

Java™ Portlet Technology Compatibility Kit User's Guide

For Technology Licensees

Version 1.0

October 2003

Copyright © 2002 Sun
Microsystems, Inc. All
rights reserved.

Sun, Sun Microsystems, the Sun logo, SunSoft, SunDocs, SunExpress, Solaris, Java, the Java Compatible logo, JavaOS, JavaSoft, JavaStation, the Java Coffee Cup logo, Java Compatibility Kit, JDK, and HotJava are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the United States and other countries. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

Federal Acquisitions: Commercial Software—Government Users Subject to Standard License Terms and Conditions

The product described in this document is distributed under licenses restricting its use, copying, distribution, and decompilation. No part of the product or this document may be reproduced in any form by any means without prior written authorization of the Sun and its licensors, if any.

THIS DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2002 Sun Microsystems, Inc. Tous droits réservés.

Sun, Sun Microsystems, le logo Sun, SunSoft, SunDocs, SunExpress, Solaris, Java, le Java Compatible logo, JavaOS, JavaSoft, JavaStation, le Java Coffee Cup logo, Java Compatibility Kit, JDK, et HotJava sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et d'autre pays. UNIX est une marque enregistree aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company Ltd.

Le produit décrit dans ce document est distribué selon des conditions de licence qui en restreignent l'utilisation, la copie, la distribution et la décompilation. Aucune partie de ce produit ni de ce document ne peut être reproduite sous quelque forme ou par quelque moyen que ce soit sans l'autorisation écrite préalable de Sun et, le cas échéant, de ses bailleurs de licence.

CETTE DOCUMENTATION EST FOURNIE “EN L'ÉTAT”, ET TOUTES CONDITIONS EXPRESSES OU IMPLICITES, TOUTES REPRÉSENTATIONS ET TOUTES GARANTIES, Y COMPRIS TOUTE GARANTIE IMPLICITE D'APTITUDE À LA VENTE, OU À UN BUT PARTICULIER OU DE NON CONTREFAÇON SONT EXCLUES, EXCEPTÉ DANS LA MESURE OÙ DE TELLES EXCLUSIONS SERAIENT CONTRAIRES À LA LOI.

Contents

Preface	5
Who Should Use This Book	5
Before You Read This Book	6
How This Book Is Organized	6
Related Books	7
Typographic Conventions Used in This Book	7
Chapter 1 Introduction	9
Compatibility Testing	9
Why Compatibility Testing is Important	9
TCK Compatibility Rules	10
TCK Overview	10
Java Community Process (JCP) Program and Compatibility Testing	11
The Portlet TCK	11
Portlet TCK Specifications and Requirements	11
Portlet TCK Components	12
JavaTest Harness	12
TCK Compatibility Test Suite	13
Exclude Lists	13
Portlet TCK—Getting Started	14
Chapter 2 Procedure for Portlet 1.0 Certification	17
Certification Overview	17
Compatibility Requirements	17
Definitions	17
Rules for Portlet Products	19
Portlet Test Appeals Process	21

Process Used to Manage Challenges to Portlet 1.0 Tests:	21
Portlet TCK Test Appeals Steps	22
Reference Implementation for Portlet 1.0	24
Specifications for Portlet 1.0	24
Libraries for Portlet 1.0	24
 Chapter 3 Requirements	25
Hardware Requirements	25
Software Requirements	25
 Chapter 4 Installation	27
Obtaining the Software	27
Installing the Software	27
 Chapter 5 Setup and Configuration for Vendor Implementation	29
Allowed Modifications	29
Configure Your Test Environment	29
Overview	30
Configuring the Test Environment	31
Where to Go Next?	37
 Chapter 6 Setup and Configuration for Reference Implementation	39
Allowed Modifications	39
Configure Your Test Environment	39
Overview	40
Configuring the Test Environment	40
 Chapter 7 Running the Portlet TCK	43
Starting JavaTest	43
 Chapter 8 Reporting and Logging	47
Using the JavaTest GUI to Configure and Save a Report	47
 Chapter 9 Debugging Test Problems	49
Using tsant	49
Running Tests without JavaTest	50
Cleaning and Rebuilding Test Areas	51
Listing the Contents of dist/classes Directories	51
Test Tree	52
Folder Information	52

Test Information	53
Report Files	53
Configuration Failures	54
Test Manager Properties	54
Test Suite Errors	54
How Tests are Executed	55
How Tests are Selected for a Test Run	55
 Chapter 10 Troubleshooting	57
Common CTS Problems and Resolutions	57
Support	58
 Appendix A Frequently Asked Questions	59

Preface

This guide describes how to install, configure, and run the Technology Compatibility Kit (TCK) that provides tests for the Java™ Portlet 1.0 technology.

The Portlet TCK is designed as a portable, configurable automated test suite for verifying the compliance of a licensee's implementation of the Portlet 1.0 Specification (hereafter referred to as a licensee implementation). The Portlet TCK uses the JavaTest™ harness version 3.1.3 to run the test suite.

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

Refer to the Java Partner Engineering Web site (<https://javapartner.sun.com>) for answers to frequently asked questions and send questions you may have to your Java Partner Engineering contact. For more information about joining the Java Partner program, please see <http://www.sun.com/software/jpe>.

Who Should Use This Book

This guide is for licensees of Sun Microsystems' and IBM's Portlet 1.0 technology to assist them in running the test suite that verifies compliance of their implementation of the Portlet 1.0 Specification.

Before You Read This Book

Before reading this guide, you should familiarize yourself with the Java programming language and the JSR 168 Specification. A good resource for the Java Programming language is the Sun Microsystems, Inc. Web site, located at:

<http://java.sun.com>

The Portlet TCK 1.0 is based on the JSR 168 Specification. Links to the specification and other product information can be found on the Web at:

<http://www.jcp.org/en/jsr/detail?id=168>

Sun Microsystems recommends that before you run the tests in the Portlet TCK, read and familiarize with the JSR168 Specification and the JavaTest User's Guide, which describes the main JavaTest harness. The JavaTest User's Guide is located at:

[TS_HOME/docs/javatest/javatest.pdf](#)

in the Portlet TCK 1.0 distribution.

How This Book Is Organized

If you are installing and using the Portlet TCK for the first time, Sun Microsystems recommends that you read chapters 1, 2, and 3 completely for the necessary background information, and then perform the steps outlined in chapters 4, 5 or 6, and 7, while referring to chapters 8, 9, 10, and the appendix as necessary.

Chapter 1, “Introduction,” gives an overview of the principles that apply generally to all Technology Compatibility Kits (TCKs) and describes the Portlet TCK. It also includes a listing of the basic steps needed to get up and running with the Portlet TCK.

Chapter 2, “Procedure for Portlet 1.0 Certification,” describes the conformance testing procedure and testing requirements.

Chapter 3, “Requirements,” lists the hardware and software requirements that must be met before the Portlet Compatibility Test Suite can be run.

Chapter 4, “Installation,” explains how to install Portlet TCK on machines that run the Solaris, Linux, and Windows XP/2000 operating systems.

Chapter 5, “Setup and Configuration for Vendor Implementation,” explains how to configure your test environment and the Portlet Compatibility Test Suite, including any special setup instructions that need to be completed prior to deploying and/or running selected tests.

