

TCK User's Guide for Technology Implementors

Table of Contents

Eclipse Foundation	1
Preface	2
Who Should Use This Book	2
Before You Read This Book	2
Typographic Conventions	2
Shell Prompts in Command Examples	3
1 Introduction	4
1.1 Compatibility Testing	4
1.2 About the TCK	6
1.3 Getting Started With the TCK	10
2 Procedure for Certification	12
2.1 Certification Overview	12
2.2 Compatibility Requirements	12
2.3 Test Appeals Process	18
2.4 Specifications for Jakarta XML Binding	20
2.5 Libraries for Jakarta XML Binding	20
3 Installation	22
3.1 Obtaining a Compatible Implementation	22
3.2 Installing the Software	22
4 Setup and Configuration	24
4.1 Configuring the XML Binding TCK	24
4.2 Unused Section	39
4.3 Unused Section	39
4.4 Custom Configuration Handlers	39
4.5 Custom Deployment Handlers	40
4.6 Using the JavaTest Harness Software	41
4.7 Using the JavaTest Harness Configuration GUI	42
5 Executing Tests	46
5.1 Starting JavaTest	46
5.2 Running a Subset of the Tests	47
5.3 Running the TCK Against another CI	49
5.4 Running the TCK Against a Vendor's Implementation	49
5.5 Test Reports	50
6 Debugging Test Problems	52
6.1 Overview	52
6.2 Test Tree	53

6.3 Folder Information	53
6.4 Test Information	53
6.5 Report Files	54
6.6 Configuration Failures	54
6.7 Test Manager Properties	55
6.8 Test Suite Errors	55
6.9 How Tests are Executed	55
A Frequently Asked Questions	58
A.1 Where do I start to debug a test failure?	58
A.2 How do I restart a crashed test run?	58
A.3 What would cause tests be added to the exclude list?	58

Eclipse Foundation

Technology Compatibility Kit User's Guide for Jakarta XML Binding

Release 3.0 for Jakarta EE

September 2020

Technology Compatibility Kit User's Guide for Jakarta XML Binding, Release 3.0 for Jakarta EE

Copyright © 2017, 2020 Oracle and/or its affiliates. All rights reserved.

This program and the accompanying materials are made available under the terms of the Eclipse Public License v. 2.0, which is available at <http://www.eclipse.org/legal/epl-2.0>.

SPDX-License-Identifier: EPL-2.0

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

References in this document to JAXB refer to the Jakarta XML Binding unless otherwise noted.

Preface

This guide describes how to install, configure, and run the Technology Compatibility Kit (TCK) that is used to test the Jakarta XML Binding (XML Binding 3.0) technology.

The XML Binding TCK is a portable, configurable automated test suite for verifying the compatibility of a vendor's implementation of the XML Binding 3.0 Specification (hereafter referred to as the vendor implementation or VI). The XML Binding TCK uses the JavaTest harness version 5.0 to run the test suite



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.

Jakarta EE is a community sponsored and community run program. Organizations contribute, along side individual contributors who use, evolve and assist others. Commercial support is not available through the Eclipse Foundation resources. Please refer to the Eclipse EE4J project site (<https://projects.eclipse.org/projects/ee4j>). There, you will find additional details as well as a list of all the associated sub-projects (Implementations and APIs), that make up Jakarta EE and define these specifications. If you have questions about this Specification you may send inquiries to jaxb-dev@eclipse.org. If you have questions about this TCK, you may send inquiries to jaxb-dev@eclipse.org.

Who Should Use This Book

This guide is for vendors that implement the XML Binding 3.0 technology to assist them in running the test suite that verifies compatibility of their implementation of the XML Binding 3.0 Specification.

Before You Read This Book

You should be familiar with the XML Binding 3.0, version 3.0 Specification, which can be found at <https://jakarta.ee/specifications/xml-binding/3.0/>.

Before running the tests in the XML Binding TCK, you should familiarize yourself with the JavaTest documentation which can be accessed at the [JT Harness web site](#).

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. <code>machine_name% *su*</code> <code>Password:</code>
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 <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name% you have mail.</code>
<i>Italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.	Read Chapter 6 in the <i>User's Guide</i> . Do <i>not</i> save the file. The command to remove a file is <code>rm filename</code> .

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	<code>machine_name%</code>
C shell for superuser	<code>machine_name#</code>
Bourne shell and Korn shell	<code>\$</code>
Bourne shell and Korn shell for superuser	<code>#</code>
Bash shell	<code>shell_name-shell_version\$</code>
Bash shell for superuser	<code>shell_name-shell_version#</code>

1 Introduction

This chapter provides an overview of the principles that apply generally to all Technology Compatibility Kits (TCKs) and describes the Jakarta XML Binding TCK (XML Binding 3.0 TCK). It also includes a high level listing of what is needed to get up and running with the XML Binding TCK.

