Oracle®

Technology Compatibility Kit User's Guide for Java EE Connector Architecture 1.7

Release 1.7 for Technology Licensees

September 2017

ORACLE®

Technology Compatibility Kit User's Guide for Java EE Connector Architecture 1.7, Release 1.7 for Technology Licensees

Primary Author: Eric Jendrock

Contents

4 Setup and Configuration

5 Executing Tests

6 Rebuilding Tests and Debugging Test Problems

A Frequently Asked Questions

Preface

This guide describes how to install, configure, and run the Technology Compatibility Kit (TCK) that is used to test the Java EE Connector Architecture (JCA) technology.

The JCA TCK is a portable, configurable automated test suite that uses JavaTest harness version 4.4.1 to run the test suite and verify the compatibility of a licensee's implementation of the JCA 1.7 Specification.

Note: URLs have been included to help you locate resources quickly. These URLs are subject to change and are beyond the control of the authors of this guide.

Who Should Use This Book

This guide is for licensees of the JCA 1.7 technology to assist them in running the test suite that verifies compatibility of their implementation of the JCA 1.7 Specification.

Documentation Accessibility

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

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Before You Read This Book

Before reading this guide, you should be familiar with the Java programming language. *Java Programming Language*, *Java Language Specification Fourth Edition*, and *Java Virtual Machine Specification Second Edition* are good sources of information. You should also be familiar with the Java EE Connector Architecture 1.7 Specification, which can be found at http://jcp.org/en/jsr/detail?id=322.

Before running the tests in the JCA TCK, you should familiarize yourself with the JavaTest documentation that is included in the JCA TCK documentation bundle.

Typographic Conventions

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

Convention	Meaning	Example

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

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

Shell Prompts in Command Examples

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

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

1 Introduction

This chapter provides an overview of the principles that apply generally to all Technology Compatibility Kits (TCKs) and describes the Java EE Connector Architecture TCK (JCA TCK 1.7) (JSR 322). It also includes a high level listing of what is needed to get up and running with the JCA TCK.

1.1 Compatibility Testing

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

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

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

1.1.1 Why Compatibility Testing is Important

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

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

1.1.2 TCK Compatibility Rules

Compatibility criteria for all technology implementations are embodied in the TCK Compatibility Rules that apply to a specified technology. Each TCK tests for adherence to these Rules as described in Chapter 2, "Procedure for Java EE Connector Architecture (JCA) 1.7 Certification."

1.1.3 TCK Overview

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

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

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

1.1.4 Java Community Process (JCP) Program and Compatibility Testing

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

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

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

1.2 About the JCA TCK 1.7

The JCA TCK 1.7 is designed as a portable, configurable, automated test suite for verifying the compatibility of a licensee's implementation of JCA 1.7 Specification.

1.2.1 JCA TCK Specifications and Requirements

This section lists the applicable requirements and specifications.

- **Specification Requirements**: Software requirements for a JCA implementation are described in detail in the JCA 1.7 Specification. Links to the JCA specification and other product information can be found at http://jcp.org/aboutJava/communityprocess/final/jsr322/index.html.
- **JCA Version**: The JCA TCK 1.7 is based on the JCA Specification, Version 1.7.
- **Reference Runtime**: The designated Reference Runtime for conformance testing of implementations based upon the JCA Specification 1.7 is JCA 1.7 Reference Implementation. The JCA RI is available from Java Licensee Engineering (https://javapartner.oracle.com).

1.2.2 JCA TCK Components

The JCA TCK 1.7 includes the following components:

- **JavaTest harness** version 4.4.1 and related documentation, which is available in the JCA TCK documentation bundle
- **JCA TCK signature tests**, which check that all public APIs are supported and/or defined as specified in the JCA Version 1.7 implementation under test
- **API tests** for all of the packages that comprise the required class libraries for Java EE Connector Architecture 1.7
- **End-to-end tests**, which demonstrate compliance with the Java EE Connector Architecture 1.7 specification

1.2.3 JavaTest Harness

The JavaTest harness version 4.4.1 is a set of tools designed to run and manage test suites on different Java platforms. The JavaTest harness can be described as both a Java application and a set of compatibility testing tools. It can run tests on different kinds of Java platforms and it allows the results to be browsed online within the JavaTest GUI, or offline in the HTML reports that the JavaTest harness generates.

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

- Sequencing of tests, allowing them to be loaded and executed automatically
- Graphic user interface (GUI) for ease of use
- Automated reporting capability to minimize manual errors
- Failure analysis
- Test result auditing and auditable test specification framework
- Distributed testing environment support

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

1.2.4 TCK Compatibility Test Suite

The *test suite* is the collection of tests used by the JavaTest harness to test a particular technology implementation. In this case, it is the collection of tests used by the JCA TCK 1.7 to test a JCA 1.7 implementation. The tests are designed to verify that a licensee's runtime implementation of the technology complies with the appropriate specification. The individual tests correspond to assertions of the specification.

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

1.2.5 Exclude Lists

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

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

Note: In these instructions, variables in angle brackets need to be expanded for each platform. For example, `<TS_HOME>` becomes `$TS_HOME` on Solaris/Linux and `%TS_HOME%` on Windows. In addition, the forward slashes (`/`) used in all of the examples need to be replaced with backslashes (`\`) for Windows. Finally, be sure to use the appropriate separator for your operating system when specifying multiple path entries (`;` on Windows, `:` on Solaris/Linux).

