Oracle®

Technology Compatibility Kit User's Guide for Java
Authorization Contract for Containers 1.5

Release 1.5 for Technology Licensees

September 2017

ORACLE®

Technology Compatibility Kit User's Guide for Java Authorization Contract for Containers 1.5, Release 1.5 for Technology Licensees

Contents

Preface

This guide describes how to install, configure, and run the Technology Compatibility Kit (TCK) that is used to test the Java Authorization Contract for Containers (JACC 1.5) (JSR 115) technology.

The JACC TCK is designed as a portable, configurable automated test suite for verifying the compatibility of a licensee's implementation of the JACC 1.5 Specification (hereafter referred to as the licensee implementation). The JACC TCK uses the JavaTest harness version 4.4.1 to run the test suite

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

Who Should Use This Book

This guide is for licensees of the JACC 1.5 technology to assist them in running the test suite that verifies compatibility of their implementation of the JACC 1.5 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

You should be familiar with the Java Authorization Contract for Containers 1.5 Specification, which can be found at http://jcp.org/en/jsr/detail?id=115.

Before running the tests in the JACC TCK, you should familiarize yourself with the JavaTest documentation that is included in the JACC 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, and Korn shell.

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

1

Introduction

This chapter provides an overview of the principles that apply generally to all Technology Compatibility Kits (TCKs) and describes the Java Authorization Contract for Containers TCK (JACC TCK 1.5) (JSR 115). It also includes a high level listing of what is needed to get up and running with the JACC 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 Authorization Contract for Containers (JACC) 1.5 Certification."

1.1.3 TCK Overview

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

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

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

1.1.4 Java Community Process (JCP) Program and Compatibility Testing

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

- Technology Specification

- Reference Implementation

- Technology Compatibility Kit (TCK)

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

1.2 About the JACC TCK 1.5

The JACC TCK 1.5 is designed as a portable, configurable, automated test suite for verifying the compatibility of a licensee's implementation of the JACC 1.5 Specification.

1.2.1 JACC TCK Specifications and Requirements

This section lists the applicable requirements and specifications.

- **Specification Requirements**: Software requirements for a JACC implementation are described in detail in the JACC 1.5 Specification. Links to the JACC specification and other product information can be found at http://jcp.org/en/jsr/detail?id=115.

- **JACC Version**: The JACC TCK 1.5 is based on the JACC Specification, Version 1.5.

- **Reference Implementation**: The Reference Implementation (RI) for JACC 1.5 can be downloaded from the Java Licensee Engineering (https://javapartner.oracle.com) Web site. This RI can be used with any container that meets the minimum requirements for a container as defined in the JACC 1.5 Specification. The JACC 1.5 APIs have also been integrated into the Java EE 6 RI, which is available from Java Licensee Engineering.

See the *Java Authorization Contract for Containers TCK 1.5 Release Notes* for more specific information about JDK version requirements, supported platforms, restrictions, and so on.

1.2.2 JACC TCK Components

The JACC TCK 1.5 includes the following components:

- **JavaTest harness** version 4.4.1 and related documentation. The JavaTest documentation bundle is available on the Java Licensee Engineering (https://javapartner.oracle.com) Web site.

- **JACC TCK signature tests** check that all public APIs are supported and/or defined as specified in the JACC Version 1.5 implementation under test.

- **API tests** for the JACC APIs in the javax.security.jacc package.

- **End-to-end tests** that demonstrate compliance with the Java Authorization Contract for Containers 1.5specification.

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 JACC TCK 1.5 to test a JACC 1.5 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 run or pass any test on the Exclude List. The Exclude List file, `<TS_HOME>/bin/ts.jtx`, is included in the JACC TCK.

> **Note:** From time to time, updates to the Exclude List are made available on the Java Licensee Engineering web site (Java Licensee Engineering (`https://javapartner.oracle.com`)). You should always make sure you are using an up-to-date copy of the Exclude List before running the JACC 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 Oracle 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, "Java Authorization Contract for Containers Version 1.5 Test Appeals Process."

1.2.6 JACC 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 JACC tests, as described in Chapter 4, "Setup and Configuration."

1.3 Getting Started With the JACC TCK

This section provides an general overview of what needs to be done to install, set up, test, and use the JACC 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:

   - The Java EE 7 RI, which is the Reference Implementation for JACC

   - Java SE 7

- JACC TCK version 1.5, which includes Apache Ant 1.7.0

- The JACC implementation to be tested. If you are running against the Reference Implementation, install the Java EE 7 RI software; if not, install your JACC Vendor Implementation (VI)

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

2. Set up the JACC 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. (The *install_directory*/`jacctck` directory is the test suite home, `<TS_HOME>`.)

   c. Configure the JavaTest harness.

