contains the list of classes required by the Java EE 6 tools.

H.5.2.1 Nested Elements

`wsimport` supports the following nested element parameters.

binding

To specify more than one binding file at the same time, use a nested `<binding>` element, which has the same syntax as `<fileset>`. See the FileSet section in the Ant manual for more information.

arg

Additional command line arguments passed to the `wsimport` Ant task. For details about the syntax, see the arg section in the Ant manual. This nested element can be used to specify various options not natively supported in the `wsimport` Ant task. For example, currently there is no native support for the `-XdisableSSLHostnameVerification` command-line option for `wsimport`. This nested element can be used to pass –X command-line options directly, as done with –`XadditionalHeaders`. To use any of these features from the `wsimport` Ant task, you must specify the appropriate nested `<arg>` elements.

xjcarg

The usage of `xjcarg` is similar to that of the `<arg>` nested element, except that these arguments are passed directly to the XJC tool (JAXB Schema Compiler), which compiles the schema that the WSDL references. For details about the syntax, see the arg section in the Ant manual.

xmlcatalog

The xmlcatalog element is used to resolve entities when parsing schema documents.

H.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.

◆ ◆ ◆ APPENDIX I

Context Root Mapping Rules for Web Services Tests

The context root mapping rules that are described in this chapter apply to all of the web services test areas, including `jaxrpc`, `jaxws`, `jws`, `webservices`, `webservices12`, and `webservices13`.

I.1 Servlet-Based Web Service Endpoint Context Root Mapping

This section describes the context root mapping for servlet-based web services endpoints and clients. Since most of the Sun application runtime and Sun Web runtime deployment descriptor files have been removed in Java EE 6, the context root mapping for web archives in the Java EE 6 RI becomes the base name of the Web archive file without the file extension. For example, the context root for the archive `web-client.war` defaults to `web-client`.

This covers the mapping for all servlet-based web services endpoints and clients under theJava EE 6 CTS test trees [jaxrpc, jaxws, jws, webservices, webservices12, webservices13].

For example, for the `jaxws` test directory that is shown below, the context root mapping will be:

```
WSW2JDLHttpTest_web.war    --> context root: WSW2JDLHttpTest_web
WSW2JDLHttpTest_wsservlet_vehicle_web.war --> context root: WSW2JDLHttpTest_wsservlet_vehicle_web

% cd $TS_HOME/src/com/sun/ts/tests/jaxws/ee/w2j/document/literal/httptest
% $TS_HOME/tools/ant/bin/ant ld

[echo] WSW2JDLHttpTest.ear
[echo] WSW2JDLHttpTest_web.war
[echo] WSW2JDLHttpTest_web.war.sun-web.xml
[echo] WSW2JDLHttpTest_wsappclient_vehicle.ear
[echo] WSW2JDLHttpTest_wsappclient_vehicle_client.jar
[echo] WSW2JDLHttpTest_wsappclient_vehicle_client.jar.sun-application-client.xml
[echo] WSW2JDLHttpTest_wsejb_vehicle.ear
[echo] WSW2JDLHttpTest_wsejb_vehicle_client.jar
[echo] WSW2JDLHttpTest_wsejb_vehicle_client.jar.sun-application-client.xml
[echo] WSW2JDLHttpTest_wsejb_vehicle_ejb.jar
```

```
[echo] WSW2JDLHttpTest_wsejb_vehicle_ejb.jar.sun-ejb-jar.xml
[echo] WSW2JDLHttpTest_wsservlet_vehicle.ear
[echo] WSW2JDLHttpTest_wsservlet_vehicle_web.war
[echo] WSW2JDLHttpTest_wsservlet_vehicle_web.war.sun-web.xml
```

