Oracle®

Technology Compatibility Kit User's Guide for Java API for JSON Processing 1.1

Release 1.1 for Technology Licensees

September 2017

ORACLE®

Technology Compatibility Kit User's Guide for Java API for JSON Processing 1.1, Release 1.1 for Technology Licensees

2013 - 2017,

Primary Author: Eric Jendrock

Contributing Author: Jan Supol

Contributor:

Contents

Preface

This guide describes how to install, configure, and run the Technology Compatibility Kit (TCK) that is used to test the Java API for JSON Processing (JSON-P 1.1) technology.

The JSON-P TCK is designed as a portable, configurable automated test suite for verifying the compatibility of a licensee's implementation of the JSON-P 1.1 Specification (hereafter referred to as the licensee implementation). The JSON-P TCK uses the JavaTest harness version 4.4.1 to run the test suite

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

Refer to the Java 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 JSON-P 1.1 technology to assist them in running the test suite that verifies compatibility of their implementation of the JSON-P 1.1 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 API for JSON Processing Specification.

A link to the specification can be found at `http://jcp.org/en/jsr/detail?id=374`. Before you run the tests in the JSON-P TCK familiarize yourself with the JavaTest documentation that is included in the JSON-P 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 API for JSON Processing TCK (JSON-P TCK 1.1). It also includes a high level listing of what is needed to get up and running with the JSON-P TCK.

This chapter contains the following sections:

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 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 API for JSON Processing 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 EE 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 JSON-P TCK 1.1

The JSON-P TCK 1.1 is designed as a portable, configurable, automated test suite for verifying the compatibility of a licensee's implementation of the JSON-P 1.1 Specification.

1.2.1 JSON-P TCK Specifications and Requirements

This section lists the applicable requirements and specifications.

- **Specification Requirements**: Software requirements for a JSON-P implementation are described in detail in the JSON-P 1.1 Specification. Links to the JSON-P specification and other product information can be found at `http://jcp.org/en/jsr/detail?id=374`.
- **JSON-P Version**: The JSON-P TCK 1.1 is based on the JSON-P Specification, Version 1.1.

- **Reference Runtime**: The designated Reference Runtime for conformance testing of implementations based upon the JSON-P Specification 1.1 is the Oracle Corporation JSON-P 1.1 Reference Implementation (RI).

1.2.2 JSON-P TCK Components

The JSON-P TCK 1.1 includes the following components:

- **JavaTest harness**: Version 4.4.1 of the JavaTest harness and related documentation. See the `README-javatest.html` file, the *JavaTest User's Guide*, and the `ReleaseNotes-javatest.html` file for additional information.
- **JSON-P TCK signature tests**: Check that all public APIs are supported and/or defined as specified in the JSON-P Version 1.1 implementation under test.

The JSON-P TCK tests have been tested with the following:

- JSON-P 1.1 Reference Implementation
- Java Standard Edition (Java SE), Versions 1.8.0

The JSON-P TCK tests run on the following platforms:

- Oracle Linux 7.1
- Windows 10

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.

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 JSON-P TCK 1.1 to test a JSON-P 1.1 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 JSON-P TCK.

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 JSON-P 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 Java API for JSON Processing Version 1.1 Test Appeals Process.

1.2.6 JSON-P TCK Configuration

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

1.3 Getting Started With the JSON-P TCK

This section provides an general overview of what needs to be done to install, set up, test, and use the JSON-P 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 software**: Java Standard Edition (Java SE), Version 1.8.0
   - **JSON-P software**: The implementation of the JSON-P 1.1 specification under test

   See the documentation for each of these software applications for installation instructions.
2. Install the JSON-P TCK 1.1 software.

   See Chapter 3, "Installation" for additional information.
3. Set up the JSON-P 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.
4. Test the JSON-P 1.1 implementation.

   Test the JSON-P implementation installation by running the test suite. See Chapter 5, "Executing Tests" for information about running the test suite.

2

Procedure for Java API for JSON Processing Certification

This chapter describes the compatibility testing procedure and compatibility requirements for Java API for JSON Processing 1.1.

This chapter contains the following sections:

2.1 Certification Overview

The certification process for Java API for JSON Processing Version 1.1 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 Section 2.2, "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 API for JSON Processing Version 1.1 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.
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 API for JSON Processing Version 1.1 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 API for JSON Processing Version 1.1.
Operating Mode	Any Documented option of a Product that can be changed by a user in order to modify the behavior of the Product. For example, an Operating Mode can be binary (enable/disable optimization), an enumeration (select from a list of protocols), or a range (set the maximum number of active threads). Note that an Operating Mode may be selected by a command line switch, an environment variable, a GUI user interface element, a configuration or control file, etc.
Product	A licensee product in which the Technology Under Test is implemented or incorporated, and that is subject to compatibility testing.