3. Test the JACC 1.5 implementation.

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

2

Procedure for Java Authorization Contract for Containers (JACC) 1.5 Certification

This chapter describes the compatibility testing procedure and compatibility requirements for Java Authorization Contract for Containers 1.5. This chapter contains the following sections:

- Certification Overview
- Compatibility Requirements
- Java Authorization Contract for Containers Version 1.5 Test Appeals Process
- Specification for Java Authorization Contract for Containers Version 1.5
- Library for Java Authorization Contract for Containers Version 1.5

2.1 Certification Overview

The certification process for Java Authorization Contract for Containers Version 1.5 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 Authorization Contract for Containers Version 1.5 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 Authorization Contract for Containers Version 1.5 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. [Maint-Lead] is the Maintenance Lead for Java Authorization Contract for Containers Version 1.5.
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 Authorization Contract for Containers Version 1.5.
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 Authorization Contract for Containers Version 1.5 Products

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

JACC1 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.

JACC1.1 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.

JACC1.2 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.

JACC1.3 An API Definition Product is exempt from all functional testing requirements defined here, except the signature tests.

JACC2 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.

JACC3 The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

JACC4 The Exclude List associated with the Test Suite cannot be modified.

JACC5 The Maintenance Lead can define exceptions to these Rules. Such exceptions would be made available to and apply to all licensees.

JACC6 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.

JACC7 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.

JACC7.1 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.

JACC8 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.

JACC9 The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.

2.3 Java Authorization Contract for Containers Version 1.5 Test Appeals Process

[Maint-Lead] 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. [Maint-Lead], as Java Authorization Contract for Containers 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 Authorization Contract for Containers 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 Authorization Contract for Containers 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 Authorization Contract for Containers Version 1.5 TCK Test Appeals Steps."

All tests found to be invalid will either be placed on the Exclude List for that version of the Java Authorization Contract for Containers 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 Authorization Contract for Containers TCK licensees on the Java Authorization Contract for Containers TCK website.

- [Maint-Lead], 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 Authorization Contract for Containers TCK website.

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

2.3.1 Java Authorization Contract for Containers Version 1.5 TCK Test Appeals Steps

1. **Java Authorization Contract for Containers TCK licensee writes a test challenge to Java Licensee Engineering contesting the validity of one or a related set of Java Authorization Contract for Containers 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, [Maint-Lead] places the tests on the Exclude List for that version of the Java Authorization Contract for Containers 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, [Maint-Lead] 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** [Maint-Lead] **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 Authorization Contract for Containers Version 1.5

The Specification for Java Authorization Contract for Containers is found on the JCP web site at http://jcp.org/en/jsr/detail=?id=115.

2.5 Library for Java Authorization Contract for Containers Version 1.5

The following package constitutes the required class library for Java Authorization Contract for Containers:

javax.security.jacc

3

Installation

This chapter explains how to install the Java Authorization Contract for Containers TCK 1.5 (JACC TCK) software. After installing the software according to the instructions in this chapter, proceed to Chapter 4, "Setup and Configuration," for instructions on configuring your test environment.

3.1 Obtaining the JACC 1.5 Reference Implementation

You can obtain the JACC 1.5 Reference Implementation (RI) software from Java Licensee EngineeringJava Licensee Engineering (`https://javapartner.oracle.com`).

3.2 Installing the Software

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

- Java SE 7

- JACC TCK version 1.5, which includes Apache Ant 1.7.0

- The JACC implementation to be tested

    If you are running against the JACC 1.5 Reference Implementation, install the Java EE 7 RI software; if not, install your JACC Vendor Implementation (VI).

1. Install the Java SE 7 software, if it is not already installed.

    Download and install the Java SE 7 software from `http://www.oracle.com/technetwork/java/javase/downloads/index.html`. Refer to the installation instructions that accompany the software for additional information.

2. Install the JACC TCK 1.5 software.

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

        You can obtain the JACC TCK software from the Java Licensee Engineering (`https://javapartner.oracle.com`) web site. The JACC TCK software is located in the `OPTPKG-XML/jacctck` directory in the Web site's Download Center area.