This chapter includes the following topics:

- [Compatibility Testing](#)
- [About the TCK](#)
- [Getting Started With the TCK](#)

1.1 Compatibility Testing

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

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

Compatibility test development for a given feature relies on a complete specification and compatible implementation (CI) for that feature. Compatibility testing is not primarily concerned with robustness, performance, nor ease of use.

1.1.1 Why Compatibility Testing is Important

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

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

all Jakarta 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 Certification."](#)

1.1.3 TCK Overview

A TCK is a set of tools and tests used to verify that a vendor's compatible implementation of a Jakarta EE technology conforms to the applicable specification. All tests in the TCK are based on the written specifications for the Jakarta EE platform. A TCK tests compatibility of a vendor's compatible 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 version of this TCK.

1.1.4 Jakarta EE Specification Process (JESP) Program and Compatibility Testing

The Jakarta EE Specification Process (JESP) program is the formalization of the open process that has been used since 2019 to develop and revise Jakarta EE technology specifications in cooperation with the international Jakarta EE community. The JESP program specifies that the following three major components must be included as deliverables in a final Jakarta EE technology release under the direction of the responsible Expert Group:

- Technology Specification
- Compatible Implementation (CI)
- Technology Compatibility Kit (TCK)

For further information about the JESP program, go to Jakarta EE Specification Process community page <https://jakarta.ee/specifications>.

1.2 About the TCK

The XML Binding TCK 3.0 is designed as a portable, configurable, automated test suite for verifying the compatibility of a vendor's implementation of the XML Binding 3.0 Specification.

1.2.1 TCK Specifications and Requirements

This section lists the applicable requirements and specifications.

- **Specification Requirements:** Software requirements for a XML Binding implementation are described in detail in the XML Binding 3.0 Specification. Links to the XML Binding specification and other product information can be found at <https://jakarta.ee/specifications/xml-binding/3.0/>.
- **XML Binding Version:** The XML Binding 3.0 TCK is based on the XML Binding Specification, Version 3.0.
- **Compatible Implementation:** One XML Binding 3.0 Compatible Implementation, Eclipse Implementation of JAXB is available from the Eclipse EE4J project (<https://projects.eclipse.org/projects/ee4j>). See the CI documentation page at <https://github.com/eclipse-ee4j/jaxb-ri> for more information.

See the XML Binding TCK Release Notes for more specific information about Java SE version requirements, supported platforms, restrictions, and so on.

1.2.2 TCK Components

The XML Binding TCK 3.0 includes the following components:

- JavaTest harness version 5.0 and related documentation. See the [ReleaseNotes-jtharness.html](#) file and the [JT Harness web site](#) for additional information.
- XML Binding TCK signature tests; check that all public APIs are supported and/or defined as specified in the XML Binding Version 3.0 implementation under test.
- If applicable, an exclude list, which provides a list of tests that your implementation is not required to pass.
- API tests for all of the XML Binding API in all related packages:
 - [jakarta.xml.bind](#)
 - [jakarta.xml.bind.annotation](#)
 - [jakarta.xml.bind.annotation.adapters](#)
 - [jakarta.xml.bind.attachment](#)

- `jakarta.xml.bind.helpers`
- `jakarta.xml.bind.util`

The XML Binding TCK tests run on the following platforms:

- CentOS Linux 7

1.2.3 JavaTest Harness

The JavaTest harness version 5.0 is a set of tools designed to run and manage test suites on different Java platforms. To JavaTest, Jakarta EE can be considered another platform. The JavaTest harness can be described as both a Java application and a set of compatibility testing tools. It can run tests on different kinds of Java platforms and it allows the results to be browsed online within the JavaTest GUI, or offline in the HTML reports that the JavaTest harness generates.

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

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

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

1.2.4 TCK Compatibility Test Suite

The test suite is the collection of tests used by the JavaTest harness to test a particular technology implementation. In this case, it is the collection of tests used by the XML Binding TCK 3.0 to test a XML Binding 3.0 implementation. The tests are designed to verify that a vendor'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 vendor's compatible implementation is not required to pass or run any test on the Exclude List. The Exclude List file, `<TS_HOME>/lib/jaxb_tck23.jtx`, is included in the XML Binding TCK.



From time to time, updates to the Exclude List are made available. The exclude list is included in the Jakarta TCK ZIP archive. Each time an update is approved and released, the version number will be incremented. You should always make sure you are using an up-to-date copy of the Exclude List before running the XML Binding 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 compatible implementations. Any tests that fail when run on a compatible Jakarta 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.