Note: From time to time, updates to the Exclude List are made available on the Java Licensee Engineering (https://javapartner.oracle.com) Web site. You should always make sure you are using an up-to-date copy of the Exclude List before running the JCA TCK to verify your implementation.

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

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

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

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 Section 2.3.1, "Java EE Connector Architecture Version 1.7 TCK Test Appeals Steps."

1.2.6 JCA TCK Configuration

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

1.3 Getting Started With the JCA TCK

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

1. Make sure that the following software has been correctly installed on the system hosting the JavaTest harness:
   - Java SE 7

- Java EE 7 Web Profile RI or an implementation of the Java EE Connector Architecture 1.7 specification
- Java EE Connector Architecture TCK Version 1.7

Note: You should not run the standalone Java EE Connector Architecture TCK tests with an RI that is compliant with the Java EE 7 Reference Implementation (Classic Profile). If you plan to run with this configuration, you must pass the Java EE Connector Architecture tests that are bundled as part of the Java EE 7 platform's Compatibility Test Suite (CTS).

See the documentation for each of these software applications for installation instructions. See Chapter 3, "Installation," for instructions on installing the JCA TCK.

2. Set up the JCA TCK software.

   See Chapter 4, "Setup and Configuration," for details about the following steps.

   a. Set up your shell environment.

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

   c. Configure the JavaTest harness, if you are planning to run the TCK tests through the JavaTest GUI.

3. Test the JCA 1.7 implementation.

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

2

Procedure for Java EE Connector Architecture (JCA) 1.7 Certification

This chapter describes the compatibility testing procedure and compatibility requirements for Java EE Connector Architecture 1.7. This chapter contains the following sections:

2.1 Certification Overview

The certification process for Java EE Connector Architecture Version 1.7 consists of the following activities:

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

2.2 Compatibility Requirements

The compatibility requirements for Java EE Connector Architecture Version 1.7 consist of meeting the requirements set forth by the rules and associated definitions contained in this section.

2.2.1 Definitions

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

Table 2–1 Definitions

Term	Definition
API Definition Product	A Product for which the only Java class files contained in the product are those corresponding to the application programming interfaces defined by the Specifications, and which is intended only as a means for formally specifying the application programming interfaces defined by the Specifications.
Computational Resource	A piece of hardware or software that may vary in quantity, existence, or version, which may be required to exist in a minimum quantity and/or at a specific or minimum revision level so as to satisfy the requirements of the Test Suite. Examples of computational resources that may vary in quantity are RAM and file descriptors. Examples of computational resources that may vary in existence (that is, may or may not exist) are graphics cards and device drivers. Examples of computational resources that may vary in version are operating systems and device drivers.
Configuration Descriptor	Any file whose format is well defined by a specification and which contains configuration information for a set of Java classes, archive, or other feature defined in the specification.
Conformance Tests	All tests in the Test Suite for an indicated Technology Under Test, as distributed by the Maintenance Lead, excluding those tests on the Exclude List for the Technology Under Test.
Documented	Made technically accessible and made known to users, typically by means such as marketing materials, product documentation, usage messages, or developer support programs.
Exclude List	The most current list of tests, distributed by the Maintenance Lead, that are not required to be passed to certify conformance. The Maintenance Lead may add to the Exclude List for that Test Suite as needed at any time, in which case the updated Exclude List supplants any previous Exclude Lists for that Test Suite.
Libraries	The class libraries, as specified through the Java Community Process (JCP), for the Technology Under Test. The Libraries for Java EE Connector Architecture Version 1.7 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. Oracle is the Maintenance Lead for Java EE Connector Architecture Version 1.7.
Operating Mode	Any Documented option of a Product that can be changed by a user in order to modify the behavior of the Product. For example, an Operating Mode can be binary (enable/disable optimization), an enumeration (select from a list of protocols), or a range (set the maximum number of active threads). Note that an Operating Mode may be selected by a command line switch, an environment variable, a GUI user interface element, a configuration or control file, etc.

Table 2–1 (Cont.) Definitions

Term	Definition
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.
Resource	A Computational Resource, a Location Resource, or a Security Resource.
Rules	These definitions and rules in this Compatibility Requirements section of this User's Guide.
Security Resource	A security privilege or policy necessary for the proper execution of the Test Suite. For example, the user executing the Test Suite will need the privilege to access the files and network resources necessary for use of the Product.
Specifications	The documents produced through the Java Community Process that define a particular Version of a Technology. The Specifications for the Technology Under Test are referenced later in this chapter.
Technology	Specifications and a reference implementation produced through the Java Community Process.
Technology Under Test	Specifications and the reference implementation for Java EE Connector Architecture Version 1.7.
Test Suite	The requirements, tests, and testing tools distributed by the Maintenance Lead as applicable to a given Version of the Technology.
Version	A release of the Technology, as produced through the Java Community Process.

2.2.2 Rules for Java EE Connector Architecture Version 1.7 Products

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

JCA1 The Product must be able to satisfy all applicable compatibility requirements, including passing all Conformance Tests, in every Product Configuration and in every combination of Product Configurations, except only as specifically exempted by these Rules.

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

JCA1.1 If an Operating Mode controls a Resource necessary for the basic execution of the Test Suite, testing may always use a Product Configuration of that Operating Mode providing that Resource, even if other Product Configurations do not provide that Resource. Notwithstanding such exceptions, each Product must have at least one set of Product Configurations of such Operating Modes that is able to pass all the Conformance Tests.