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

        `unzip jacctck-1.4_date.zip`

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

3. Install the JACC 1.5 Reference Implementation.

   The Reference Implementation is used to validate your initial configuration and setup of the JACC TCK 1.5 tests, which are explained further in Chapter 4, "Setup and Configuration."

   You can obtain the JACC RI from Java Licensee Engineering at Java Licensee Engineering (`https://javapartner.oracle.com`).

4. Install the JACC Vendor Implementation (VI) to be tested.

   Follow the installation instructions for the particular VI under test.

4

Setup and Configuration

This chapter describes how to set up the JACC 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 JACC TCK.

This chapter includes the following topics:

- Configuring Your Environment to Run the JACC TCK Against the JACC Reference Implementation
- Configuring Your Environment to Run the JACC TCK Against Your JACC 1.5 Vendor Implementation
- Custom Deployment Handlers
- Using the JavaTest Harness Software
- Using the JavaTest Harness Configuration GUI

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

After configuring your environment as described in this section, continue with the instructions in Section 4.4, "Using the JavaTest Harness Software."

> **Note:** In these instructions, variables in angle brackets need to be expanded for each platform. For example, `<TS_HOME>` becomes `$TS_HOME` on Solaris/Linux and `%TS_HOME%` on Windows. In addition, the forward slashes (/) used in all of the examples need to be replaced with backslashes (\) for Windows. Finally, be sure to use the appropriate separator for your operating system when specifying multiple path entries (; on Windows, : on UNIX/Linux).
>
> On Windows, you must escape any backslashes with an extra backslash in path separators used in any of the following properties, or use forward slashes as a path separator instead.

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

    a. `JAVA_HOME` to the directory in which Java SE 7 is installed

    b. `PATH` to include the `<TS_HOME>/bin` and `<TS_HOME>/tools/ant/bin` directories

    c. `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 variables:

    a. Set the `jacc.home` property to the location where your JACC runtime implementation is installed.

    b. Set the `jacc.host` property to the hostname of the system where your JACC runtime is installed.

    c. Set the `jacc.classes` property to point to the classes or JAR file that contains the JACC classes.

    d. Set the `sigTestClasspath` property to point to the classes or JAR file for the runtime implementation of the JSR 115 API and any additional required signature classes.

3. Copy the `tsharness.jar` and `jacctck.jar` files to the server's extension directory, change to the `<TS_HOME>/bin` directory and execute the following commands:

```
ant config.vi
ant enable.jacc
```

4.2 Configuring Your Environment to Run the JACC TCK Against Your JACC 1.5 Vendor Implementation

After configuring your environment as described in this section, continue with the instructions in Section 4.4, "Using the JavaTest Harness Software."

> **Note:** In these instructions, variables in angle brackets need to be expanded for each platform. For example, `<TS_HOME>` becomes `$TS_HOME` on Solaris/Linux and `%TS_HOME%` on Windows. In addition, the forward slashes (/) used in all of the examples need to be replaced with backslashes (\) for Windows. Finally, be sure to use the appropriate separator for your operating system when specifying multiple path entries (; on Windows, : on UNIX/Linux).
>
> On Windows, you must escape any backslashes with an extra backslash in path separators used in any of the following properties, or use forward slashes as a path separator instead.

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

    a. `JAVA_HOME` to the directory in which Java SE 7 is installed

    b. `PATH` to include the `<TS_HOME>/bin` and `<TS_HOME>/tools/ant/bin` directories

    c. `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 variables:

    a. Set the `jacc.home` property to the location where your JACC runtime implementation is installed.

    b. Set the `jacc.host` property to the hostname of the system where your JACC runtime is installed.

    c. Set the `jacc.classes` property to point to the classes or JAR file that contains the JACC classes.

    d. Set the `sigTestClasspath` property to point to the classes or JAR file for the runtime implementation of the JSR 115 API and any additional required signature classes.

3. Copy the `tsharness.jar` and `jacctck.jar` files to your server's extension directory and deploy.

4. Change to the `<TS_HOME>/bin` directory and execute the following command:

```
ant config.vi
```

The `config.vi` Ant task performs several actions, including:

- Sets the following JACC JVM options:

```
-Djavax.security.jacc.policy.provider=
        com.sun.ts.tests.jacc.provider.TSPolicy
-Dvendor.javax.security.jacc.policy.provider=
        com.sun.enterprise.security.provider.PolicyWrapper
-Djavax.security.jacc.PolicyConfigurationFactory.provider=
        com.sun.ts.tests.jacc.provider.TSPolicyConfigurationFactoryImpl
-Dvendor.javax.security.jacc.PolicyConfigurationFactory.provider=
        com.sun.enterprise.security.provider.PolicyConfigurationFactoryImpl
-Dlog.file.location=${log.file.location}
```

  Note that the `log.file.location` comes from the property of the same name in the `ts.jte` file.