Vendors are not permitted to alter or modify Exclude Lists. Changes to an Exclude List can only be made by using the procedure described in [Section 2.3.1, "TCK Test Appeals Steps."](#)

1.2.6 TCK Configuration

You need to set several variables in your test environment, modify properties in the `<TS_HOME>/lib/jaxb_tck23.jte` file, and then use the JavaTest harness to configure and run the XML Binding tests, as described in [Chapter 4, "Setup and Configuration."](#)

1.2.7 XML Binding Technology Overview

The XML Binding TCK 3.0 test suite comprises six test categories:

- Schema to Java mapping tests
- Java to schema mapping tests
- XML Binding framework API tests
- Content tree tests
- XML-to-Java name convention tests
- XML validation tests

Schema Compiler Tests

The schema compiler tests verify the following:

1. The schema compiler finds errors in invalid schema.
2. The schema compiler successfully compiles valid schema and the signatures of generated Java classes are appropriate.
3. The schema compiler appropriately compiles schema annotated with XML Binding custom binding declarations:
 - Successfully compiles valid source schema with well-formed binding declarations.
 - Fails to compile valid source schema with non-well-formed binding declarations.

XML Binding Framework API Tests

The XML Binding framework API tests cover public classes and interfaces defined in the `javax.xml.bind` package and sub-packages.

Content Tree Tests

The content tree tests test the classes generated by the XML Binding schema compiler, including:

1. Unmarshalling valid documents: the documents must be unmarshalled and validated correctly.
2. Marshalling documents: invokes marshalling for valid documents and checks that no errors are reported.
3. Content tree evaluating: check data in content tree after unmarshalling.
4. Content tree validating: modify data in content tree to make the tree invalid and, if the modification does not report an error, then enable XML Binding 3.0 validation on Marshaller, and verify that Marshaller reports the error.

XML-to-Java Name Conversion Tests

The XML-to-Java name conversion tests check that XML-to-Java name conversion rules are implemented correctly and support most of the Unicode characters that are allowed for XML names.

Schema Generator Tests

The schema generator tests verify that:

1. The schema generator supports all the Java Types to XML mapping.
2. The schema generator appropriately maps Java program elements to XML Schema constructs.
3. The schema generator appropriately maps Java program elements that are customized using mapping annotations and program annotations that are based on JSR-175 program annotation facility:
 - Successfully generates valid target schema from correctly annotated Java program element.
 - Fails to process annotated Java program elements that have constraint violations as specified in the [Java Types to XML](#) section of the XML Binding 3.0 specification and [javax.xml.bind.annotation](#) javadocs.
 - The schema generator successfully produces a well-formed schema; all documents expected to be valid against generated schema are valid and all documents expected to be invalid against generated schema are invalid.

XML Validation Tests

XML Validation tests are designed to check schema validation when a mechanisms other than JAXP is used. Note that XML validation tests are mandatory when XML validation is not performed via JAXP.



The Jakarta EE Specification Process support multiple compatible implementations. These instructions explain how to get started with the Eclipse Implementation of JAXB CI. If you are using another compatible implementation, refer to material provided by that implementation for specific instructions and procedures.

1.3 Getting Started With the TCK

This section provides an general overview of what needs to be done to install, set up, test, and use the XML Binding 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 8
 - A CI for XML Binding 3.0. One example is Eclipse Implementation of JAXB.

- XML Binding TCK version 3.0, which includes:
 - (no additional libraries)
- The XML Binding 3.0 Vendor Implementation (VI)

See the documentation for each of these software applications for installation instructions. See [Chapter 3, "Installation,"](#) for instructions on installing the XML Binding TCK.

 1. Set up the XML Binding TCK software.

See [Chapter 4, "Setup and Configuration,"](#) for details about the following steps.

 1. Set up your shell environment.
 2. Modify the required properties in the `<TS_HOME>/lib/jaxb_tck23.jte` file.
 3. Configure the JavaTest harness.
 2. Test the XML Binding 3.0 implementation.

Test the XML Binding implementation installation by running the test suite. See [Chapter 5, "Executing Tests."](#)

2 Procedure for Certification

This chapter describes the compatibility testing procedure and compatibility requirements for Jakarta XML Binding. This chapter contains the following sections:

- [Certification Overview](#)
- [Compatibility Requirements](#)
- [Test Appeals Process](#)
- [Specifications for Jakarta XML Binding](#)
- [Libraries for Jakarta XML Binding](#)

2.1 Certification Overview

The certification process for XML Binding 3.0 consists of the following activities:

- Install the appropriate version of the Technology Compatibility Kit (TCK) and execute it in accordance with the instructions in this User's Guide.
- Ensure that you meet the requirements outlined in [Compatibility Requirements](#) below.
- Certify to the Eclipse Foundation that you have finished testing and that you meet all of the compatibility requirements, as required by the Eclipse Foundation TCK License.