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

JCA1.2 A Product Configuration of an Operating Mode that causes the Product to report only version, usage, or diagnostic information is exempted from these compatibility rules.

JCA1.3 An API Definition Product is exempt from all functional testing requirements defined here, except the signature tests.

JCA2 Some Conformance Tests may have properties that may be changed. Properties that can be changed are identified in the configuration interview. Properties that can be changed are identified in the JavaTest Environment (.jte) files in the lib 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 Conformance Test may be altered in any way without prior written permission. Any such allowed alterations to the Conformance Tests would be posted to the Java Licensee Engineering web site and apply to all licensees.

JCA3 The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

JCA4 The Exclude List associated with the Test Suite cannot be modified.

JCA5 The Maintenance Lead can define exceptions to these Rules. Such exceptions would be made available to and apply to all licensees.

JCA6 All hardware and software component additions, deletions, and modifications to a Documented supporting hardware/software platform, that are not part of the Product but required for the Product to satisfy the compatibility requirements, must be Documented and available to users of the Product.

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

JCA7 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.

JCA7.1 If a Product includes Technologies in addition to the Technology Under Test, then it must contain the full set of combined public and protected classes and interfaces. The API of the Product must contain the union of the included Technologies. No further modifications to the APIs of the included Technologies are allowed.

JCA8 Except for tests specifically required by this TCK to be rebuilt (if any), the binary Conformance Tests supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

JCA9 The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.

2.3 Java EE Connector Architecture Version 1.7 Test Appeals Process

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

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

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

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

Test challenges must be made in writing to Java Partner Engineering and include all relevant information as described in Example 2–1, "Test Challenge Form". The process used to determine the validity or invalidity of a test (or related group of tests) is described in Section 2.3.1, "Java EE Connector Architecture Version 1.7 TCK Test Appeals Steps."

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

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

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

2.3.1 Java EE Connector Architecture Version 1.7 TCK Test Appeals Steps

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

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

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

   If the appeal is incomplete or unclear, it is returned to the submitting licensee for correction. If all is in order, Java Licensee Engineering will check with the

responsible test developers to review the purpose and validity of the test before writing a response as described in Example 2–2, "Test Challenge Response Form". Java Licensee Engineering will attempt to complete the response within 5 business days. If the challenge is similar to a previously rejected test challenge (i.e., same test and justification), Java Licensee Engineering will send the previous response to the licensee.

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

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

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

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

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

2.3.2 Test Challenge and Response Forms

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

Example 2–1 Test Challenge Form

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

Example 2–2 Test Challenge Response Form

```
Test Defender Name and Company:
Test Defender Role in Defense (e.g., test developer, Maintenance Lead, etc.):
Specification Name(s) and Version(s):
Test Suite Name and Version:
Test Name:
Defense (argument for why test is valid):
[Multiple challenges and corresponding responses may be listed here.]
Implications of test invalidity (e.g., other affected tests and test framework
```

```
code, creation or exposure of ambiguities in spec (due to unspecified
requirements), invalidation of the reference implementation, creation of serious
holes in test suite):
Alternatives (e.g., are alternate test(s) appropriate?):
```

2.4 Specification for Java EE Connector Architecture Version 1.7

The Specification for Java EE Connector Architecture is found on the JCP web site at http://jcp.org/en/jsr/detail=?id=322.

2.5 Libraries for Java EE Connector Architecture Version 1.7

The following packages constitute the required class libraries for Java EE Connector Architecture 1.7:

javax.resource

javax.resource.cci

javax.resource.spi

javax.resource.spi.endpoint

javax.resource.spi.security

javax.resource.spi.work

3 Installation

This chapter explains how to install the Java EE Connector Architecture TCK 1.7 software. After installing the software according to the instructions in this chapter, proceed to Chapter 4, "Setup and Configuration," for instructions on configuring your test environment.

3.1 Installing the Software

Before you can run the JCA TCK tests, you need to install and set up the following software components:

- Java SE 7
- JCA 1.7 implementation under test
- JCA TCK Version 1.7

1. Install Java SE 7, if it is not already installed.
2. Install the implementation under test, if it is not already installed.

   Download, install, and configure the JCA TCK configuration that is under test. To familiarize yourself with the JCA TCK suite and the JavaTest software before you begin testing with your own implementation, you can optionally do a trial run using the Java EE 7 Web Profile RI.
3. Install the JCA TCK 1.7 software.

   a. Copy or download the JCA TCK software to your local system.

      The JCA TCK software is located in the Download Center area of the Java Licensee Engineering web site at `https://javapartner.oracle.com`.

   b. Change to the directory in which you want to install the JCA TCK software:

      ```
      cd install_directory
      ```

   c. Use the `unzip` command to extract the bundle:

      ```
      unzip connectortck-1.7_date.zip
      ```

      where *date* indicates the month and year in which the TCK bundle was created. For example, the JCA TCK bundle name could be `connectortck-1.7_15-May-2013.zip`

      When the bundle is unzipped, the `connectortck` directory is created. The `<install_directory>/connectortck` directory is the test suite home, `<TS_HOME>`.