- Deploys the JACC Provider (from `<TS_HOME>/lib/tsprovider.jar`) to your server's library directory (for example, `glassfish4/glassfish/lib`) where it can be picked up and loaded by the server

- Enables the Security manager with the `-Djava.security.manager` JVM option

- Creates users required by the TCK tests on the server under test

- Deploys `tsharness.jar` and `jacctck.jar` files to your JACC server's `/lib` directory

5. Enable the TCK JACC provider:

```
ant enable.jacc
```

After running the JACC TCK tests, disable the JACC provider by running the disable.jacc Ant task:

```
ant disable.jacc
```

6. Change to the appropriate JACC TCK test subdirectory (`<TS_HOME>/src/com/sun/ts/tests/jacc/web` or `<TS_HOME>/src/com/sun/ts/tests/jacc/ejb`) for the tests that you plan to run and execute the `ant deploy` command to deploy the desired tests.

To deploy the JACC EJB tests:

```
cd ${TS_HOME}/src/com/sun/ts/tests/jacc/ejb
ant deploy
```

Or, to deploy the JACC Web tests:

```
cd ${TS_HOME}/src/com/sun/ts/tests/jacc/web
ant deploy
```

Repeat this deployment step for each JACC test directory after you run the tests in the current directory, as described in Chapter 5, "Executing Tests."

JACC tests translate security configurations into corresponding JACC permissions. If multiple test directories are deployed simultaneously, the result can be permissions that are stricter than what is expected, which can lead to test failures. To avoid this potential problem, deploy and run individual test directories separately, not simultaneously.

4.3 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. (Do not implement for the JACC TCK.)
undeploy.all	Undeploys all deployed archives. (Do not implement for the JACC TCK.)

The Java Authorization Contract for Containers TCK provides the following 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 or 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).

> **Note:**   The JACC TCK does not require you to implement the `-deploy.all` or `-undeploy.all` targets. In fact, it is recommended that you run each JACC TCK test subdirectory individually, not collectively.

4.3.1 Creating a Custom Deployment Handler

To deploy tests to another JACC 1.5 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 `glassfish` directory for guidance as you create the same targets for the Web container in which you will run your implementation of JACC.

4. Set the `impl.vi.deploy.dir` 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.

4.4 Using the JavaTest Harness Software

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

- Through the JavaTest GUI; if using this method, please continue on to Section 4.5, "Using the JavaTest Harness Configuration GUI."

- In JavaTest batch mode, from the command line in your shell environment; if using this method, please proceed directly to Chapter 5, "Executing Tests."

4.5 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 JACC TCK test environment.

Note: It is only necessary to proceed with this section if you want to run the JavaTest harness in GUI mode. If you plan to run the JavaTest harness in command-line mode, skip the remainder of this chapter, and continue with Chapter 5, "Executing Tests."

4.5.1 Configuration GUI Overview

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

- **Test environment** : This is data used by the tests. For example, the path to the Java runtime, how to start the product being tested, network resources, and other information required by the tests in order to run. This information does not change frequently and usually stays constant from test run to test run.

- **Test parameters**: This is information used by the JavaTest harness to run the tests. Test parameters are values used by the JavaTest harness that determine which tests in the test suite are run, how the tests should be run, and where the test reports are stored. This information often changes from test run to test run.

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

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

4.5.2 Starting the Configuration GUI

Before you start the JavaTest harness software, you must have a valid test suite and Java SE 7 installed on your system.

The JACC TCK includes an Ant script that is used to execute the JavaTest harness from the <TS_HOME> directory. Using this Ant script to start the JavaTest harness is part of the procedure described in Section 4.5.3, "To Configure the JavaTest Harness to Run the JACC TCK Tests."

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

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

- If the JavaTest harness is unable to open a test suite, it displays a Welcome to JavaTest dialog box that guides you through the process of opening both a test suite and a work directory as described in the JavaTest documentation.

After you specify a work directory, you can use the Test Manager to configure and run tests as described in Section 4.5.3, "To Configure the JavaTest Harness to Run the JACC TCK Tests."