2.2 Compatibility Requirements

The compatibility requirements for XML Binding 3.0 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	<p>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.</p> <p>Examples of computational resources that may vary in quantity are RAM and file descriptors.</p> <p>Examples of computational resources that may vary in existence (that is, may or may not exist) are graphics cards and device drivers.</p> <p>Examples of computational resources that may vary in version are operating systems and device drivers.</p>
Configuration Descriptor	Any file whose format is well defined by a specification and which contains configuration information for a set of Java classes, archive, or other feature defined in the specification.
Conformance Tests	All tests in the Test Suite for an indicated Technology Under Test, as released and distributed by the Eclipse Foundation, excluding those tests on the published Exclude List for the Technology Under Test.
Container	An implementation of the associated Libraries, as specified in the Specifications, and a version of a Java Platform, Standard Edition Runtime Product, as specified in the Specifications, or a later version of a Java Platform, Standard Edition 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, released and distributed by the Eclipse Foundation, that are not required to be passed to certify conformance. The Jakarta EE Specification Committee may add to the Exclude List for that Test Suite as needed at any time, in which case the updated TCK version supplants any previous Exclude Lists for that Test Suite.
Libraries	<p>The class libraries, as specified through the Jakarta EE Specification Process (JESP), for the Technology Under Test.</p> <p>The Libraries for Jakarta XML Binding are listed at the end of this chapter.</p>

Term	Definition
Location Resource	<p>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.</p> <p>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.</p>
Maintenance Lead	<p>The corresponding Jakarta EE Specification Project is responsible for maintaining the Specification, and the TCK for the Technology. The Specification Project Team will propose revisions and updates to the Jakarta EE Specification Committee which will approve and release new versions of the specification and TCK.</p>
Operating Mode	<p>Any Documented option of a Product that can be changed by a user in order to modify the behavior of the Product.</p> <p>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).</p> <p>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.</p>
Product	<p>A vendor's product in which the Technology Under Test is implemented or incorporated, and that is subject to compatibility testing.</p>
Product Configuration	<p>A specific setting or instantiation of an Operating Mode.</p> <p>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.</p>
Rebuildable Tests	<p>Tests that must be built using an implementation-specific mechanism. This mechanism must produce specification-defined artifacts. Rebuilding and running these tests against a known compatible implementation verifies that the mechanism generates compatible artifacts.</p>
Resource	<p>A Computational Resource, a Location Resource, or a Security Resource.</p>
Rules	<p>These definitions and rules in this Compatibility Requirements section of this User's Guide.</p>
Runtime	<p>The Containers specified in the Specifications.</p>

Term	Definition
Security Resource	<p>A security privilege or policy necessary for the proper execution of the Test Suite.</p> <p>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.</p>
Specifications	<p>The documents produced through the Jakarta EE Specification Process (JESP) that define a particular Version of a Technology.</p> <p>The Specifications for the Technology Under Test are referenced later in this chapter.</p>
Technology	Specifications and one or more compatible implementations produced through the Jakarta EE Specification Process (JESP).
Technology Under Test	Specifications and a compatible implementation for Jakarta XML Binding Version 3.0.
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 Jakarta EE Specification Process (JESP).
Schema Compiler	A schema compiler generates Java source files or compiled classes that represent XML data described by W3C XML Schema recommendation [XSD Part 1][XSD Part 2].
Schema Compiler Output	The output of a Schema Compiler intended to be run in combination with the Libraries.
Schema Generator	A software development tool that maps a set of existing Java sources or class files to Schema Generator Output. The mapping is described by program annotations. The Schema Generator can be invoked directly by a programmer without programming.
Schema Generator Output	The output of a Schema Generator intended to be a W3C XML Schema defined in the XML Schema recommendation [XSD Part 1][XSD Part 2]. The output of a Schema Generator that is in the format defined by the W3C XML Schema recommendation [XSD Part 1][XSD Part 2].

2.2.2 Rules for Jakarta XML Binding Products

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

XmlBinding1 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.

XmlBinding1.1 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.

XmlBinding1.2 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.

XmlBinding1.3 An API Definition Product is exempt from all functional testing requirements defined here, except the signature tests.

XmlBinding2 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 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 will be provided via the Jakarta EE Specification Project website and apply to all vendor compatible implementations.

XmlBinding3 The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

XmlBinding4 The Exclude List associated with the Test Suite cannot be modified.

XmlBinding5 The Maintenance Lead can define exceptions to these Rules. Such exceptions would be made available as above, and will apply to all vendor implementations.

XmlBinding6 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.

XmlBinding7 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.

XmlBinding7.1 If a Product includes Technologies in addition to the Technology Under Test, then it must contain the full set of combined public and protected classes and interfaces. The API of the Product must contain the union of the included Technologies. No further modifications to the APIs of the included Technologies are allowed.

XmlBinding8 Except for tests specifically required by this TCK to be rebuilt (if any), the binary Conformance Tests supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

XmlBinding9 The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.