Chapter 6, “Setup and Configuration for Reference Implementation,” explains how to setup and run test suite on the reference implementation.

Chapter 7, “Running the Portlet TCK,” describes how to start and use the Portlet TCK.

Chapter 8, “Reporting and Logging,” explains how status and error information is logged and reported.

Chapter 9, “Debugging Test Problems,” describes several approaches for dealing with tests that fail to execute properly.

Chapter 10, “Troubleshooting,” lists common problems that could be encountered during test execution and explains how to resolve these problems.

Appendix A, “Frequently Asked Questions,” provides answers to frequently asked questions.

Related Books

- *JavaTest User’s Guide* and JavaTest online help (located in the `doc/javatest` directory in the Portlet TCK distribution)
- *The Java™ Architecture for Portlet Specification, Version 1.0*
(<http://java.sun.com/portlet/index.html>)
- *The Java Programming Language*
(<http://java.sun.com/docs/books/javaprog/>)
- *The Java Language Specification Second Edition*
(<http://java.sun.com/docs/books/jls/>)
- *The Java Virtual Machine Specification 2nd Edition, Java 2 Platform*
(<http://java.sun.com/docs/books/vmspec/>)

Typographic Conventions Used in This Book

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

Table 1 Typographic Conventions Used in This Book

Typeface or Symbol	Meaning	Example
Courier AaBbCc123	The names of commands, files, and directories, variable and method names; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name%</code> You have mail.
Bold AaBbCc123	What you type, contrasted with on-screen computer output	<code>machine_name%</code> su Password:
<i>ABCDE</i>	Command-line placeholder: replace with a real name or value	Example 1: To delete a file, type rm <i>filename</i> Example 2: <i>TS_HOME</i>
<i>Italics</i> AaBbCc123	Book titles, new words or terms, or words to be emphasized or set apart from the other text	Read Chapter 4 in <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be root to do this.
\	Indicates that a long line has been broken in two, typically to improve legibility, particularly in code	<code>jjava classname \</code> <i>[options for classname]</i>

NOTE	The top-most Portlet TCK installation directory is referred to as <i>TS_HOME</i> throughout the Portlet TCK documentation.
-------------	--

Introduction

This chapter provides an overview of the principles that apply generally to all Technology Compatibility Kits (TCKs) and describes the Java™ Portlet Technology Compatibility Kit (Portlet TCK). It also includes a listing of what is needed to get up and running with the Portlet TCK.

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.

Why Compatibility Testing is Important

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

- Compatibility testing is the means by which Sun Microsystems ensures that the Java platform does not become fragmented as it is ported to different operating systems and hardware environments.

- Compatibility testing benefits developers working in the Java programming language, allowing them to write applications once and then to deploy them across heterogeneous computing environments without porting.
- Compatibility testing allows application users to obtain applications from disparate sources and deploy them with confidence.
- Compatibility testing benefits Java platform implementors by ensuring a level playing field for all Java platform ports.

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 Portlet 1.0 Certification.”](#)

TCK Overview

A TCK is a set of tools and tests used to verify that a licensee’s implementation of Sun Microsystems’s and IBM’s Portlet 1.0 technology conforms to the applicable specification. All tests in the TCK are based on the written specifications for the Java platform. A TCK tests compatibility of a licensee’s implementation of Sun Microsystems and IBM Portlet 1.0 technology to the applicable specification of the technology. Compatibility testing is a means of ensuring correctness, completeness, and consistency across all implementations developed by Sun Microsystems and IBM Portlet 1.0 technology licensees.

The set of tests included with each TCK is called the “test suite.” All tests in the Portlet TCK’s test suite are self-checking and do not require tester interaction. Most tests return either a Pass or Fail status. For a given licensee’s implementation to be certified, all of the required tests must pass.

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 from the Java Partner Web site (<https://javapartner.sun.com>). For more information about joining the Java Partner program, please see <http://www.sun.com/software/jpe>.

Java Community Process (JCP) Program and Compatibility Testing

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

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

For further information on the JCP program see this URL: <http://www.jcp.org>.

The Portlet TCK

The Portlet TCK 1.0 is designed as a portable, configurable, automated test suite for verifying the compliance of a licensee's implementation with Sun Microsystems and IBM's Portlet 1.0 technology.

Portlet TCK Specifications and Requirements

This section lists the applicable requirements and specifications.

- **Specification Requirements** – Software requirements for a Portlet implementation are described in detail in the Java™ Portlet Specification. Links to the JSR 168 specification and other product information can be found at <http://www.jcp.org/en/jsr/detail?id=168>.
- **Java™ Portlet Specification Version** – The Portlet TCK 1.0 is based on the JSR 168 Specification, Version 1.0.
- **JavaTest™ Harness** – The Portlet TCK requires version 3.1.3 of the JavaTest harness.
- **Reference Implementation** – The designated Reference Implementation for conformance testing of implementations based upon JSR 168 Specification 1.0 is Apache Open Source Project, Pluto 1.0.

Portlet TCK Components

The Portlet TCK 1.0 includes the following components:

- JavaTest harness and supplemental java classes, which are used to select and run TCK tests and generate reports that show the results of test runs
- TCK interview component, which is used to configure a test run
- The test suite, which is a collection of tests and supplemental files that provide data for the automatic running of tests through the JavaTest harness
- An exclude list, which provides a list of tests that your implementation is not required to pass
- TCK documentation, including JavaTest readme file, Release Notes, *JavaTest User's Guide*, and *Portlet TCK User's Guide*

JavaTest Harness

The JavaTest harness version 3.1.3 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 on-line within the JavaTest GUI, or off-line 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
- Environment configuration

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 6](#), “[Setup and Configuration for Reference Implementation.](#)”

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 Portlet TCK 1.0 to test an implementation of the JSR 168 specification. The tests are designed to verify that a licensee's implementation of the technology complies with the appropriate specification. The individual tests correspond to assertions of the specification.

The descriptions of tests that make up the TCK compatibility test suite are precompiled into a special file. When the JavaTest is started it loads the file and extracts information about the tests. Depending on how a test run is configured some tests can be filtered out.

The Portlet TCK 1.0 test suite comprises two test categories:

- A **signature test** that checks that all of the public APIs are supported in the Portlet, Version 1.0 implementation that is being tested.
- **API/SPEC tests** for the public APIs and language elements that are defined by the JSR 168 Specification.

Exclude Lists

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

A licensee is not required to pass any test—or even run any test—on the Exclude List.

The Exclude List file included in the Portlet TCK is located in the `TS_HOME/bin` directory. For example, the Exclude List for version 1.0 of the Portlet TCK is `TS_HOME/bin/ts.jtx`.

NOTE The exclude list that is shipped with the bundle reflects the state of the release at the time of the release. Since that time, additional tests may have been added to the Exclude List. All updates to the Exclude List are made available on the Java Partner Engineering Web site. You should always make sure you are using an up-to-date copy of the Exclude List before running the Portlet TCK to verify your implementation.