4.5.3 To Configure the JavaTest Harness to Run the JACC TCK Tests

The answers you give to some of the configuration interview questions, 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.

Note that you only need to complete all these steps the first time you start the JavaTest test harness. After you complete these steps, you can run the tests by completing the steps in Section 5.1, "Starting JavaTest."

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

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

2. From the **File** menu, select **Open Quick Start Wizard**.

   The Welcome screen displays.

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

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

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

   You are prompted to select a test suite.

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

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

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

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

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

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

   The JavaTest harness starts running the tests.

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

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

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

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

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

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

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

4.5.4 Modifying the Default Test Configuration

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

- **Group 1**: Available from the JavaTest **Configure/Change Configuration** submenus, the following options are displayed in a tabbed dialog box:

  – Tests to Run

- – Exclude List

- – Keywords

- – Prior Status

- – Test Environment

- – Concurrency

- – Timeout Factor

- **Group 2**: Available from the JavaTest **Configure**/**Change Configuration**/**Other Values** submenu, or by pressing **Ctrl+E**, the following options are displayed in a paged dialog box:

  - – Environment Files

  - – Test Environment

  - – Specify Tests to Run

  - – Specify an Exclude List

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

5

Executing Tests

The JACC TCK uses the JavaTest harness to execute the tests in the test suite. For detailed instructions that explain how to run and use JavaTest, see the *JavaTest User's Guide and Reference* in the documentation bundle.

This chapter includes the following topics:

- Starting JavaTest
- Running a Subset of the Tests
- Running the JACC TCK Against the JACC RI
- Running the JACC TCK Against a Vendor's Implementation
- Test Reports

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.

5.1 Starting JavaTest

There are two general ways to run the JACC TCK using the JavaTest harness software:

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

Note: The `ant` command referenced in the following two procedures refers to the Ant build tool, which is included in the JACC TCK bundle. The `build.xml` file in `<TS_HOME>/bin` contains the various Ant targets for the JACC TCK test suite

---

5.1.1 To Start JavaTest in GUI Mode

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

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

5.1.2 To Run the JACC TCK Tests in Command-Line Mode

The JACC TCK compatibility requirements can be met by performing the TCK tests with an implementation of the following target platforms: an EJB container, a Web container, or both EJB and Web containers as is the case in the Java EE 7 RI. Keywords can be used to indicate in which container specific tests should be run.

The following examples assume that you have already deployed the JACC TCK EJB tests, Web tests, or both sets of tests. See Step 6 in Section 4.2, "Configuring Your Environment to Run the JACC TCK Against Your JACC 1.5 Vendor Implementation," for more information about deploying the JACC TCK tests.

Example 5–1   JACC TCK EJB Tests

To run the JACC TCK EJB tests, enter the following commands:

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

Alternatively, from the <TS_HOME>/src/com/sun/ts/tests/jacc directory, you can also run the JACC EJB tests by using the appropriate keyword:

```
cd <TS_HOME>/src/com/sun/ts/tests/jacc
ant -Dkeywords=jacc_ejb
```

Example 5–2   JACC TCK Web Tests

To run the JACC TCK Web tests, enter the following commands:

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

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

```
cd <TS_HOME>/src/com/sun/ts/tests/jacc
ant -Dkeywords=jacc_web
```

Example 5–3   All JACC TCK Tests

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

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

Alternatively, from the <TS_HOME>/src/com/sun/ts/tests/jacc directory, you can also run the JACC EJB tests by using the appropriate keyword:

```
cd <TS_HOME>/src/com/sun/ts/tests/jacc
ant -Dkeywords=jacc
```

Example 5–4   JACC TCK Signature Tests

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

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

5.2 Running a Subset of the Tests

5.2.1 To Run a Subset of Tests in GUI Mode

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

   The tabbed Configuration Editor dialog box is displayed.

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

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

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

4. Click **Save File**.

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

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

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

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

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

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

5.2.2 To Run a Subset of Tests in Batch Mode Based on Prior Result Status

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

Invoke `ant` with the `priorStatus` property.

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

- `fail`

- `pass`

- `error`

- `notRun`

For example, you could run all the JACC TCK web or ejb tests with a status of failed and error by invoking the following commands:

```
cd <TS_HOME>/src/com/sun/ts/tests/jacc/web|ejb
ant -DpriorStatus="fail,error" runclient
```

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

5.3 Running the JACC TCK Against the JACC RI

This test scenario is ensures that the configuration and deployment of all the JACC TCK tests against the JACC 1.5 Reference Implementation are successful, and that the

TCK is ready for compatibility testing against the Vendor and JACC 1.5 Implementations.