4

Setup and Configuration

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

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

This chapter includes the following topics:

- Configuring Your Environment to Run the JCA TCK
- Custom Deployment Handlers
- Deploying the JCA TCK Tests

4.1 Configuring Your Environment to Run the JCA TCK

This section describes how to configure the JCA TCK for your environment. After configuring your environment, continue with the instructions in Chapter 5, "Executing Tests."

> **Note:** In these instructions, variables in angle brackets need to be expanded for each platform. For example, `<TS_HOME>`becomes `$TS_HOME` on Solaris/Linux and `%TS_HOME%` on Windows. In addition, the forward slashes (`/`) used in all of the examples need to be replaced with backslashes (`\`) for Windows. Finally, be sure to use the appropriate separator for your operating system when specifying multiple path entries (`;` on Windows, `:` on Solaris/Linux).

1. Set the following environment variables in your shell environment:
   a. `JAVA_HOME` to the directory in which Java SE 7 is installed
   b. `JAVAEE_HOME` to the installation directory of the Web server to which the JCA TCK 1.7 tests are to be deployed.

      For example, to deploy the JCA TCK tests to theJava EE 7 Web Profile RI, you would set this variable to the directory in which the Java EE 7 Web Profile RI was installed.
   c. `<TS_HOME>` to the directory in which the JCA TCK 1.7 software is installed
   d. `PATH` to include the `<TS_HOME>/bin` and <TS_HOME>/tools/ant/bin directories
   e. `ANT_HOME` should *not* be set in your environment. If it is set, unset it.

2. Edit your <TS_HOME>/bin/ts.jte file and set the following properties:
   a. webServerHost to the name of the host on which your Web server is running.
   b. webServerPort to the port number of the host on which the Web server is running.
   c. connector.home to the location where your JCA runtime implementation is installed.
   d. orb.host to the hostname of the system on which your JCA runtime was installed.
   e. orb.port to the port number of the system on which your JCA runtime was installed.
   f. connector.classes to the JCA classes or JAR files that contain the JCA classes.
   g. sigTestClasspath to the to the classes or JAR file (or files) for the runtime implementation of the JSR 322 API and any additional required signature or JCA API classes.
   h. impl.vi to the directory where vendor-specific targets can best be stored and isolated. The default directory, glassfish, is where details about the installation and configuration of the GlassFish enterprise server can be found.
   i. report.dir to the default directory in which JavaTest creates a test report for the most recent test run.

      This property is a required property for the TCK Ant targets; it must be set. To disable reporting, set the report.dir property to none.
   j. work.dir to the default directory in which JavaTest writes temporary files that are created during test execution.

      This property is required for the TCK Ant targets.
3. Provide your own implementations of the porting package interfaces provided with the JCA TCK.

   TSURLInterface.java obtains URL strings for web resources in an implementation-specific manner. API documentation for the TSURLInterface.java porting package interface is available in the documentation bundle in the docs/api directory.
4. Execute the config.vi Ant task.

   This target performs the following tasks:
   - Stops the application server running the JCA 1.7 RI
   - Copies the TCK-dependent files ${tslib.name}.jar and tsharness.jar into the application server's external library folder
   - Starts the application server
   - Sets the appropriate JVM options in the runtime
   - Creates connection pools and connection resources
   - Deploys the required RAs
   - Creates users in the runtime, then maps RA users to these runtime users

   Ensure that the users and passwords specified in the ts.jte file are created in the JCA server. These users and passwords include the ts.jte properties of rauser1,

rapassword1, user, password, authuser, authpassword, user_vi, password_vi, and so on.

4.2 Custom Deployment Handlers

Deployment handlers are used to deploy and undeploy the WAR files that contain the tests to be run during the certification process. A deployment handler is an Ant build file that contains at least the required targets listed in Table 4–1.

Table 4–1 Required Deployment Handler Targets

Required Ant Task	Description
-deploy	Deploys an archive or any archive from the current working directory and its children directories.
-undeploy	Undeploys a deployed archive or any deployed archive from the current working directory and its children directories.
-deploy.all	Deploys all archives.
-undeploy.all	Undeploys all deployed archives.

The Java EE Connector Architecture TCK provides these deployment handlers:

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

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

4.2.1 To Create a Custom Deployment Handler

To deploy tests to another JCA 1.7 implementation, you must create a custom handler.

1. Create a new directory in the <TS_HOME>/bin/impl directory tree.

   For example, create the <TS_HOME>/bin/impl/*my_deployment_handler* directory.

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

3. Modify the required targets in the deploy.xml file.

   This is what the deploy.xml file for the none deployment handler looks like.

```
<project name="No-op Deployment" default="deploy">

  <!-- No-op deployment target -->
  <target name="-deploy">
      <echo message="No deploy target implemented for this deliverable"/>
  </target>

  <target name="-undeploy">
      <echo message="No undeploy target implemented for this deliverable"/>
  </target>

  <target name="-deploy.all">
```

```
        <echo message="No deploy target implemented for this deliverable"/>
    </target>

    <target name="-undeploy.all">
        <echo message="No undeploy target implemented for this deliverable"/>
    </target>

</project>
```

Although this example just echoes messages, it does include the four required Ant targets (`-deploy`, `-undeploy`, `-deploy.all`, `-undeploy.all`) that your custom `deploy.xml` file must contain. With this as your starting point, look at the required targets in the `deploy.xml` files in the `tomcat` and `glassfish` directories for guidance as you create the same targets for the Web container in which you will run your implementation of Java EE Connector Architecture 1.7.