A test might be included in an 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 also tested against the Reference Implementation done as an Apache Open Source Project, Pluto 1.0. 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 “Portlet Test Appeals Process” on page 21.
-------------	--

Portlet TCK—Getting Started

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

1. Make sure that the following software has been correctly installed:
 - Sun Microsystems J2SE software version 1.3.1 or higher on the system hosting the JavaTest harness.
 - The implementation of Portlet 1.0 that is under test.

Consult the documentation for each of these software applications for installation instructions.

2. Install the Portlet TCK 1.0 software.

See [Chapter 3, “Requirements”](#) and [Chapter 4, “Installation”](#) for more information.

Note that JavaTest, version 3.1.3 is bundled with the Portlet TCK and is installed as a part of the Portlet TCK installation.

3. Configure the Portlet TCK.

See [Chapter 5, “Setup and Configuration for Vendor Implementation”](#) or [Chapter 6, “Setup and Configuration for Reference Implementation”](#) for more information.

4. Run the tests.

See [Chapter 7, “Running the Portlet TCK”](#) for more information. See also [Chapter 8, “Reporting and Logging”](#) and [Chapter 9, “Debugging Test Problems”](#) as necessary.

Procedure for Portlet 1.0 Certification

This chapter describes the compatibility testing procedure and compatibility requirements for Java™ Portlet Technology Compatibility Kit (Portlet TCK), version 1.0.

Certification Overview

- 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 Java Partner Engineering that you have finished testing and that you meet all the compatibility requirements.

Compatibility Requirements

Definitions

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

Table 2-1 Definitions

Term	Definition
Computational Resource	A piece of hardware or software that may vary in quantity, existence, or version, which may be required to exist in a minimum quantity and/or at a specific or minimum revision level so as to satisfy the requirements of the Test Suite. Examples of computational resources that may vary in quantity are RAM and file descriptors. Examples of computational resources that may vary in existence (this is, may exist or not) 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.
Container	An implementation of the associated Libraries, as specified in the Specifications, and a version of a J2SE Runtime Product, as specified in the Specifications, or a later version of a J2SE Runtime Product that also meets these compatibility requirements.
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 SM (JCP SM), for the Technology Under Test. The Libraries for Portlet 1.0 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 JCP member responsible for maintaining the Specification, reference implementation, and TCK for the Technology. Sun Microsystems and IBM are the Maintenance Leads for Portlet 1.0.

Table 2-1 Definitions (*Continued*)

Term	Definition
Operating Mode	Any Documented option of a Product that can be changed by a user in order to modify the behavior of the Product. For example, an Operating Mode of a Runtime can be binary (enable/disable optimization), an enumeration (select from a list of localizations), or a range (set the initial Runtime heap size).
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 Runtime supporting an Operating Mode that permits selection of an initial heap size might have a Product Configuration that sets the initial heap size to 1 Mb.
Resource	A Computational Resource, a Location Resource, or a Security Resource.
Rules	These definitions and rules in this Compatibility Requirements section of this User's Guide.
Runtime	The Containers specified in the Specifications.
Security Resource	A security privilege or policy necessary for the proper execution of the Test Suite. For example, the user executing the Test Suite will need the privilege to access the files and network resources necessary for use of the Product.
Specifications	The documents produced through the JCP that define a particular Version of a Technology. The Specifications for the Technology Under Test can be found later in this chapter.
Technology	Specifications and a reference implementation produced through the JCP.
Technology Under Test	Specifications and the reference implementation for Portlet 1.0.
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 JCP.

Rules for Portlet Products

For each version of an operating system, software component, and hardware platform Documented as supporting the Product:

- PLT1. 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.

- PLT1.1. 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.

- PLT1.2. 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.

- PLT2. Some Conformance Tests may have properties that may be changed. Properties that can be changed are identified in the JavaTest Environment (.jte) files in the bin directory of the Test Suite installation. Apart from changing such properties and other allowed modifications described in this User's Guide, 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 Partner Engineering web site and apply to all licensees.

- PLT3. The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

- PLT4. The Exclude List associated with the Test Suite cannot be modified.

- PLT5. The Maintenance Lead can define exceptions to these Rules. Such exceptions would be made available to and apply to all licensees.

- PLT6. 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.
- PLT7. 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 in 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.
- PLT7.1. 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 subsetting, supersetting, or modifications to the APIs of the included Technologies are allowed.
- PLT8. Except for tests specifically required by this TCK to be recompiled (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.
- PLT9. The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.

Portlet Test Appeals Process

Sun has a well established process for managing challenges to its Portlet 1.0 Test Suite and plans to continue using a similar process in the future. Sun, as Maintenance Lead, will authorize representatives from Sun's 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 Portlet 1.0 TCK support.

Process Used to Manage Challenges to Portlet 1.0 Tests:

The following process will be used to manage challenges to Portlet 1.0 tests:

1. Who can make challenges to the Portlet 1.0 tests?

Only licensees of the Portlet 1.0 TCK

2. What challenges to the Portlet 1.0 tests may be submitted?

Individual (or related) tests may be challenged for reasons such as:

- Test is buggy (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.

3. How are challenges submitted?

To the Sun Java Partner Engineering contact assigned by Sun to the licensee. Appeals must be in writing as described in the Test Challenge form below.

4. How and by whom are challenges addressed?

Sun Java Partner Engineering coordinates the review and decisions made by test development and specification development engineers. See the Portlet 1.0 TCK Test Appeals Steps below.

5. How are approved changes to the Portlet 1.0 tests managed?

All tests found to be invalid are placed on the Exclude List for that version of the Portlet 1.0 TCK within 1 business day. The new Exclude List is published to all Portlet 1.0 TCK licensees on the Java Partner Engineering web site. Sun, as Maintenance Lead, has the option of creating an alternative test to address any challenge. Alternative tests (and criteria for their use) will be placed on the Java Partner Engineering web site. Note that passing an alternative test is deemed equivalent with passing the original test.

Portlet TCK Test Appeals Steps

1. Portlet licensee writes a test challenge to the Maintenance Lead contesting the validity of one or a related set of Portlet tests.

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

2. The Maintenance Lead evaluates the challenge.

If the appeal is incomplete or unclear, it is returned to the submitting licensee for correction. If all is in order, the Maintenance Lead will check with the test developers to review the purpose and validity of the test before writing a response. The Maintenance Lead 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), the Maintenance Lead 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, the Maintenance Lead places the tests on the Exclude List for that version of the Portlet (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 on the Java Partner Engineering web site. If the test is valid but difficult to pass due to hardware or operating system limitations, the Maintenance 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 the Maintenance Lead to resolve the issue and only involve the EC as a last resort.

Code 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)
```

Code 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)  
-can be iterative-  
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 appropriate?)
```

Reference Implementation for Portlet 1.0

Designated Reference Implementation for compatibility testing of Portlet 1.0 is as follows:

- Reference Implementation done as an Apache Open Source Project, Pluto 1.0
- Java™ 2 Platform, Standard Edition (J2SE™) Versions 1.3.1_02, 1.3.1_03, and 1.4
- Redhat Linux 7.2, Solaris™ Operating System V 8 and 9/SPARC, and Microsoft Windows 2000 and XP.