Rules for Products that include Support for Software Development

The following Rules, in addition to the Rules for XML Binding 3.0 Products, apply for each version of an operating system, software component, and hardware platform Documented as supporting the Product:

XmlBindingSD1 For each Product Configuration that causes the Schema Compiler to produce Schema Compiler Output, the Product must be able to satisfy all applicable compatibility requirements, including passing all Conformance Tests, except only as specifically exempted by these Rules.

XmlBindingSD2 For each Product Configuration that causes the Schema Compiler to produce Schema Compiler Output, in cases where such Schema Compiler Output contains only value classes, that Schema Compiler Output, when combined with the XML Binding Compatible Implementation Libraries, must execute properly when run on all Java SE Reference Runtime versions that correspond to versions of Java SE Documented as supporting the Product.

XmlBindingSD3 The Schema Compiler must not produce Schema Compiler Output from schema that do not conform to the W3C XML Schema recommendation [XSD Part 1][XSD Part 2].

XmlBindingSD3.1 The Schema Compiler in non-default Product Configurations may produce Schema Compiler Output when an XML Schema contains constructs for which a binding has not been defined by the Specification.

XmlBindingSD3.2 The Schema Compiler in non-default Product Configurations may produce Schema Compiler Output when a custom binding declaration is encountered.

XmlBindingSD4 The Schema Compiler Output of the Schema Compiler must be either in class format defined by the Java Virtual Machine (JVM) Specifications or in source file format defined by Java Language Specification (JLS).

XmlBindingSD5 For each Product Configuration that causes the Schema Generator to produce Schema Generator Output, the Product must be able to satisfy all applicable compatibility requirements, including passing all Conformance Tests, except only as specifically exempted by these Rules.

XmlBindingSD6 For each Product Configuration that causes the Schema Generator to produce Schema Generator Output, the Output of Schema Generator must fully meet W3C requirements for the XML schema language.

XmlBindingSD7 The Schema Generator must not produce Schema Generator Output when a program element violates mapping constraints defined by Specifications.

XmlBindingSD8 The Schema Generator may produce Schema Generator Output when a program element contains constructs for which a mapping has not been defined by the Specification.

XmlBindingSD9 A schema produced by the Schema Generator from source code containing a comment or a directive, or from a class file containing non-standard attributes, must be equivalent to a schema produced by the Schema Generator from the same source code with the comment or directive removed, or from the class file with the attributes removed. Two schemas are equivalent if and only if they validate the same set of XML documents (any XML document is either valid against both schemas or invalid against both schemas).

XmlBindingSD10 Except for tests specifically required by this TCK to be modified (if any) the source for Schema Compiler and Schema Generator Conformance Tests supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

2.3 Test Appeals Process

Jakarta has a well established process for managing challenges to its TCKs. Any implementor may submit a challenge to one or more tests in the XML Binding TCK as it relates to their implementation. Implementor means the entity as a whole in charge of producing the final certified release. **Challenges filed should represent the consensus of that entity.**

2.3.1 Valid Challenges

Any test case (e.g., test class, @Test method), test case configuration (e.g., deployment descriptor), test beans, annotations, and other resources considered part of the TCK may be challenged.

The following scenarios are considered in scope for test challenges:

- Claims that a test assertion conflicts with the specification.
- Claims that a test asserts requirements over and above that of the specification.
- Claims that an assertion of the specification is not sufficiently implementable.
- Claims that a test is not portable or depends on a particular implementation.

2.3.2 Invalid Challenges

The following scenarios are considered out of scope for test challenges and will be immediately closed if filed:

- Challenging an implementation's claim of passing a test. Certification is an honor system and these issues must be raised directly with the implementation.
- Challenging the usefulness of a specification requirement. The challenge process cannot be used to bypass the specification process and raise in question the need or relevance of a specification requirement.
- Claims the TCK is inadequate or missing assertions required by the specification. See the Improvement section, which is outside the scope of test challenges.
- Challenges that do not represent a consensus of the implementing community will be closed until such time that the community does agree or agreement cannot be made. The test challenge process is not the place for implementations to initiate their own internal discussions.
- Challenges to tests that are already excluded for any reason.
- Challenges that an excluded test should not have been excluded and should be re-added should be opened as a new enhancement request

Test challenges must be made in writing via the XML Binding specification project issue tracker as described in [Section 2.3.3, "TCK Test Appeals Steps."](#)

All tests found to be invalid will be placed on the Exclude List for that version of the XML Binding TCK.

2.3.3 TCK Test Appeals Steps

1. Challenges should be filed via the Jakarta XML Binding specification project's issue tracker using the label **challenge** and include the following information:
 - The relevant specification version and section number(s)
 - The coordinates of the challenged test(s)
 - The exact TCK and exclude list versions
 - The implementation being tested, including name and company
 - The full test name
 - A full description of why the test is invalid and what the correct behavior is believed to be
 - Any supporting material; debug logs, test output, test logs, run scripts, etc.