Table 2–1 (Cont.) Definitions

Term	Definition
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 API for JSON Processing Version 1.1.
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 API for JSON Processing Version 1.1 Products

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

JSON-P1 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.

JSON-P1.1 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.

JSON-P1.2 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.

JSON-P1.3 An API Definition Product is exempt from all functional testing requirements defined here, except the signature tests.

JSON-P2 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.

JSON-P3 The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

JSON-P4 The Exclude List associated with the Test Suite cannot be modified.

JSON-P5 The Maintenance Lead can define exceptions to these Rules. Such exceptions would be made available to and apply to all licensees.

JSON-P6 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.

JSON-P7 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.

JSON-P8 The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.

2.3 Java API for JSON Processing Version 1.1 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 API for JSON Processing 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 API for JSON Processing TCK support.

If a test is determined to be invalid in function or if its basis in the specification is suspect, the test may be challenged by any licensee of the Java API for JSON Processing 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 API for JSON Processing Version 1.1 TCK Test Appeals Steps"

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

- Tests that are placed on the Exclude List will be placed on the Exclude List within one business day after the determination of test validity. The new Exclude List will be made available to all Java API for JSON Processing TCK licensees on the Java API for JSON Processing 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 API for JSON Processing TCK website.

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

2.3.1 Java API for JSON Processing Version 1.1 TCK Test Appeals Steps

1. **Java API for JSON Processing TCK licensee writes a test challenge to Java Licensee Engineering contesting the validity of one or a related set of Java API for JSON Processing 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 API for JSON

Processing 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 Specifications for Java API for JSON Processing Version 1.1

The Specifications for Java API for JSON Processing are found on the JCP web site at http://jcp.org/en/jsr/detail=?id=374.

2.5 Libraries for Java API for JSON Processing Version 1.1

The following is the list of packages that constitute the required class libraries for Java API for JSON Processing Version 1.1:

javax.json

javax.json.spi

javax.json.stream

3 Installation

This chapter explains how to install the Java API for JSON Processing TCK 1.1 (JSON-P 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.

This chapter contains the following sections:

- Section 3.1, "Obtaining the JSON-P 1.1 Reference Implementation"
- Section 3.2, "Installing the JSON-P Software"

3.1 Obtaining the JSON-P 1.1 Reference Implementation

You can obtain the JSON-P Reference Implementation (RI), Version 1.1 software from the Java Community Process (`http://jcp.org/en/home/index`) web site. The JSON-P RI software is not necessary for running the JSON-P TCK, but it can be useful as a test base for familiarizing yourself with the TCK before testing your own JSON-P implementation.

3.2 Installing the JSON-P Software

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

- Java Standard Edition (Java SE), Version 1.8.0 software
- JSON-P TCK, Version 1.1 software
- An implementation of the JSON-P 1.1 Specification

1. Install the Java SE 8 software, if it is not already installed.

   Download and install the Java SE 8 software from the `http://www.oracle.com/technetwork/java/javase/downloads/index.html` Web site. Refer to the installation instructions that accompany the software for additional information.

2. Install the JSON-P TCK 1.1 software.

   a. Copy or download the JSON-P TCK software to your local system.

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

b. Change to the directory in which you want to install the JSON-P TCK software:

```
cd install_directory
```

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

```
unzip jsonptck-1.1_dd-Mmm-YYYY.zip
```

where *dd* indicates the day, *Mmm* indicates the month, and *YYYY* indicates the year in which the TCK bundle was created. For example, the JSON-P TCK bundle name could be `jsonptck-1.1_15-May-2017.zip`.

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

3. Install the JSON-P implementation to be tested.

Before testing your own JSON-P implementation, it is recommended that you run the JSON-P TCK against the JSON-P RI to familiarize yourself with the JSON-P TCK suite and JavaTest software. See Section 3.1, "Obtaining the JSON-P 1.1 Reference Implementation," for more information.

4

Setup and Configuration

This chapter describes how to set up the JSON-P 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 JSON-P TCK.

4.1 Configuration Overview

Setting up and configuring the JSON-P TCK and JavaTest harness involves three general steps:

- Section 4.2, "Configuring Your Environment to Run the JSON-P TCK"
- Section 4.3, "Using the JavaTest Harness Software"
- Section 4.4, "Using the JavaTest Harness Configuration GUI"

> **Note:** The third general step, Section 4.4, "Using the JavaTest Harness Configuration GUI", is not necessary if you plan on running the JavaTest harness in command-line mode. See Section 5.3, "Running a Subset of the Tests", for more information about running tests in command-line mode.