Specifications for Portlet 1.0

The following web sites contain the Specifications for Java™ Portlet 1.0:

<http://www.jcp.org/en/jsr/detail?id=168>

Libraries for Portlet 1.0

The following packages constitute the required class libraries for Java™ Portlet 1.0:

- javax.portlet

Requirements

This chapter lists the required hardware configurations and prerequisite software that must be present before you can run the Java™ Portlet Technology Compatibility Kit (Portlet TCK), version 1.0.

Hardware Requirements

You can run the Portlet TCK software on compatible Java™ platforms on both Sun workstations and on personal computers. The following section lists the hardware requirements for both the TCK and the reference implementation. Hardware requirements for other implementations will vary.

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

- CPU running at 500 MHz minimum
- 512MB of RAM minimum
- 1024MB of swap space minimum
- 512 MB of free disk space minimum for writing data to log files
- Network access

Software Requirements

You can run the Portlet TCK software on Sun Solaris™ operating system, Linux, and Windows XP/2000 platforms that meet the following minimum software requirements:

- Operating Systems:

- Sun Solaris V 8 and 9
- Microsoft Windows 2000 Professional or Microsoft Windows XP Pro
- Redhat Linux 7.1
- Java™ 2 Platform, Standard Edition (J2SE™) SDK: Version 1.3.1-02 or later
- Java™ 2 Platform, Enterprise Edition (J2EE™) SDK Version 1.3
- JSR 168 platform software, such as a vendor's implementation

Installation

This chapter explains how to install the Portlet TCK on a system running the Sun Solaris™, Redhat Linux, or Microsoft Windows operating system.

Obtaining the Software

You can obtain the Portlet TCK, version 1.0 software from the Java Partner Engineering web site (<http://javapartner.sun.com>) in the web site's Download Center area. The reference implementation Pluto 1.0 can be obtained from the following URL:

<http://jakarta.apache.org/pluto>

Installing the Software

1. Install the Java™ 2 Platform, Standard Edition (J2SE™) 1.3.1 or higher software.

Download the J2SE 1.3.1 or higher software from the Java Software web site and install. See the installation instructions that accompany the software for additional information.

2. Install a Portlet 1.0-compliant portlet container, such as the reference implementation done as an Apache Open Source Project, Pluto 1.0.

See the installation instructions that accompany the software for additional information.

3. Download and install the Sun Microsystems J2EE SDK 1.3.1 software from the Java software web site (http://java.sun.com/j2ee/sdk_1.3).

See the installation instructions that accompany the software for additional information. This is needed only if you wish to recompile the TCK software for debugging purposes.

4. Install the Portlet TCK, version 1.0 software.

Download the Portlet TCK software from the Java Partner Engineering web site and install:

- a. Copy or download the `portlet-1_0-tck.zip` file to the directory on your local system where you will install the Portlet TCK.

You can obtain the Portlet TCK 1.0 software from the Java Partner Engineering web site (<http://javapartner.sun.com>).

- b. Change to the `<install_directory>` directory and use the `unzip` command to extract the bundle:

```
cd <install_directory>
unzip portlet-1_0-tck.zip
```

NOTE The `<install_directory>/portlettck` directory will be `TS_HOME`.

Setup and Configuration for Vendor Implementation

This chapter describes a generic Portlet TCK configuration to work with a vendor implementation. See [Chapter 6, “Setup and Configuration for Reference Implementation”](#) for information on running Portlet TCK on the reference implementation done as an Apache Open Source Project, Pluto 1.0.

Allowed Modifications

You can modify the following test suite components only:

- Your implementation of the porting package
- The `build.properties` environment file
- The `ts.jte` environment file

Configure Your Test Environment

This section includes the following sections:

- Overview
- Configuring the Test Environment

Overview

This section provides an overview of the general procedure for configuring the vendor implementation and the Portlet TCK. To get the TCK running on a vendor's implementation, install and bring up the vendor implementation and configure and set up the Portlet TCK.

For TCK configuration, the steps are the same for the Sun Solaris™, Microsoft Windows 2000/XP, and Linux operating systems. For detailed information on these steps, see [“Configuring the Test Environment” on page 31](#).

1. Setup a JSR 168 compliant implementation/server.

See [“Setup a JSR 168 Compliant Implementation on a Server” on page 31](#) for more information on modifications that must be made to the vendor's server.

2. Setup the TCK environment variables.

See [“Setting Up TCK Environment Variables” on page 31](#) for more details.

3. Setup the TCK `TS_HOME/bin/build.properties` file to match your test environment.

See [“Setting Up TCK Properties in build.properties” on page 31](#) for more details.

4. Deploy TCK Portlet Web Applications on server under test.

Portlet TCK includes portlet web applications containing servlets, portlets, and JSPs to be deployed on the implementation being tested. See [“Deploying Portlet Web Applications on the Server” on page 32](#) for more details.

5. Modify TCK `TS_HOME/bin/ts.jte` properties.

See [“Setting Up TCK Properties in ts.jte” on page 32](#) for more details.

6. Optionally, set up a user-identity for any test requiring authentication.

See [“Properties to Configure TCK for User Identification” on page 35](#) for more information.

7. Optionally, set up HTTP headers that might be set by a vendor for every request.

See [“Setting Up Additional HTTP Headers” on page 37](#) for more information.

8. Run the TCK tests.

See [Chapter 7, “Running the Portlet TCK”](#) for information on Starting JavaTest™ and running the tests.

Configuring the Test Environment

This section provides detailed instructions for configuring your test environment.

Setup a JSR 168 Compliant Implementation on a Server

Set up a JSR 168 compliant implementation on a server such as the Reference Implementation done as an Apache Open Source Project, Pluto 1.0.

Setting Up TCK Environment Variables

Set the following environment variables in your shell environment:

1. Set the `JAVA_HOME` variable to the directory where the Java™ 2 Platform, Standard Edition (J2SE™) software has been installed.
2. Set the `TS_HOME` variable to the directory where the Portlet TCK software has been installed.
3. Set the `PATH` variable to include the `<TS_HOME>/bin` directory.

For example, on UNIX using `csh`:

```
rivendell% setenv JAVA_HOME=/usr/java1.3
rivendell% setenv TS_HOME /portlettck
rivendell% setenv PATH $TS_HOME/bin:$PATH
```

Or, on Windows:

```
C:\> set JAVA_HOME=C:\java1.3
C:\> set TS_HOME=C:\portlettck
C:\> set PATH=%TS_HOME%\bin;%PATH%
```

Setting Up TCK Properties in build.properties

Set the following properties in the `TS_HOME/bin/build.properties` file:

1. Set the `j2ee.home.ri` property to the installation directory of the J2EE SDK 1.3.1 software.

2. Set the `harness.executeMode` property to Mode 2 (Run Only) to indicate the mode in which the JavaTest harness will run the tests.

Do not specify any other mode other than Mode 2.

3. Set the `webapp.dir` to a directory, where all TCK test portlet WAR files are copied when `tsant deploy.all` command is run.

See “[Deploying Portlet Web Applications on the Server](#)” on page 32 for more details on the deployment of the portlet WAR files.

4. Set `portlet.classes` property to include the JAR file that contains the Portlet 1.0 API (called `portlet.jar` if you are using Pluto 1.0.)

Deploying Portlet Web Applications on the Server

1. Configure the property `webapp.dir` in `build.properties` file.

Set the property to a location where the `tsant` command can copy all the web applications that need to be deployed in the implementation under test.