4. Set the `impl.vi` property in the `ts.jte` file to the name of the directory, *my_deployment_handler*, that you created in Step 1.

   The required Ant targets in your `deploy.xml` file can be called from anywhere in the `<TS_HOME>/src` directory. The `deploy.all` and `undeploy.all` targets can also be called from the `<TS_HOME>/bin` directory.

4.2.2 To Create a Custom Vehicle

A custom vehicle must be created and used when standalone connector tests are run in an environment that does not contain a Web server. If your standalone connector implementation includes a Web server, you do not need to implement your own custom vehicle.

The custom vehicle exists, in stubbed out form, and must be implemented in a way that provides a wrapper in which connector tests can execute. The default `connectorservlet` vehicle is an example of a vehicle that wraps and executes tests in a Servlet container. The `connectorservlet` vehicle source can be used a reference to help you implement your own custom vehicle. The `connectorservlet` vehicle is in the `src/com/sun/ts/tests/common/vehicle/connectorservlet` directory.

1. Use the stubbed-out `customvehicle` in the `src/com/sun/ts/tests/common/vehicle/customvehicle` directory as your starting point.
2. Modify the `CustomVehicleRunner` class, using other vehicles as references.

   The `bin/xml/ts.vehicles.xml` file includes a stubbed-out section for the `customvehicle`, which you can modify to build you own `customvehicle`.
3. Build the `customvehicle` you created.
4. Modify the `src/vehicle.properties` file so that it refers to `customvehicle` instead of `connectorservlet`.

   The `vehicle.properties` file is used during runtime to indicate in which vehicle the tests should be executed.
5. Remove or rename the `src/testsuite.jtd` file.

   This allows the test harness to identify tests to be run in your `customvehicle`.

4.2.3 Extension Libraries

The JCA 1.7 TCK RAR files are deployed simultaneously (see Section 4.3, "Deploying the JCA TCK Tests," for a list of all the RAR files that are deployed). 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 standalone RAR files can access. This extension library is needed to address classloading issues.

4.2.3.1 JCA 1.7 TCK Resource Adapter Files

The Resource Adapter (RAR) files that are used with the JCA 1.7 TCK differ from those that were used in earlier JCA TCK releases. The TCK no longer includes 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 7 platform specification and section 20.2.0.1 in the JCA 1.7 specification, which explicitly state that the resource adapter server may employ the library mechanisms in Java EE 7.

- Support application-based standalone connector accessibility

  See section 20.2.0.4 of the JCA 1.7 specification, which uses the classloading requirements listed in section 20.3 in the specification.

4.2.3.2 JCA 1.7 TCK Resource Adapters and Classloading

The JCA TCK 1.7 has scenarios in which multiple standalone RAR files that use the same shared library (for example, `whitebox.jar`) are referenced from a TCK 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.

4.2.3.3 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: , .

1. RAR1's Classloader: RAR1's classes and `whitebox.jar`
2. 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 7 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 TCK's RI, `domain-dir/lib/applibs` is used as the Installed Library directory and is the location to which the `whitebox.jar` file gets copied.

4.2.3.4 Required Porting Package

The JCA 1.7 TCK 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 7 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.

4.3 Deploying the JCA TCK Tests

The JCA TCK tests should only be deployed *after* you properly configure your connector runtime. This configuration includes creating users and JVM options in the runtime, mapping RA users to existing runtime users, deploying RAs, and creating connection pools and connection resources.

Deployment of the connector resource adapters involves the deployment of 17 RAR files:

- `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
- whitebox-ibanno_no_md.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.

Deploying RAR files require the installation of a whitebox.jar file to a directory into which the server can load. The whitebox.jar file is a common library, which is referenced and used by all the RAR files. To ensure proper class loading, the whitebox.jar file must be copied into the application server's domains/domain1/lib/applibs directory, a location where it can be loaded by the server and can be accessed by all the RAR files.

To deploy the JCA TCK tests to the Java EE 7 platform, perform the following steps.

4.3.1 To Configure and Deploy the JCA TCK Tests on the Java EE 7 Web Profile RI

The config.vi Ant task performs several configuration procedures on your connector server. This target deploys some of the required Resource Adapters (RAR files) and creates some connection resources and connection pools. The test suite also provides a convenience Ant target, which deploys only the RAR files and creates connection pools and connection resources. This Ant target is in the <TS_HOME>/bin/xml/impl/glassfish/connector.xml file. See the Ant target setup.all.rars.and.pools for more information.

1. Make sure that the server to which you will deploy the JCA TCK tests is running.
2. Change to the <TS_HOME>/bin directory and execute the Ant task to configure the RI.

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

3. Execute the ant deploy.all Ant task to deploy the requisite RAR files.

   ```
   ant deploy.all
   ```

   Follow the instructions in Section 5.1, "Using the GUI for TCK Test Execution," or Section 5.2, "Using the Command-Line for TCK Test Execution," to run the tests you just deployed.

4.3.2 Configuring the JCA TCK Tests on the Vendor Implementation

Vendors need to configure their application or connector servers to run the TCK tests. This configuration needs to support the same features that are currently performed against the RI through the execution of the config.vi and setup.all.rars.and.pools Ant targets.

This section describes how to configure the Vendor Implementation (VI) before running the JCA TCK tests.