The remainder of this chapter explains these steps in detail.

4.2 Configuring Your Environment to Run the JSON-P TCK

This section describes how to configure the JSON-P TCK for your environment. After configuring your environment, continue with the instructions in Section 4.3, "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).

1. Set the following environment variables in your shell environment:
   a. `JAVA_HOME` to the directory in which Java SE 8 is installed
   b. `TS_HOME` to the directory in which the JSON-P TCK 1.1 software is installed
   c. `PATH` to include the following directories: `JAVA_HOME/bin` and `<TS_HOME>/tools/ant/bin`
2. Edit the `<TS_HOME>/bin/ts.jte` file, making sure that the following properties are set:
   - `jsonp.classes` to the JAR file(s) that contain your JSON-P 1.1 implementation classes

     For example, if you were using the Oracle JSON-P 1.1 RI, you would set this property to jsonp.classes=javax.json-1.1.0.jar.
   - `jsonp.resources` should *not* be modified

     This property points to JSON resource data files that are used by the JSON-P TCK 1.1.
   - `jsonp.alt.provider.classes` should *not* be modified

     This property is set automatically when running the JSON-P TCK pluggability tests to test the SPI provider interface when using the top-level command line `runclient` or `run.pluggability` Ant targets. The property is included as part of the `ts.run.classpath` property and is placed before `jsonp.classes` in the classpath so the JSON-P TCK can pick up the TCK-supplied provider when set, not the provider supplied with the JSON-P 1.1 implementation under test. This property should only be set when running the pluggability tests. All other tests use the provider supplied with the JSON-P 1.1 implementation under test.

4.3 Using the JavaTest Harness Software

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

- Through the JavaTest GUI; if using this method, please continue on to Section 4.4, "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".

4.4 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 JSON-P TCK test environment. This section covers the following topics:

- Configuration GUI Overview
- Starting the Configuration GUI
- Configuring the JavaTest Harness to Run the JSON-P TCK Tests
- Modifying the Default Test Configuration

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".

4.4.1 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.

4.4.2 Starting the Configuration GUI

Before you start the JavaTest harness software, you must have a valid test suite and Java SE 8 installed on your system.

The JSON-P 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 Configuring the JavaTest Harness to Run the JSON-P 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 Configuring the JavaTest Harness to Run the JSON-P TCK Tests.

4.4.3 Configuring the JavaTest Harness to Run the JSON-P 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 Starting JavaTest or run a subset of the tests by completing the steps in Running a Subset of the Tests.

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

1. Start the JavaTest test harness:

   ```
   ant gui
   ```

   The JavaTest Quick Start screen displays, and you are prompted to **Start**, **Resume**, or **Browse** the test suite.

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 JSON-P TCK is configured to run the default test suite.

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

   The test run is started.

4.4.4 Modifying the Default Test Configuration

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

- **Group 1**: Available from the JavaTest **Configure/Change Configuration** submenus, the following options are displayed in a tabbed dialog box:
  - Tests to Run
  - Exclude List
  - Keywords
  - Prior Status
  - Test Environment
  - Concurrency
  - Timeout Factor

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

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

5 Executing Tests

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

This chapter includes the following topics:

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

5.1 Starting JavaTest

There are two general ways to run the JSON-P TCK using the JavaTest harness software:

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

5.1.1 To Start JavaTest in GUI Mode

Execute the following command:

```
ant gui
```

5.1.2 To Start JavaTest in Command-Line Mode

1. Change to any subdirectory under `<TS_HOME>/src/com/sun/ts/tests`.
2. Execute the `runclient` target to start the JavaTest run:

   ```
   ant runclient
   ```

   For example, to run the JSON-P TCK signature tests, enter the following commands:

   ```
   cd <TS_HOME>/src/com/sun/ts/tests/signaturetest/jsonp
   ```

```
ant runclient
```

5.2 Running All of the JSON-P TCK Tests

Use the following modes to run the JSON-P TCK tests:

- Section 5.2.1, "To Run All of the JSON-P TCK Tests in GUI Mode"
- Section 5.2.2, "To Run All of the JSON-P TCK Tests in Command-Line Mode"

5.2.1 To Run All of the JSON-P TCK Tests in GUI Mode

In GUI mode, the JSON-P TCK tests must be run as a 2-step process. The pluggability tests require a reconfiguration to plug in the `JsonProvider` class that is supplied with the test suite. This class is used by the tests to verify the SPI layer.