The directory must exist before running this command. For example, type:

```
webapp.dir=/tmp/PortletTCKWebApps
```

2. Change to the `TS_HOME/bin` directory. For example, type:

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

3. Deploy the Portlet web applications in the location specified by `webapp.dir` property. To deploy, type:

```
tsant deploy.all
```

4. Deploy all the web applications on the implementation under test using vendor specific deployment tools or procedures.

5. Restart the server.

Restart the server only if your deployment tools or procedures require a server restart.

Setting Up TCK Properties in `ts.jte`

Before running any of the Portlet TCK tests, you must specify certain information that JavaTest needs to run the tests in your specific environment. This information exists in the `<TS_HOME>/bin/ts.jte` environment file. This file contains sets of name/value pairs that are used by the tests. You need to assign a valid value for your environment for all of the properties listed in the following sections.

Properties for Test Harness Setup

Make sure that the following properties, which are used by the test harness, have been set in the `ts.jte` file:

```
harness.temp.directory=${TS_HOME}
harness.log.port=2000
harness.log.traceflag=[true|false]
```

Here:

- The `harness.temp.directory` property specifies a temporary directory that the harness creates and to which the TCK harness and tests write temporary files. The default setting need not be changed.
- The `harness.log.port` property specifies the port that server components of the tests use to send logging output back to JavaTest. If the default port is not available on the machine running JavaTest, you must edit this property and set it to an available port. By default, this property is set to port 2000.
- The `harness.log.traceflag` property is used to turn on or turn off verbose debugging output for the tests. The value of the property is set to `false` by default. Set the property to `true` to turn debugging on.

Properties for Configuring TCK to Obtain URLs to Invoke Portlet

In TCK, every test is written as a set of one or more portlets. A test client is written for each test. The test client must interact with a portal page containing the portlets that are part of the test.

To accomplish this, TCK needs to obtain the initial URL for the portal page of each test case. All the portlets in the portal page obtained with the initial URL must be in VIEW Portlet mode and in NORMAL Window state. Subsequent requests to the test are done using URLs generated by PortletURL that are part of the returned portal pages. These subsequent requests must be treated as directed to same portal page composed of the same portlets.

Since aggregation of portlets in a portal page and the URLs used to interact with the portlets are vendor specific, TCK provides two alternative mechanisms in the framework to get the URLs to portal pages for the test cases: declarative configuration or programmatic configuration. A vendor must support at least one of these mechanisms to run the conformance tests.

Declarative Configuration of the Portal Page for a TCK Test

TCK publishes an XML file in `<TS_HOME>/bin/portletTCKTestCases.xml` containing the portlets for each test case. The `portletTCKTestCases.xml` file contains the unique name of each test in the Portlet TCK. It also contains the portlets participating in each test. It follows the rules specified in XML schema, `<TS_HOME>/bin/portletTCK_1_0.xsd`. In this guide, the `portletTCKTestCases.xml` file is referred to as the Master Test Cases file.

For the declarative configuration, vendors must refer to this file for establishing a portal page for every test. Vendors must provide an XML file with a complete URL for the portal page for each test. A call to this URL must generate a portal page with the content of all the portlets defined for the corresponding test case. If redirected to another URL, the new URL must use the same host name and port number as specified in the original URL.

More information about this file is included in TCK Chapter of JSR 168 Specification. The file provided by the vendor would be validated against the XML schema file `<TS_HOME>/bin/portletTCKVendor_1_0.xsd`. For reference, a sample file, `TS_HOME/bin/vendorTestsToURLMapping.xml`, is included.

This configuration option must be specified in the `ts.jte` file using the following two properties:

```
portalURLFetcherMode=0
vendorTestsToURLMappingFile=/tmp/sunTestsToURLMapping.xml
```

Here:

- `portalURLFetcherMode` - Set this property to 0.
- `vendorTestsToURLMapping` - Set this property to the full path name of the vendor supplied file containing URLs for all test cases.

Programmatic Configuration of the Portal Page for a TCK Test

For programmatic configuration, a vendor must provide a full URL as a configuration parameter to the TCK. The TCK will call this URL with a set of parameters indicating the set of portlets that must appear in a portal page for the given test.

Upon receiving this request, the vendor provided URL can dynamically create a portal page with the required portlets. Calls to this vendor provided URL are always HTTP GET requests.

The parameter names on the URL are multiple occurrences of the `portletName` parameter. Values of this parameter must be a string consisting of application name and portlet name supported by a forward slash (/).

The response of this call must be a portal page with the required portlets or a redirection to another URL where the portal page will be served. If redirected, the new URL must use the same host and port number as the original URL.

More information on this option is available in TCK Chapter in the JSR168 Specification document.

This configuration option is specified in the `ts.jte` file using the following two properties:

```
portalURLFetcherMode=1
vendorPortalURL=http://hostname.domain:port/portal/tckServlet
```

- `portalURLFetcherMode` - Set this property to 1.
- `vendorPortalURL` - Set this property to the URL to be used by TCK.

Properties to Configure TCK for User Identification

Some of the Portlet TCK tests may require an authenticated user. A portal vendor may indicate that certain additional test cases, not required by TCK, to be executed in the context of an authenticated user. This is useful for vendor implementations that require an authenticated user for certain functionality to work. A vendor can specify the names of these test cases in a configuration file, `TS_HOME/bin/authTestList.txt`. TCK will consult this file to decide if user authentication is needed for a test case.

Authentication can be enabled by setting the `authConfigType` property to a value of 1 or 2 in `ts.jte` file. Unique names of the test can be found in the `portletTCKTestCases.xml` file. This file is consulted only if authentication is enabled.

Portlet TCK provides two mechanisms to set up user authentication and send the user credentials:

- HTTP Basic authentication
- Java interface provided by the TCK

If the TCK framework requires no authentication, the `authConfigType` property can be set to 0. A value of 0 implies that no authentication is needed for any test. For example:

```
authConfigType=0
```

If the TCK framework is configured to use HTTP Basic authentication mechanism, an `Authorization` HTTP header, using the configured user and password values, is constructed and sent with each test case request. This configuration option is specified in the `ts.jte` file in the `authConfigType` property.

To enable HTTP Basic authentication, set the `authConfigType` property to a value of 1. If `authConfigType` property is set to a value of 1, also set the `authuser` and `authpassword` properties with the user name and password. The `authuser` and `authpassword` properties need to be set only if `authConfigType` property is set to a value of 1 or 2.

```
authConfigType=1
authuser=username
authpassword=password
```

If TCK framework is configured to use the Java interface mechanism, the value obtained from the specified interface implementation will be sent as a `Cookie` HTTP header with request of the test case.

This configuration option is specified in the `ts.jte` file in the `authConfigType` property. To enable authentication via the Java interface, set the `authConfigType` property to a value of 2. If `authConfigType` property is set to a value of 2, set the `authuser`, `authpassword` properties.