Performing the tasks of these targets should minimally include the following:

- Creating Security Mappings for the RAR Files
- Creating Required Server-Side JVM Options
- Replacing the Default Vehicle with a Custom Vehicle

4.3.2.1 Creating Security Mappings for the RAR Files

See the Ant target `create.security.eis.mappings` in the `<TS_HOME>/bin/xml/impl/glassfish/connector.xml` to see how this is done with the RI. This task maps Resource Adapter user information to existing user information in the connector runtime.

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`

4.3.2.2 Creating Required Server-Side JVM Options

Create the required JVM options to allow the passing and/or setting of user information from the `ts.jte` file to the server. These properties are set in the connector runtime for use by the RAR files, which are deployed to the connector runtime. The RAR files need to know some of the property settings used in the `ts.jte` file; this is the means used to specify some of the user properties.

To see some of required JVM options for the RI, examine the `ri.jvm.options` property in the `ts.jte` file. The following subset of JVM option specified in the `ri.jvm.options` property must be set in the connector runtime:

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

4.3.2.3 Replacing the Default Vehicle with a Custom Vehicle

If your connector server does not have web support, you will need to create your own vehicle. A vehicle is a wrapper that supports running tests in different server-side containers, such as servlet, JSP, and so on. The JCA TCK provides a default vehicle, `connectorservlet`, which supports running the TCK tests in a connector runtime that has a Servlet container. To support running tests in an environment other than a Servlet container, you need to implement your own vehicle, effectively replacing the default vehicle, `connectorservlet`.

This TCK was designed so you could use `connectorservlet` as a template for creating your own vehicle. The `connectorservlet` vehicle is used to contain and execute your client-side tests in the connector runtime.

The `connectorservlet` vehicle is located in the `<TS_HOME>/src/com/sun/ts/tests/common/vehicle/connectorservlet` directory.

To run the tests in a vehicle other than `connectorservlet`, you need to create a custom vehicle named `customvehicle`. See Section 4.2.2, "To Create a Custom Vehicle," for more information on this topic.

4.3.3 Deploying the RAR files to the VI

This section describes how deploy the RAR files to the VI before running the JCA TCK tests.

The deployment requirements for VI are similar to the requirements for the RI in that the required Resource Adapter (RAR) files must be deployed before any TCK tests can be run. Before you attempt to deploy the RAR files, ensure your connector server has been properly configured. For additional information about deploying the RAR files and creating the corresponding connection pools and connection resources, see the `setup.all.rars.and.pools` Ant target in the `<TS_HOME>/bin/xml/impl/glassfish/connector.xml file`.

5 Executing Tests

The JCA TCK uses the JavaTest harness to execute the tests in the test suite. The JCA TCK can be run using either the JavaTest harness software through the JavaTest GUI or from the command line in your shell environment.

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

This chapter includes the following topics:

- Using the GUI for TCK Test Execution
- Using the Command-Line for TCK Test Execution

5.1 Using the GUI for TCK Test Execution

The JavaTest documentation in the Java EE Connector Architecture TCK 1.7 documentation bundle contains more detailed instructions on running and using JavaTest.

> **Note:** It is only necessary to proceed with this section if you want to run the JavaTest harness in GUI mode. If you plan to run the TCK tests in command-line mode, skip the remainder of this chapter, and continue with Section 5.2, "Using the Command-Line for TCK Test Execution."

5.1.1 To Start JavaTest in GUI Mode

1. Change to the `<TS_HOME>/bin` directory.

   ```
   cd <TS_HOME>/bin
   ```

   The `build.xml` file in the `<TS_HOME>/bin` directory contains the Ant targets for the JCA TCK test suite.

2. Execute the `ant gui` target.

   ```
   ant gui
   ```

5.1.2 Configuring the JavaTest Harness in GUI Mode

In order for the JavaTest harness to execute the test suite, it requires information about how your computing environment is configured. The JavaTest harness requires two types of configuration information:

- **Test environment**: This is data used by the tests. For example, the path to the Java runtime, how to start the product being tested, network resources, and other information required by the tests in order to run. This information does not change frequently and usually stays constant from test run to test run.
- **Test parameters**: This is information used by the JavaTest harness to run the tests. Test parameters are values used by the JavaTest harness that determine which tests in the test suite are run, how the tests should be run, and where the test reports are stored. This information often changes from test run to test run.

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

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

5.1.3 To Configure the JavaTest Harness to Run the JCA TCK Tests

You only need to complete all these steps the first time you start the JavaTest test harness. After you complete these steps, you can either run all of the tests by completing the steps in Section 5.1, "Using the GUI for TCK Test Execution," or run a subset of the tests by completing the steps in Section 5.1.5, "To Run a Subset of the TCK Tests in GUI Mode."

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

1. Start the JavaTest test harness.

   The Welcome screen displays.

   If the Welcome screen does not appear, select **File** and then click **Open Quick Start Wizard**.

2. Select **Start a new test run**, and then click **Next**.

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

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

   You are prompted to select a test suite.

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

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

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

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

6. Deselect the **Start the configuration editor** option, and then click **Finish**.

7. Click **Run Tests**, then click **Start**.

   The JavaTest harness starts running the tests.

8. To reconfigure the JavaTest test harness, do one of the following:

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

9. Click **Report**, and then click **Create Report**.

10. Specify the directory in which the JavaTest test harness will write the report, and then click **OK**.

    A report is created, and you are asked whether you want to view it.