Run all of the JSON-P TCK tests except the pluggability tests:

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

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

   The Welcome screen displays.
2. Select Start a new test run, then select **Next**.
3. Enter the test suite (for example, `<TS_HOME>/src`), then select **Next**.
4. Select **Create a new configuration**, then select **Next**.
5. Enter the work directory (for example, `/tmp/JTwork`), then select **Next**.
6. Select **Configure**, then select **Start the configuration editor**, then select **Finish**.

   The Configuration Welcome screen displays.
7. Select **Next**.

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

   You are prompted to specify a test environment.
9. Select either **ts_unix** or **ts_win32**, then select **Next**.

   Select **ts_unix** if you are running the tests in a Unix or Linux environment; select **ts_win32** if you are running the tests in a Windows environment.

   After making your selection and selecting **Next**, you are prompted to specify whether you want to run all or a subset of the test suite.
10. Specify whether you want to run all or a subset of the tests, then select **Next**.

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

    Since the pluggability tests must be run separately from the other tests, select **Yes**.
11. Select the tests you want to run from the displayed test tree, then select **Next**.

    You can select entire branches of the test tree, or use Ctrl+Click or Shift+Click to select multiple tests or ranges of tests, respectively. Select all tests *except* pluggability tests by unchecking the pluggability tests.

12. Specify whether you want to use an exclude list, then select **Next**.

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

    If you select **Yes**, proceed to the next step. If you select **No**, skip to Step 15.

13. Specify the exclude list you want to use, then select **Next**.

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

    If you select **custom**, proceed to the next step. If you select **initial**, skip to Step 14.

14. Specify one or more exclude list files to use, then select **Next**.

15. Select **Done** to accept and save your configuration settings.

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

16. Specify the file in which you want to save your configuration settings, then select **Save File**.

    You are returned to the JavaTest main window.

17. If you want to run the test suite at this time using your current configuration settings, select **Run Tests**, then select **Start** from the main menu.

    The tests in the `<TS_HOME>/src/com/sun/ts/tests/jsonp` and `<TS_HOME>/src/com/sun/ts/tests/signaturetest/jsonp` directory are executed. The pluggability tests, which require additional configuration, will not be run.

Run the pluggability tests, which require a separate test run:

1. Exit the JavaTest GUI.
2. Change to the `<TS_HOME>/bin` directory and reconfigure environment to use the `JsonProvider` class that is supplied with the test suite and restart the JavaTest GUI:

   ```
   cd <TS_HOME>/bin
   ant enable.alternate.jsonp.provider
   ant/bin/ant gui
   ```

3. Select **Configure**, then select **Edit Configuration**.
4. Select **Tests to Run**.
5. Select the tests you want to run from the displayed test tree, then select **Next**.

   Select the pluggability tests only by unchecking all of the other tests.

6. Select **Done**.
7. Select **Run Tests**, then select **Start**.
8. Exit the JavaTest GUI when finished.
9. Disable the alternate JSON provider that you enabled in Step 2, before you initiated the test run for the pluggability tests:

   ```
   cd <TS_HOME>/bin
   ant disable.alternate.jsonp.provider
   ```

5.2.2 To Run All of the JSON-P TCK Tests in Command-Line Mode

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

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

```
cd <TS_HOME>/bin
ant run.all
```

This will execute all of the JSON-P TCK tests, including the pluggability tests.

5.3 Running a Subset of the Tests

Use the following modes to run a subset of the tests:

- Section 5.3.1, "To Run a Subset of the Tests in GUI Mode"
- Section 5.3.2, "To Run a Subset of Tests in Command-Line Mode"
- Section 5.3.3, "To Run an Individual Test in Command-Line Mode"
- Section 5.3.4, "To Run a Subset of Tests in Batch Mode Based on Prior Result Status"

5.3.1 To Run a Subset of the Tests in GUI Mode

1. From the JavaTest main menu, select **Configure**, then **Edit Configuration**.

   The Configuration Editor dialog box is displayed.

2. Select **Specify Tests to Run?** from the option list on the left.

   You are asked whether you want to run all or a subset of the test suite.

3. Select **Yes**, then **Next** to run a subset of tests.

4. Select the tests you want to run from the displayed test tree, then select **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.

5. Save the file, then select **Done**.

   You are returned to the JavaTest main window.

6. Select **Run Tests**, then select **Start** to run the tests you selected.

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

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

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

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

```
ant runclient
```

   The tests in the `<TS_HOME>/src/com/sun/ts/tests/jsonp/api/jsonarraytests` directory and its subdirectories are run.