The `TS_HOME/src/com/sun/ts/lib/porting` directory contains the interfaces. In this directory, we have `TSPortletAuthCookie` interface. Set `porting.ts.portletAuthCookie.class` to `com.vendor` (for example, `porting.ts.portletAuthCookie.class=com.vendor`)

Set the `porting.ts.portletAuthCookie.class` property to the vendor implementation of `TSPortletAuthCookie` interface. Append the classpath to this implementation in the `local.classes` property in the `build.properties` file. For example:

```
authConfigType=2
authuser=username
authpassword=password
porting.ts.portletAuthCookie.class=com.vendor.MYTSAuthCookieImplementation
```

Javadoc for this interface can be found in the *TS_HOME/docs/api* directory.

Setting Up Additional HTTP Headers

Vendors can set up HTTP headers in the *bin/headerlist.properties* file. These headers are sent out in every request that goes out to the server. The syntax in the file should follow the java Properties. TCK framework writes some of the headers, such as Host, User-Agent. Any such headers specified in this file will be overwritten by the framework.

Do not specify any cookie headers in header file.

Optional Policy Setting

The *TS_HOME/bin/server_policy.append* file contains the grant statement used by the test harness, signature tests, and portlet tests. Append this file to the Java policy files on your portlet container.

Where to Go Next?

This completes the configuration of the Portlet TCK for a vendor implementation. In order to configure the Portlet TCK for Pluto 1.0, see [Chapter 6, “Setup and Configuration for Reference Implementation.”](#) Otherwise, proceed to [Chapter 7, “Running the Portlet TCK”](#) for instructions on running the Portlet TCK against your implementation.

Where to Go Next?

Setup and Configuration for Reference Implementation

This chapter describes a generic Portlet TCK configuration to work with Apache Open Source Project, Pluto 1.0. See [Chapter 5, “Setup and Configuration for Vendor Implementation”](#) for information on running Portlet TCK on a vendor implementation.

Allowed Modifications

You can modify the following test suite components only:

- Your implementation of the porting package
- The `build.properties` environment file
- The `ts.jte` environment file

Configure Your Test Environment

This section includes the following sections:

- Overview
- Configuring the Test Environment

Overview

This section provides an overview of the general procedure for configuring the reference implementation and the Portlet TCK. To get the TCK running on Pluto 1.0, install and bring up Pluto 1.0 and configure and set up the Portlet TCK.

For TCK configuration, the steps are same for the Sun Solaris™, Microsoft Windows 2000/XP, and Linux operating systems. For detailed information on these steps, see [“Configuring the Test Environment” on page 40](#).

1. Set up a Pluto 1.0.

See [“Setup Pluto 1.0” on page 40](#) for more information on modifications that must be made to the Pluto 1.0.

2. Setup the TCK environment variables.

See [“Setting Up TCK Environment Variables” on page 41](#) for more details.

3. Setup the TCK `<TSHOME>/bin/build.properties` file to match your test environment.

See [“Setting Up TCK Properties in build.properties” on page 41](#) for more details.

4. Deploy TCK Portlet Web Applications on server under test.

Portlet TCK includes portlet web applications containing servlets, portlets, and JSPs to be deployed on the implementation being tested. See [“Deploying Portlet Web Applications on Pluto 1.0” on page 41](#) for more details.

5. Modify TCK `<TSHOME>/bin/ts.jte` properties.

See [“Setting Up TCK Properties in ts.jte” on page 42](#) for more details.

6. Run the TCK tests.

See [Chapter 7, “Running the Portlet TCK”](#) for information on Starting JavaTest™ and running the tests.

Configuring the Test Environment

This section provides detailed instructions for configuring your test environment.

Setup Pluto 1.0

Set up the JSR 168 compliant Pluto 1.0 according to the Pluto 1.0 reference implementation documentation.

Setting Up TCK Environment Variables

Set the following environment variables in your shell environment:

1. Set the `JAVA_HOME` variable to the directory where the Java™ 2 Platform, Standard Edition (J2SE™) software has been installed.
2. Set the `TS_HOME` variable to the directory where the Portlet TCK software has been installed.
3. Set the `PATH` variable to include the `<TS_HOME>/bin` directory.

For example, on UNIX using `csh`:

```
rivendell% setenv JAVA_HOME=/usr/java1.3
rivendell% setenv TS_HOME /portlettck
rivendell% setenv PATH $TS_HOME/bin:$PATH
```

Or, on Windows:

```
C:\> set JAVA_HOME=C:\java1.3
C:\> set TS_HOME=C:\portlettck
C:\> set PATH=%TS_HOME%\bin;%PATH%
```

Setting Up TCK Properties in build.properties

Set the following properties in the `<TS_HOME>/bin/build.properties` file:

1. Set the `j2ee.home.ri` property to the installation directory of the J2EE SDK 1.3.1 software.
2. Set the `webapp.dir` to a directory, where all TCK test portlet wars are copied when `tsant deploy.all` command is run.

See [“Deploying Portlet Web Applications on Pluto 1.0” on page 41](#) for more details on the deployment of the portlet WAR files.

Deploying Portlet Web Applications on Pluto 1.0

1. Set `webapp.dir` in `TS_HOME/bin/build.properties` to `PLUTO_HOME/portlets` where `PLUTO_HOME` is the directory where Pluto 1.0 is installed.

2. Change to *TS_HOME/bin* directory and type the following command to copy all the TCK portlet WAR files to the location specified in *webapp.dir*:

```
tsant deploy.all
```

3. Change to the *PLUTO_HOME/build* directory and type the following command:

```
build.bat deploy_portlets
```

4. Restart Pluto 1.0.
5. Ensure that portlets are deployed correctly. To ensure, type the following URL in a browser:

```
http://hostname:8080/pluto/tck?portletName=portlet_jp_RenderRequest_web/GetAttribute
```

Setting Up TCK Properties in *ts.jte*

The Pluto 1.0 uses programmatic configuration of the portal page for a TCK test. Its mechanism is defined in detail in [“Programmatic Configuration of the Portal Page for a TCK Test” on page 34](#). For Pluto 1.0, *ts.jte* is configured with default setup needed to run TCK on Pluto 1.0.

```
portalURLFetcherMode=1  
vendorPortalURL=http://localhost:8080/pluto/tck
```

- *portalURLFetcherMode* - Set this property to 1.
- *vendorPortalURL* - Set this property to the URL to be used by TCK.

Running the Portlet TCK

The Portlet Compatibility Test Suite 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.

NOTE	For consistent results, reset any vendor specific persistent state before running the Portlet TCK. Also restart your portlet container.
-------------	---

Before starting the JavaTest, ensure that you have completed the steps outlined in [Step 1](#) through [Step 5 on page 44](#) for a vendor implementation or [Step 1](#) through [Step 5 on page 44](#) for Pluto 1.0.

Starting JavaTest

There are two general ways to run the Portlet test using the JavaTest harness software:

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

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

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

NOTE The `tsant` command referenced in this guide is a wrapper around the Ant build tool, which is included in the TCK bundle. The `build.xml` file in `TS_HOME/bin` contains the various Ant targets for the TCK test suite.

► **To Start the JavaTest in GUI Mode**

1. Set `TS_HOME` to the directory where Portlet TCK was installed.
2. Change directory to `TS_HOME/bin`.
3. Execute the `tsant gui` target to start the JavaTest GUI:

```
<TS_HOME>/bin/tsant gui
```

4. Select Create Work Directory in the Welcome screen.
5. Specify the directory to which the JavaTest test harness will write temporary files (for example, `/tmp/JTwork`) and select Next Work Dir.