11. Click **Yes** to view the report.

5.1.4 Modifying the Default Test Configuration in GUI Mode

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

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

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

5.1.5 To Run a Subset of the TCK Tests in GUI Mode

1. From the JavaTest main menu, click **Configure**, then click **Change Configuration**, and then click **Tests to Run**.

   The tabbed Configuration Editor dialog box is displayed.

2. Click **Specify** from the option list on the left.

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

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

4. Click **Save File**.

5. Click **Run Tests**, and then click **Start** to run the tests you selected.

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

6. Click **Report**, and then click **Create Report**.

7. Specify the directory in which the JavaTest test harness will write the report, and then click **OK**.

   A report is created, and you are asked whether you want to view it.

8. Click **Yes** to view the report.

5.2 Using the Command-Line for TCK Test Execution

The following examples assume that you have already deployed the JCA TCK tests. See Step 6 in Section 4.3, "Deploying the JCA TCK Tests," for more information about deploying the JCA TCK tests.

Example 5–1 All JCA TCK Tests

To run all of the JCA TCK tests, enter the following commands:

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

Example 5–2 MDB-Specific JCA TCK Tests

To run the MDB-specific JCA TCK tests, enter the following commands:

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

Alternatively, from the <TS_HOME>/src/com/sun/ts/tests/connector directory, you can also run the MDB-specific JCA TCK tests by using the appropriate keyword:

```
cd <TS_HOME>/src/com/sun/ts/tests/connector
ant -Dkeywords="connector_mdb_optional" runclient
```

Example 5–3 EJB and Servlet Resource Definition Annotation Tests

To run the JCA TCK EJB and Servlet Resource Definition Annotation Tests, enter the following commands:

```
cd <TS_HOME>/src/com/sun/ts/tests/connector/resourceDefs
```

```
ant runclient
```

Alternatively, from the <TS_HOME>/src/com/sun/ts/tests/connector directory, you can also run the JCA TCK Web tests by using the appropriate keyword:

```
cd <TS_HOME>/src/com/sun/ts/tests/connector
ant -Dkeywords=
  "(connector_resourcedef_ejb_optional | connector_resourcedef_servlet_optional)"
  runclient
```

Example 5–4 All JCA TCK Tests Except the EJB and Servlet Resource Definition Annotation Tests

From the <TS_HOME>/src/com/sun/ts/tests/connector directory, you can run all the JCA TCK tests except the EJB and Servlet Resource Definition Annotation Tests by using the appropriate keyword:

```
cd <TS_HOME>/src/com/sun/ts/tests/connector
ant -Dkeywords=
  "!(connector_resourcedef_ejb_optional | connector_resourcedef_servlet_optional)"
  runclient
```

Example 5–5 JCA TCK Signature Tests

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

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

5.2.1 To Run Tests that Failed (PriorStatus) From the Command Line

You can run certain tests based on the test's prior run status by specifying the `priorStatus` system property when invoking `ant`.

1. Use the keywords command to select the tests for a test run based on their outcome on a prior test run:

   `priorStatus` *`status-arguments`*

   The *status-arguments* that can be used are `pass`, `fail`, `error`, and `notRun`.

2. Invoke `ant` with the `priorStatus` keyword.

   Use commas to separate multiple arguments.

   For example, to run all the tests that had a status of failed and error during a previous test run, you would invoke the following command:

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

6

Rebuilding Tests and Debugging Test Problems

There are a number of reasons that tests can fail to execute properly. This chapter provides some approaches to help you deal with these failures.

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

6.1 Overview

The goal of a test run is for all tests in the test suite that are not filtered out to have passing results. If a test run includes 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. 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 GUI tools described in this chapter. Ant test execution tasks provide command-line users with immediate test execution feedback to the display. Available JTR report files and log files can also help command-line users troubleshoot test run problems.

For every test run, the JavaTest harness creates a set of report files in the reports directory, which you specified by setting the `report.dir` property in the `ts.jte` file. The report files contain information about the test description, environment, messages, properties used by the test, status of the test, and test result. After a test run is completed, the JavaTest harness writes HTML reports for the test run. You can view these files in the JavaTest ReportBrowser when running in GUI mode, or in the Web browser of your choice outside the JavaTest interface. To see all of the HTML report files, enter the URL of the `report.html` file. This file is the root file that links to all of the other HTML reports.

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

The work directory, which you specified by setting the `work.dir` property in the `ts.jte` file, contains several files that were deposited there during test execution:

`harness.trace`, `log.txt`, `lastRun.txt`, and `testsuite`. Most of these files provide information about the harness and environment in which the tests were executed.

Note: You can set the `harness.log.traceflag` property in the `<TS_HOME>/bin/ts.jte` file and the HARNESS_DEBUG environment variable to "true" to get more debugging information.

If a large number of tests failed, you should read Section 6.6, "Recognizing Configuration Failures," to see if a configuration issue is the cause of the failures. If the failures are more limited or more concentrated in one or two areas, you can use Ant to build, deploy, and run selected portions of the test suite to debug test problems.

If, after examining the test results, you see errors in just a few areas, you may want to refer to Section 6.4, "Building Tests Using Ant," and Section 6.5, "Deploying Tests Using Ant," to see how to rebuild, deploy, and run selected portions of the TCK. Do not lose sight of the fact, though, that to prove compatibility, you must run the *entire* test suite.

6.2 Debugging Test Results with the JavaTest GUI

The following sections describe features of the JavaTest GUI that enable you to view and analyze the results of your test runs.