For example, for the `jaxrpc` test directory shown below, the context root mapping will be:

```
W2JREMarshallTest_web.war --> context root: W2JREMarshallTest_web
W2JREMarshallTest_jsp_vehicle_web.war --> context root: W2JREMarshallTest_jsp_vehicle_web
W2JREMarshallTest_servlet_vehicle_web.war --> context root: W2JREMarshallTest_servlet_vehicle_web

% cd $TS_HOME/src/com/sun/ts/tests/jaxrpc/ee/w2j/rpc/encoded/marshalltest
% $TS_HOME/tools/ant/bin/ant ld

[echo] W2JREMarshallTest.ear
[echo] W2JREMarshallTest_appclient_vehicle.ear
[echo] W2JREMarshallTest_appclient_vehicle_client.jar
[echo] W2JREMarshallTest_appclient_vehicle_client.jar.sun-application-client.xml
[echo] W2JREMarshallTest_ejb_vehicle.ear
[echo] W2JREMarshallTest_ejb_vehicle_client.jar
[echo] W2JREMarshallTest_ejb_vehicle_client.jar.sun-application-client.xml
[echo] W2JREMarshallTest_ejb_vehicle_ejb.jar
[echo] W2JREMarshallTest_ejb_vehicle_ejb.jar.sun-ejb-jar.xml
[echo] W2JREMarshallTest_jsp_vehicle.ear
[echo] W2JREMarshallTest_jsp_vehicle_web.war
[echo] W2JREMarshallTest_jsp_vehicle_web.war.sun-web.xml
[echo] W2JREMarshallTest_servlet_vehicle.ear
[echo] W2JREMarshallTest_servlet_vehicle_web.war
[echo] W2JREMarshallTest_servlet_vehicle_web.war.sun-web.xml
[echo] W2JREMarshallTest_web.war
[echo] W2JREMarshallTest_web.war.sun-web.xml
```

For Web archives, the context root mapping becomes the base name of the Web archive file minus the extension.

I.2 EJB-Based Web Service Endpoint Context Root Mapping

This section describes the context root mapping for EJB-based web services endpoints and clients. The context root mapping for EJB-based web services and clients is based on the following mapping rules that are used for the Java EE 6 RI.

The following algorithm describes the context root mapping rules that are used by the Java EE 6 Reference Implementation.

```
if sun-ejb-jar.xml deployment descriptor exists
    if <endpoint-address-uri> tag exists
        context root = value of <endpoint-address-uri>
    else
        if WebService.name annotation is specified on implementation bean
            context root = WSDL Service Name + / + WebService.name
        else
            context root = WSDL Service Name + / + Simple Bean Class Name
        endif
    endif
else
    if WebService.name annotation is specified on implementation bean
        context root = WSDL Service Name + / + WebService.name
    else
        context root = WSDL Service Name + / + Simple Bean Class Name
    endif
endif
```

For example, these are the context root mappings for the `webservices12/ejb/annotations` directory:

```
-------------            -----------------------------------
Test Directory           Context Root = <endpoint-address-uri>
-------------            -----------------------------------
WSEjbMultipleClientInjectionTest1    "WSEjbMultipleClientInjectionTest1/ejb"
WSEjbMultipleClientInjectionTest2    "WSEjbMultipleClientInjectionTest2/ejb"
WSEjbNoWebServiceRefInClientTest    "WSEjbNoWebServiceRefInClientTest/ejb"
WSEjbNoWebServiceRefInClientTest    "WSEjbNoWebServiceRefInClientTest/ejb"
WSEjbPortFieldInjectionTest        "WSEjbPortFieldInjectionTest/ejb"
WSEjbPortMethodInjectionTest    "WSEjbPortMethodInjectionTest/ejb"
WSEjbSOAPHandlersTest        "WSEjbSOAPHandlersTest/ejb"
WSEjbSOAPHandlersTest2        "WSEjbSOAPHandlersTest2"/ejb"
WSEjbWebServiceProviderTest        "WSEjbWebServiceProviderTest/ejb"
WSEjbWebServiceRefTest2        "WSEjbWebServiceRefTest2/ejb"
WSEjbAsyncTest            "WSEjbAsyncTest/ejb"
```

The following two test directories under the `webservices12/ejb/annotations` do not specify the `<endpoint-address-uri>` deployment tag or do not contain a Sun EJB JAR runtime deployment descriptor file. Because of this, the context root is calculated using the previously described formula. In both cases, the context root is calculated as (WSDL Service Name + / + Simple Bean Class Name).

```
-------------            -----------------------------------------------------
Test Directory           Context Root = <WSDL Service Name/Simple Bean Class Name>
-------------            -----------------------------------------------------
WSEjbWebServiceRefTest1        "WSEjbWebServiceRefTest1HelloService/HelloBean"
WSEjbWebServiceRefWithNoDDsTest    "WSEjbWSRefWithNoDDsTestHelloEJBService/WSEjbWSRefWithNoDDsTestHelloEJB"
```

The context root mappings for some, but not all, tests also exist in the DAT files under the `$TS_HOME/bin` directory. These include the `jaxrpc-url-props.dat`, `jaxws-url-props.dat`, `jws-url-props.dat`, and `webservices12-url-props.dat` files.

Licensees can use the previously described information in their porting implementation layer for web services.