5.3.3 To Run an Individual Test in Command-Line Mode

1. Change to the directory containing the individual test you want to run.

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

2. Run the test by executing the following command:

```
ant -Dtest.client=Client.java -Dtest=jsonArrayTest1 runclient
```

Just the client test `jsonArrayTest1` in the `jsonarraytests` directory will be run.

5.3.4 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 JSON-P tests with a status of failed and error by invoking the following commands:

```
cd $TS_HOME/src/com/sun/ts/tests/jsonp
ant -DpriorStatus="fail,error" runclient
```

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

5.4 Running the Pluggability Tests

Use the following modes to run the pluggability tests:

- Section 5.4.1, "To Run the Pluggability Tests in GUI Mode"
- Section 5.4.2, "To Run the Pluggability Tests in Command-Line Mode"

5.4.1 To Run the Pluggability Tests in GUI Mode

1. If the JavaTest GUI is running, exit the GUI.
2. Change to the `<TS_HOME>/bin` directory and reconfigure environment to use the `JsonProvider` class that is supplied with the test suite and restart the JavaTest GUI:

   ```
   cd <TS_HOME>/bin
   ant enable.alternate.jsonp.provider
   ant gui
   ```

3. Select **Configure**, then select **Edit Configuration**.
4. Select **Tests to Run**.
5. Select the tests you want to run from the displayed test tree, then select **Next**.

   Select the pluggability tests only by unchecking all of the other tests.
6. Select **Done**.
7. Select **Run Tests**, then select **Start**.
8. Exit the JavaTest GUI when finished.
9. Disable the alternate JSON provider that you enabled in Step 2, before you initiated the test run for the pluggability tests:

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

```
ant disable.alternate.jsonp.provider
```

5.4.2 To Run the Pluggability Tests in Command-Line Mode

There are two ways to run the pluggability tests from the command line. It is highly recommended to use the first method, since it automatically calls the Ant targets to enable and disable the test provider. If you choose to use the second method, you will need to execute those Ant targets manually before and after you run the tests.

The first method is by executing the `run.pluggability` target in Ant:

1. Execute the `run.pluggability` target using Ant:

   ```
   cd <TS_HOME>/bin
   ant run.pluggability
   ```

   The `run.pluggability` target calls Ant targets that enable the test provider, run the pluggability tests, then disable the test provider.

The pluggability tests can also be run an alternate way, albeit one that requires more manual intervention:

1. Execute the `enable.alternate.jsonp.provider` target using Ant:

   ```
   cd <TS_HOME>/bin
   ant enable.alternate.jsonp.provider
   ```

   The `enable.alternate.jsonp.provider` target enables the alternate test provider.

2. Change to the `<TS_HOME/src/com/sun/ts/tests/jsonp/pluggability>` directory:

   ```
   cd <TS_HOME/src/com/sun/ts/tests/jsonp/pluggability>
   ```

3. Start the pluggability test run by executing the following command:

   ```
   ant runclient
   ```

   This will run just the pluggability tests under the `<TS_HOME/src/com/sun/ts/tests/jsonp/pluggability` directory, using a test-supplied `JsonProvider Class` for testing the SPI. All `JsonProvider` method calls will be invoked and verified by the tests to ensure SPI layer is working.

4. Execute the `disable.alternate.jsonp.provider` target using Ant:

   ```
   cd <TS_HOME>/bin
   ant disable.alternate.jsonp.provider
   ```

   The `disable.alternate.jsonp.provider` target disables the alternate test provider.

5.5 Test Reports

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

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

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

5.5.1 Creating Test Reports

Use the following modes to create test reports:

- Section 5.5.1.1, "To Create a Test Report in GUI Mode"
- Section 5.5.1.2, "To Create a Test Report in Command-Line Mode"

5.5.1.1 To Create a Test Report in GUI Mode

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

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

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

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

   You are asked whether you want to view the report now.

3. Select **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 `<TS_HOME>/src/com/sun/ts/tests/signaturetests/jsonp`.

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

Use the following modes to view an existing test report:

- Section 5.5.2.1, "To View an Existing Report in GUI Mode"
- Section 5.5.2.2, "To View an Existing Report in Command-Line Mode"

5.5.2.1 To View an Existing Report in GUI Mode

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

   The selected report set is opened in the JavaTest ReportBrowser.

5.5.2.2 To View an Existing Report in Command-Line Mode

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

This chapter contains the following sections:

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.

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.

6.6 Configuration Failures

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

A

Frequently Asked Questions

This appendix contains the following questions.

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 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.