2. Specification project evaluates the challenge.

Challenges can be resolved by a specification project lead, or a project challenge triage team, after a consensus of the specification project committers is reached or attempts to gain consensus fails. Specification projects may exercise lazy consensus, voting or any practice that follows the

principles of Eclipse Foundation Development Process. The expected timeframe for a response is two weeks or less. If consensus cannot be reached by the specification project for a prolonged period of time, the default recommendation is to exclude the tests and address the dispute in a future revision of the specification.

3. Accepted Challenges.

A consensus that a test produces invalid results will result in the exclusion of that test from certification requirements, and an immediate update and release of an official distribution of the TCK including the new exclude list. The associated **challenge** issue must be closed with an **accepted** label to indicate it has been resolved.

4. Rejected Challenges and Remedy.

When a `challenge` issue is rejected, it must be closed with a label of **invalid** to indicate it has been rejected. There appeal process for challenges rejected on technical terms is outlined in Escalation Appeal. If, however, an implementer feels the TCK challenge process was not followed, an appeal issue should be filed with specification project's TCK issue tracker using the label **challenge-appeal**. A project lead should escalate the issue with the Jakarta EE Specification Committee via email (jakarta.ee-spec.committee@eclipse.org). The committee will evaluate the matter purely in terms of due process. If the appeal is accepted, the original TCK challenge issue will be reopened and a label of **appealed-challenge** added, along with a discussion of the appeal decision, and the **challenge-appeal** issue will be closed. If the appeal is rejected, the **challenge-appeal** issue should be closed with a label of **invalid**.

5. Escalation Appeal.

If there is a concern that a TCK process issue has not been resolved satisfactorily, the [Eclipse Development Process Grievance Handling](#) procedure should be followed to escalate the resolution. Note that this is not a mechanism to attempt to handle implementation specific issues.

2.4 Specifications for Jakarta XML Binding

The Jakarta XML Binding specification is available from the specification project web-site: <https://jakarta.ee/specifications/xml-binding/3.0/>.

2.5 Libraries for Jakarta XML Binding

The following is a list of the packages comprising the required class libraries for XML Binding 3.0:

- **jakarta.xml.bind**
- **jakarta.xml.bind.annotation**
- **jakarta.xml.bind.annotation.adapters**
- **jakarta.xml.bind.attachment**
- **jakarta.xml.bind.helpers**

-
- `jakarta.xml.bind.util`

For the latest list of packages, also see:

<https://jakarta.ee/specifications/xml-binding/3.0/>

3 Installation

This chapter explains how to install the Jakarta XML Binding TCK software.

After installing the software according to the instructions in this chapter, proceed to [Chapter 4, "Setup and Configuration,"](#) for instructions on configuring your test environment.

3.1 Obtaining a Compatible Implementation

Each compatible implementation (CI) will provide instructions for obtaining their implementation. Eclipse Implementation of JAXB is a compatible implementation which may be obtained from <https://github.com/eclipse-ee4j/jaxb-ri>

3.2 Installing the Software

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

- Java SE 8
- A CI for XML Binding 3.0, one example is Eclipse Implementation of JAXB
- XML Binding TCK version 3.0, which includes:
 - (no additional libraries)
- The XML Binding 3.0 Vendor Implementation (VI)

Follow these steps:

1. Install the Java SE 8 software, if it is not already installed.
Download and install the Java SE 8 software from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. Refer to the installation instructions that accompany the software for additional information.
2. Install the XML Binding TCK 3.0 software.
 1. Copy or download the XML Binding TCK software to your local system.
You can obtain the XML Binding TCK software from the Jakarta EE site <https://jakarta.ee/specifications/xml-binding/3.0/>.
 2. Use the `unzip` command to extract the bundle in the directory of your choice:
`unzip jakarta-xml-binding-tck-3.0.0.zip`
This creates the TCK directory. The TCK is the test suite home, `<TS_HOME>`.
3. The zip file above contains the XML Binding TCK installer jar file; use this installer to install the

XML Binding TCK.

1. Extract the XML Binding TCK 3.0 product bundle in the directory of your choice:

```
java -jar PATH-TO-XML-BINDING-TCK-JAR options [-o outputdir]
```

The following descriptions are provided for the installation command:

- **java**: Java SE 8 or later platform reference implementation.
 - **PATH-TO-XML-BINDING-TCK-JAR**: The path of the XML Binding TCK 3.0 JAR file to install.
 - **options**: Unless a user specifies options, the default behavior installs XML Binding TCK 3.0 in verbose mode (writes the file names it is installing to standard out) into the current directory. The installation utility accepts the following options:
 - **-q**: Installs XML Binding TCK 2.3 in quiet mode.
 - **-l**: Prints a list to standard output of the options that will be installed.
 - **-onError mode** : Sets the mode of sensitivity that the installer has to errors. The **-onError** : Instructs what to do when it encounters an error while installing XML Binding TCK 3.0. This option has three valid modes:
 - **ignore**: Instructs the installer to ignore any errors and continue installation.
 - **report**: Instructs the installer to continue with the installation but report any errors when the installation is completed.
 - **fail**: Instructs the installer to stop installation once an error is encountered.
 - **-usage**: Prints a usage message to standard out and exits (not listed as an option in the usage message).
 - **-help**: Prints a help message to standard out and exits (not listed as an option in the help message).
 - **[-o outputdir]**: Installs XML Binding TCK 3.0 into the directory specified by **outputdir**.
2. This action creates the **jaxb-tck23** directory. The topmost XML Binding TCK installation directory is referred to as the Test Suite Home, or **<TS_HOME>**.

4. Install a XML Binding 3.0 Compatible Implementation.

A Compatible Implementation is used to validate your initial configuration and setup of the XML Binding TCK 3.0 tests, which are explained further in [Chapter 4, "Setup and Configuration."](#)

The Compatible Implementations for XML Binding are listed on the Jakarta EE Specifications web site: <https://jakarta.ee/specifications/xml-binding/3.0/>.

5. Install the XML Binding VI to be tested.

Follow the installation instructions for the particular VI under test.

4 Setup and Configuration



The Jakarta EE Specification process provides for any number of compatible implementations. As additional implementations become available, refer to project or product documentation from those vendors for specific TCK setup and operational guidance.

This chapter describes how to set up the XML Binding 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 XML Binding TCK.

4.1 Configuring the XML Binding TCK

4.1.1 TCK Configuration Concepts

XML Binding 3.0 comprises of a schema compiler and a schema generator. Both are types of compilers; the former maps from a schema to Java, and the latter maps from Java to schema. For the purposes of brevity in these instructions, the term compiler refers to both the schema compiler and the schema generator.

GUI Versus Batch Mode

Once you have configured your XML Binding TCK environment using the JavaTest GUI, you can run the tests through the GUI or in batch mode using scripts. However, you cannot edit your XML Binding TCK configuration files (*.jti) outside the JavaTest GUI. Even though the *.jti configuration files are straight ASCII text, they have embedded checksum strings to ensure validity and consistency with the requirements of the XML Binding TCK suite.

Batch mode is particularly useful after you have worked out the basics of your test configurations through the JavaTest GUI. Batch mode provides a convenient way to automate your test runs, and also allows you to change work and report directories, which is often desirable when running tests repeatedly over progressive builds.

Using the XML Binding TCK Configuration Interfaces

You can configure the XML Binding TCK and the JavaTest harness from the JavaTest GUI, or by passing configuration parameters directly on the command line in a command shell.

XML Binding TCK Configuration With the JavaTest GUI

1. Start the JavaTest harness GUI. See [Section, "Starting the JavaTest Harness"](#).
2. Complete the JavaTest harness **Quick Start Wizard** or use the **Test Manager** menu to perform the following tasks:
 - Select **File** and then **Open Test Suite** to open a test suite.
 - Select **File** and then **New Work Dir** to create a new work directory, or **File** and then **Open Work Dir** to open an existing work directory.
 - Select **Configure** then **New Configuration** to create a new configuration, **Configure** then **Load Configuration** to load an existing configuration, or **Configure** and then **Change Values** to modify the current configuration for your test environment.
The JavaTest harness opens the configuration editor window. See the JavaTest Harness User's Guide: Graphical User Interface included with this release or the JavaTest harness online help for detailed instructions on using the configuration editor window.
 - Click **Start Running Tests** on the **Test Manager** tool bar , or select **Run Tests** and then **Start** from the **Test Manager** menu.
See the JavaTest Harness User's Guide: Graphical User Interface included with this release or the JavaTest harness online help for detailed instructions about running the tests.

TCK Configuration From the JavaTest Command Line

To run the tests non-interactively, for example as part of a product build process, use the JavaTest harness command-line interface. To run tests from the command line, you must use a configuration file. The JavaTest harness command-line interface provides a number of commands that you can use to specify individual configuration values used to run tests. Values that are set in the command line will override but not change the values in the configuration file. Permanent changes to the configuration are made by using the Configuration Editor to change the values in the configuration file.

See the JavaTest Harness User's Guide: Command Line Interface included with this release or the JavaTest harness online help for detailed instructions about using the command line to run tests and to modify the configuration.

Setting Configuration Values on the Command Line

If you intend to set one or more configuration values on the command line for a test run, then see the JavaTest Harness User's Guide: Command-Line Interface, or the JavaTest harness online help for a detailed description.