For debugging purposes, after running the test, look at the files created in this directory for each test case.

6. Select Configuration from the pull-down list and select Edit Configuration.
The Welcome screen is displayed.
7. Select Next Question button from the Welcome screen.
8. Accept the default location of the environment files and select Next Question.
9. Specify your test environment and select Next Question.

Your test environment must be `ts_unix` or `ts_win32`. That is:

- Select the `ts_unix` option for the Solaris or Linux platform.
 - Select the `ts_win32` option for the Microsoft Windows XP or Windows 2000 Professional platform.
10. Accept the default set of tests to run and select Next Question.
 11. Agree to the default exclude list and select Next Question.
 12. Verify the default location of the exclude list and select Next Question.
 13. Specify where you wish to save this configuration information.
 14. Select Finished in the Congratulations screen.

15. Select Run and then select Start.

The tests will be executed.

NOTE You will lose all your default settings if you run the Parameter Editor before running the Configuration Editor.

➤ **To Start JavaTest in Command-Line Mode**

1. Set <TS_HOME> to the directory in which Portlet TCK was installed.
2. Change to any subdirectory under <TS_HOME>/src/com/sun/ts/tests.
3. Execute the `tsant runclient` target to start the JavaTest run:

`<TS_HOME>/bin/tsant runclient`

4. To run a single test directory, type:

```
cd
<TS_HOME>/src/com/sun/ts/tests/portlet/api/javax_portlet/Portlet
Session
tsant runclient
```

5. To run a single test within a test directory, type:

```
cd
<TS_HOME>/src/com/sun/ts/tests/portlet/api/javax_portlet/RenderR
equest
tsant runclient -Dtest=GetAttributeTest
```

This command will only run the `GetAttributeTest` in the `RenderRequest` test directory. Select the test name to run by looking at the `testName` tags in the `URLClient.java` file.

If the test is in `SpecURLClient.java`, specify the following additional parameter:

`-Dtest.client=SpecURLClient.java`

6. To run a subset of test directories, type:

```
cd <TS_HOME>/src/com/sun/ts/tests/portlet/api  
tsant runclient
```

This command will run all the test directories under the `api` directory.

Reporting and Logging

This chapter describes how to use the JavaTest™ Graphical User Interface (GUI) to access detailed configuration, output, and result information for the tests that were executed during the test run.

Using the JavaTest GUI to Configure and Save a Report

Use the following procedure to use the JavaTest GUI to configure and save a report for a specific test area:

1. Start the JavaTest GUI:

```
cd TS_HOME/bin tsant gui  
tsant gui
```

2. Configure the test area that will be tested.
 - a. Using the configure pull-down menu, select Edit Configuration.
 - b. A window will pop up with a message stating that a work directory is required. Select Create Work Directory, unless you already configured a work directory. Then select Open Work Directory.
 - c. The Configuration Editor window will be displayed.
3. Use the Configuration Editor to set up the test harness:
 - a. The Welcome screen displays. Press Next.
 - b. Specify your Environment File. By default *TS_HOME*/bin/ts.jte is selected. Press Next.
 - c. Specify your Test Environment. Select *ts_unix* or *ts_win32*. Select Next.

- d. Specify which test to run, then select Yes. Press Next.
 - e. Go down the through the folders in the tree structure and locate the `<TS_HOME>/com/sun/ts/tests` directory. Highlight the folder of the test area you want to run (appclient, for example). You can specify a single test as well by going further down the directory tree. Press Next.
 - f. Specify an Exclude List. Select Yes. Select Next.
 - g. Select the exclude list to use. Leave the default (initial). Select Next.
 - h. The Congratulations screen is displayed. Select Done.
 - i. The Save File screen is displayed. Save your configuration.
4. Run the test area (appclient, for example). Use the Run Test pull-down menu and select Start.
 5. Once the test(s) have finished running, you can save the test results to a file:
 - a. Use the Report pull-down menu and select New Report.
 - b. Enter the directory name in the Directory field (appclient_results, for example).
 - c. Leave the Filter selection on Correct Configuration.
 - d. You can view the Report, which is in HTML format, through a Browser or through the JavaTest GUI.

Debugging Test Problems

For final certification and branding, all tests must be run through the JavaTest™ test harness. There are a number of reasons that tests can fail to execute properly. This chapter provides some approaches for dealing with these failures.

You can execute different `tsant` targets during your build and debug cycle.

Using tsant

The following sections will address how to use `tsant` with the following targets to rebuild, list, and run tests:

- `runclient`
- `clean`
- `build`
- `ld, lld, lc, llc`

NOTE Licensees *can only* run the Sun-built version of the tests for certification.

You need to set `TS_HOME` and `JAVA_HOME` before using `tsant`. And unless `tsant` is in your path, you'll need to use the following command to execute `tsant`:

```
$TS_HOME/bin/tsant target
```

To see all of the targets you can use, look at the `TS_HOME/bin/build.xml` file.

Running Tests without JavaTest

You can use the `runclient` target to run tests outside the JavaTest harness during the debugging process. That is, you need to execute the following command to run the tests without running the JavaTest GUI:

```
tsant -v runclient
```

If you include the `-v` switch, you will see additional information that should help you debug test failures.

To run the PORTLET TCK tests using `tsant`, complete the following steps:

To run a single test directory:

1. Go to the directory where you want to run the tests. For example:

```
cd
TS_HOME/src/com/sun/ts/tests/portlet/api/javax_portlet/RenderRe
quest
```

2. Type `tsant runclient`.

This will run all tests in the `RenderRequest` test directory.

To run a single test within a test directory, type the following:

1. Go to the directory where you want to run the tests. For example:

```
cd
TS_HOME/src/com/sun/ts/tests/jaxrpc/api/javax_portlet/RenderReq
uest
```

2. Type `tsant runclient -Dtest=GetAttributeTest`

This will run only the `GetAttributeTest` in the `RenderRequest` test directory. You select the test name to run by looking at the `testName` tags in the `URLClient.java` file. If the test is in `SpecURLClient.java`, you would need to give an additional parameter `-Dtest.client=SpecURLClient.java`

To run a subset of test directories type the following:

1. Go to the directory where you want to run the tests. For example:

```
cd TS_HOME/src/com/sun/ts/tests/portlet/api
```

2. Type `tsant runclient`

This will run all the test directories under the `api` directory.

Additionally, the `harness.log.traceflag` property is used to turn on or turn off verbose debugging output for the tests. The value of the property is set to `false` by default. Set the property to `true` to turn debugging on.

Note that you must run the tests through the JavaTest harness for final certification.

Cleaning and Rebuilding Test Areas

You can use the `clean` and `build` targets to clean and rebuild selected test areas during the debugging process. That is, you need to execute the following command to clean test areas before rebuilding:

```
tsant clean
```

Then, to rebuild, execute the following command:

```
tsant build
```

Before cleaning and rebuilding, go to the directory that you want to rebuild.

When you are debugging test failures, you may find it helpful to modify tests. If you modify the tests, you must rebuild them before rerunning the tests. Note that you can only run the Sun-built version of the tests for final certification.