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

2. Run the tests, as described in Section 5.1, "Starting JavaTest," and, if desired, Section 5.2, "Running a Subset of the Tests."

5.4 Running the JACC TCK Against a Vendor's Implementation

This test scenario is one of the compatibility test phases that all Vendors must pass.

1. Verify that you have followed the configuration instructions in Section 4.2, "Configuring Your Environment to Run the JACC TCK Against Your JACC 1.5 Vendor Implementation."

2. Run the tests, as described in Section 5.1, "Starting JavaTest," and, if desired, Section 5.2, "Running a Subset of the Tests."

5.5 Test Reports

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

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

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

5.5.1 Creating Test Reports

5.5.1.1 To Create a Test Report in GUI Mode

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`.

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.

5.5.1.2 To Create a Test Report in Command-Line Mode

Specify where you want to create the test report.

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

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

Reports are written for the last test run to the directory you specify. The default location is /tmp/JTreport.

2. To specify the default report directory, set the report.dir property in <TS_ HOME>/bin/ts.jte.

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

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

   For example:

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

5.5.2 Viewing an Existing Test Report

5.5.2.1 To View an Existing Report in GUI Mode

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

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

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

   The selected report set is opened in the JavaTest ReportBrowser.

5.5.2.2 To View an Existing Report in Command-Line Mode

Use the Web browser of your choice to view the report.html file in the report directory you specified from the command line or in the ts.jte.

6

Debugging Test Problems

There are a number of reasons that tests can fail to execute properly. This chapter provides some approaches for dealing with these failures. Please note that most of these suggestions are only relevant when running the test harness in GUI mode.

6.1 Overview

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

- **Errors**: Tests with errors could not be executed by the JavaTest harness. These errors usually occur because the test environment is not properly configured.

- **Failures**: Tests that fail were executed but had failing results.

The Test Manager GUI provides you with a number of tools for effectively troubleshooting a test run. See the *JavaTest User's Guide* and JavaTest online help for detailed descriptions of the tools described in this chapter.

6.2 Test Tree

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

- **Green**: Passed

- **Blue**: Test Error

- **Red**: Failed to pass test

- **White**: Test not run

- **Gray**: Test filtered out (not run)

6.3 Folder Information

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

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

6.4 Test Information

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

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

- **Test Run Messages** contains a Message list and a Message section that display the messages produced during the test run.

- **Test Run Details** contains a two-column table of name/value pairs recorded when the test was run.

- **Configuration** contains a two-column table of the test environment name/value pairs derived from the configuration data actually used to run the test.

> **Note:** You can set `harness.log.traceflag=true` in `<TS_HOME>/bin/ts.jte` to get more debugging information. Additionally, if you are running the tests from the command line, you can get extra debug information by setting the environment variable HARNESS_DEBUG to true.

6.5 Report Files

Report files are another good source of troubleshooting information. You may view the individual test results of a batch run in the JavaTest Summary window, but there are also a wide range of HTML report files that you can view in the JavaTest ReportBrowser or in the external browser or your choice following a test run. See Section 5.5, "Test Reports," for more information.

6.6 Configuration Failures

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

6.7 Troubleshooting Tips

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

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

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

execution, the `JACCLog.txt` file is read and used for validating that JACC behavior is correct.

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

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

  – `-Dlog.file.location` (this comes from the `ts.jte` property)

  – `-Djavax.security.jacc.policy.provider=com.sun.ts.tests.jacc.provider.T
    SPolicy`

  – `-Djavax.security.jacc.PolicyConfigurationFactory.provider=com.sun.ts.t
    ests.jacc.provider.TSPolicyConfigurationFactoryImpl`

  – `-Dvendor.javax.security.jacc.policy.provider=com.sun.enterprise.securi
    ty.provider.PolicyWrapper`

  – `-Dvendor.javax.security.jacc.PolicyConfigurationFactory.provider=com.s
    un.enterprise.security.provider.PolicyConfigurationFactoryImpl`

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

A

Frequently Asked Questions

This appendix contains the following questions.

- Where do I start to debug a test failure?

- How do I restart a crashed test run?

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

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

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

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

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

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

The JavaTest exclude file (`*.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 the 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.