6.2.1 Using the Test Tree in the GUI

Use the test tree in the JavaTest GUI to identify specific folders and tests that had errors or failing results. Color codes are used to indicate status as follows:

- **Green**: Passed
- **Blue**: Test Error
- **Red**: Failed to pass test
- **White**: Test not run
- **Gray**: Test filtered out (not run)

6.2.2 Displaying Folder Information in the GUI

Click a folder in the test tree in the JavaTest GUI to display its tabs.

Choose the **Error** and the **Failed** tabs to view the lists of all tests in and under a folder that were not successfully run. You can double-click a test in the lists to view its test information.

6.2.3 Displaying Test Information in the GUI

To display information about a test in the JavaTest GUI, click its icon in the test tree or double-click its name in a folder status tab. The tab contains detailed information about the test run and, at the bottom of the window, a brief status message identifying the type of failure or error. This message may be sufficient for you to identify the cause of the error or failure.

If you need more information to identify the cause of the error or failure, use the following tabs listed in order of importance:

- **Test Run Messages** contains a Message list and a Message section that display the messages produced during the test run.
- **Test Run Details** contains a two-column table of name/value pairs recorded when the test was run.
- **Configuration** contains a two-column table of the test environment name/value pairs derived from the configuration data actually used to run the test.

6.2.4 Creating and Viewing Test Reports in GUI Mode

This section explains how to use the GUI to create and view report files.

6.2.4.1 To Create a Test Report

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

   You are prompted to specify a directory to use for your test reports. The default location is `/tmp/JTreport` or whatever was specified by the `report.dir` property in the `ts.jte` file.

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

   Use the **Filter** list to specify whether you want to generate reports for the current configuration, for all tests, or for 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.

6.2.4.2 To View an Existing Report

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

   You are prompted to specify the directory containing the report you want to open.

2. Select the report directory you want to open, and then click **Open**.

   The selected report set is opened in the JavaTest ReportBrowser.

6.3 Creating and Viewing Report and Log Files Using Ant

This section explains how to use Ant to create and view report files.

6.3.1 To Create a Test Report

Specify where you want to create the test report.

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

   ```
   ant -Dreport.dir="<report_dir>"
   ```

   Reports for the next test run will be written to the directory you specify.

2. To disable reporting, set the `report.dir` property to `"none"` on the command line.

   For example:

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

If you do not specify a directory or disable reporting, reports will be written to the location specified by the `report.dir` property in the `ts.jte` file.

6.3.2 To View an Existing Test Report

1. Change to the report directory you that you specified from the command line or set in the `ts.jte` file.
2. Start the Web browser of your choice from that directory.
3. View the `report.html` file.

6.3.3 To Examine Log Files

1. Change to the work directory you that you set in the `ts.jte` file.
2. Look in the `harness.trace`, `log.txt`, `lastRun.txt`, and `testsuite` files to see if configuration issues related to the test environment or the test harness were the cause of the test failures.

6.4 Building Tests Using Ant

If your test run resulted in failures that were localized in one area, it may be beneficial to build, deploy, and run that area instead of running the entire test suite.

This section explains how to use Ant to build a single test directory or a subset of test directories, and shows how to list the classes directory and distribution directory of archives for the directory that was built.

1. To build a single test directory, change to a test directory that has no subdirectories and type:

   ```
   ant clean build
   ```

   This cleans and builds the tests in the test directory that you specified.

2. To list the classes directory for this test that was built, type:

   ```
   ant lc
   ```

   or

   ```
   ant llc
   ```

3. To list the distribution directory of archives for this test that was built, type:

   ```
   ant ld
   ```

   or

   ```
   ant lld
   ```

4. To build a subset of test directories, change to a test directory that has subdirectories and type:

   ```
   ant clean build
   ```

   This cleans and builds all the test directories under the specified test directory.

6.5 Deploying Tests Using Ant

If your test run resulted in failures that were localized in one area, it may be beneficial to build, deploy, and run that area instead of running the entire test suite. Section 6.4, "Building Tests Using Ant," explains how to rebuild parts of the test suite; this section explains how to use Ant to deploy and run the parts that you rebuilt.

This section explains how to use Ant to deploy a single test directory, a subset of test directories, and a single test.

1. Change to a directory under `<TS_HOME>/src/com/sun/ts/tests/`*technology*`/`*technology-testdir*.

   To deploy a single test directory, change to a test directory that has no subdirectories.

   To deploy a subset of test directories, change to a test directory that has subdirectories.

2. Type the following command:

   ```
   ant deploy
   ```

   If you changed to a directory that has no subdirectories, the test EAR, WAR, or JAR file built for the specified test directory will be deployed.

   If you changed to a directory that has subdirectories, the EAR, WAR, or JAR files built for the parent directory and its subdirectories will be deployed.

3. Follow the instructions in Section 5.2, "Using the Command-Line for TCK Test Execution," to run the tests in the deployed EAR, WAR, or JAR files.

6.6 Recognizing Configuration Failures

Configuration failures are easily recognized because many tests fail the same way. When all your tests begin to fail, you may want to stop the run immediately and start viewing individual test output. However, in the case of full-scale launching problems where no tests are actually processed, report files are usually not created (though sometimes a small `harness.trace` file in the report directory is written).

A

Frequently Asked Questions

This appendix contains the following questions.

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

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

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

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

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

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

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