Listing the Contents of dist/classes Directories

You can use the `ld`, `lld`, `lc`, and `llc` targets to list the contents of corresponding `dist/classes` directories from the `src` directory without leaving the `src` directory. All listings are sorted by modification time, with the most recent modification listed first. Output is redirected to `more`. The format may vary on Windows and Unix. `tsant` does not support changing directory into the `dist/classes` directories, but you can copy and paste the first line of the output, which is the target path.

The list targets are as follows:

- `tsant ld` lists the contents of the `dist` directory
- `tsant lld` provides a long list of the contents of the `dist` directory
- `tsant lc` lists the contents of the `classes` directory
- `tsant llc` provides a long list of the contents of the `classes` directory

If you run these targets in a directory that is not under the `src` directory, they will list the contents of the current directory. Note that you should not combine targets when using `ld`, `lld`, `lc`, or `llc` targets.

Test Tree

Use the test tree 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)

When you click a folder icon in the test tree, the JavaTest harness displays its folder view in the Test Manager information area. The information in the folder view is changed by the view filter. Refer to the *JavaTest User's Guide* or the JavaTest online help for a detailed description of the test tree.

Folder Information

Click a folder icon in the test tree to display its information in the information pane. The folder view displays a Summary tab, five status tabs, and a status display.

During a test run, you can use the folder view to monitor the status of a folder and its tests. You can also use the folder view during troubleshooting to quickly locate and open individual tests that had errors or failed during the test run.

The Summary tab displays the number of tests by their test results in a folder. In addition to Summary information, the folder view contains five status tabs that group and list the tests by their results. Choose the Error and the Failed panes to view the lists of all tests in and under a folder that were not successfully run. You can double-click a test in the lists to view its test information.

The status display at the bottom of the pane displays messages about the selected tab. The messages may indicate that tests in the folder are loading or may provide detailed status information about a selected test.

Refer to the *JavaTest User's Guide* or the JavaTest online help for a detailed description of the folder view.

Test Information

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

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

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

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

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

The Test Description tab displays the name/value pairs contained in the test description. The values specified in the test description fields are used by the JavaTest harness when the tests are run.

The Files tab contains a drop down list of source files from the test description. You can display the contents of a file by clicking its name in the list. XML documents (*.xml files) used by the test are not initially in the list because they are not the sources of the test but its input data. To view an XML document, click the appropriate reference when you browse the HTML file with the test description.

Report Files

Report files are another good source of troubleshooting information. The JavaTest harness does not automatically generate reports at the end of a test run, but can easily be configured to do so. Refer to [Chapter 8, “Reporting and Logging”](#) for information about generating test reports.

Configuration Failures

Configuration failures are more difficult to correct. They are easily recognized because many tests fail the same way. When all of 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 launch problems when tests are not run, individual test output is not available. Other places to look for errors include:

Check your `.jti` file settings in the Configuration Editor. You may have launched the JavaTest harness with “prior status” set to “any” instead of “ignore.”

Although you may not produce an `env.html` report file, the JavaTest GUI provides the ability to view evaluated variables used by the tests. Choose **Configure>Show Test Environment** from the Test manager menu bar to view the contents of the test environment.

The Configuration Editor generates a configuration Question Log when you complete a configuration interview. You can use the configuration Question Log to review all of the questions and your answers in the current configuration. Choose **Configure>Show Question Log** from the Test Manager menu bar to view the current configuration interview.

Test Manager Properties

You can view the properties of a test manager by choosing **View>Properties** from the Test Manager menu bar. The Test Manager Properties dialog box contains Test Suite, Work Directory, Configuration, and Plug-Ins information.

Refer to the *JavaTest User's Guide* or the JavaTest online help for a detailed description of the Test Manager Properties dialog box.

Test Suite Errors

If the JavaTest harness detects test suite errors, it displays an advisory dialog box. You can view detailed information about the test suite errors by choosing **View>Test Suite Errors** from the Test Manager menu bar.

Refer to the *JavaTest User's Guide* or the JavaTest online help for a detailed description of the Test Manager: Test Suite Errors dialog box.

How Tests are Executed

To better understand and debug the test failures, this section describes how the tests are selected for a test run and how they are then executed.

Test execution results are reported as one of the three states:

- **Pass** – A test passes when the functionality being tested behaves as expected. All tests are expected to pass.
- **Fail** – A test fails when the functionality being tested does not behave as expected.
- **Error** – A test is considered to be in error when something (usually a configuration problem) keeps the test from being executed as expected. Errors often indicate a systematic problem—a single configuration problem can cause many tests to fail. For example, if the path to the Java runtime is configured incorrectly, no tests can run and all will be in error.

How Tests are Selected for a Test Run

Immediately prior to the start of a test run, the JavaTest harness selects tests for the run based on the following factors:

- **Tests to be Run** – The JavaTest harness finds tests listed in the “Tests to be Run” field of the JavaTest Configuration. You can specify sub-branches of the tree as a way of limiting which tests are executed during a test run. The JavaTest harness walks the tree starting with the sub-branches or tests you specify and executes all tests that it finds.
- **Exclude List** – Tests listed in the appropriate exclude list are “deselected” prior to the start of a test run. For details about exclude lists and their role in the certification process, see [Chapter 2, “Procedure for Portlet 1.0 Certification.”](#)
- **Prior Status** – Use the combo box and check boxes to select tests in a test run based on their outcome on a prior test run. Prior status is evaluated on a test-by-test basis using information stored in result files (`.jtr`) written in the work directory.

Troubleshooting

This chapter explains how to debug test failures that you could encounter as you run the Portlet Compatibility Test Suite.

Common CTS Problems and Resolutions

This section lists common problems that you may encounter as you run the Portlet Compatibility Test Suite software on the J2EE SDK. It also proposes resolutions for the problems, where applicable.

Problem The following exception may occur when a Portlet TCK test tries to write a very long tracelog:

```
java.lang.StringIndexOutOfBoundsException: String index out of
range:
-13493
at java.lang.String.substring(String.java:1525)
at java.lang.String.substring(String.java:1492)
at javax.sqe.javatest.TestResult$Section
$WritableOutputBuffer.write(TestResult.java:650)
at java.io.Writer.write(Writer.java:153)
at java.io.PrintWriter.write(PrintWriter.java:213)
at java.io.PrintWriter.write(PrintWriter.java:229)
at java.io.PrintWriter.print(PrintWriter.java:360)
at java.io.PrintWriter.println(PrintWriter.java:497)
at javax.sqe.javatest.lib.ProcessCommand
$StreamCopier.run(ProcessCommand.java:331)
```

Resolution Set the `-Djavatest.maxOutputSize=nnn` system parameter in the `TS_HOME/bin/ts.jte` file to a value that is higher than the default setting of 100,000 on the JavaTest VM.

Support

If, after completing the troubleshooting process explained in this chapter, you still cannot resolve your test problems, please contact the Java Partner Engineering representative that has been assigned to your account.

Frequently Asked Questions

Q: Where do I start to debug a test failure?

A: 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 9, “Debugging Test Problems”](#) for more information.

Q: How do I restart a crashed test run?

A: 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.

Q: Why are there so many tests in the Exclude List?

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

- An error in Pluto 1.0 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.