You must use one of the following sources to identify the configuration question-tag names used in the command line:

- **Question Log file:** ([questionLognnnn.html](#)) generated by the Configuration Editor in [workdirectory/jtData](#)
- **Config.html report:** generated from the JavaTest harness GUI

- **Question Tags:** displayed in the Configuration Editor

See the *JavaTest Harness User's Guide: Graphical User Interface* or the *JavaTest harness online help* for a detailed description of displaying the Question Tags in the Configuration Editor and generating reports containing the `config.html` file.

4.1.2 XML Binding-Specific Configuration Settings

A number of questions in the JavaTest harness configuration interview are generic and apply to most Java product TCK suites. This section describes configuration settings that are specific to the XML Binding TCK.

The XML Binding TCK 3.0 includes a special configuration interview used by the JavaTest harness to collect information from you about your test environment and to create the configuration used to run tests. The TCK configuration interview (TCK interview) consists of the following sections:

- **Test Environment:** the first section of the interview collects information about the test environment used to run the XML Binding TCK 3.0 (such as the type of environment and the type of agent used).
- **Schema Compiler Questions:** the second section of the interview collects information about how to compile the source schema on your platform.
 - XML Validation Tests: optional tests that must be run if schema validation is performed using a mechanism other than JAXP.
- **Schema Generator Questions:** the third section of the interview collects information about how to generate XML schema from Java sources on your platform.
- **How Tests are Run:** the last section of the interview collects information on how the tests are run (such as Keywords and the Exclude List).

The Configuration Editor Questions

The table describes the Configuration Editor questions, lists the question tags and text, and gives examples of relevant answers to correctly configure the JavaTest harness to run the test suite.



To view the question tags in the Configuration Editor, select **View** and then **Question Tag** in the editor menu.

Table 4-1 Configuration Editor Questions and Relevant Answers

Item Name	Question Tag and Text	Comments and Proposed Answers
Welcome!	Tag: <code>jck.prolog</code> Welcome to the Configuration Editor for XML Binding TCK testsuite!	None
Configuration Name	Tag: <code>jck.env.envName</code> Provide a short identifier that names the configuration you are creating.	This is a short string that identifies this particular configuration. Select a name that describes the configuration. Example: <code>jaxb_win32</code> The name must begin with a letter, followed by letters, digits, or an underscore, and must not contain a white space or punctuation characters.
Description	Tag: <code>jck.env.description</code> Provide a short description that identifies the configuration you are creating.	JavaTest uses this short description to provide more detail about the configuration, for example, in reports. This information might be useful to an auditor looking at the test run reports, and could include the version number of the product and the name of the tester. Example: <code>John Smith x86 Java Architecture for XML Binding 2.3</code>

Item Name	Question Tag and Text	Comments and Proposed Answers
Execution Mode	<p>Tag: <code>jck.env.testPlatform.multiJVM</code></p> <p>Specify whether you want to run the tests in SingleJVM or MultiJVM mode. Depending on the selected mode, you can configure to run the compiler either from the command line, or via the Java Compiler API, or via a custom class accessed by the core reflection API.</p>	<p>Answer MultiJVM Mode if the JavaTest harness can directly start your product, for example, from a command line in an MSDOS window or UNIX shell; this is the required mode for the XML Binding TCK.</p> <p>Typically, applications that do not support such actions run in point-and-click environments or in browsers. For example, the JavaTest harness cannot start a new instance of a Java virtual machine inside a browser.</p> <p>If you answer MultiJVM Mode , then you need to answer additional interview questions to specify whether you plan to run the tests locally. In addition, you need to specify the details about the Java Virtual Machine in which the tests will run (the questions in the Runtime Other JVM section of this interview) and the details about the Java launcher.</p>
Execute Locally	<p>Tag: <code>jck.env.testPlatform.local</code></p> <p>Will you run the TCK tests on the system that hosts the JavaTest harness?</p>	<p>If you plan to run the tests on the system on which you are running this interview, then the editor gathers configuration information directly from the system. For example, you will be able to browse the local file system to point to the required files.</p>
Agent Type	<p>Tag: <code>jck.env.jaxb.agent.agentType</code></p> <p>The JavaTest Agent can be run in two modes: active and passive. Which mode do you wish to use?</p>	<p>Select the agent type you wish to use. If you want to use the passive agent, then two additional questions will appear asking you to specify the passive agent host and default agent port.</p>
Passive Host	<p>Tag: <code>jck.env.jaxb.agent.agentPassiveHost</code></p> <p>What is the of the host on which you run the tests?</p>	<p>This is the name of the system on which the passive agent runs and executes tests. The name of the system must be accessible from the system on which JavaTest runs. Since the JavaTest agent is passive, you can only change the machine you specify here by running this interview again and changing the name of the passive host.</p>

Item Name	Question Tag and Text	Comments and Proposed Answers
Default Agent Port	<p>Tag: <code>jck.env.jaxb.agent.useAgentPortDefault</code></p> <p>Do you wish to use the default port for the JavaTest harness to contact the JavaTest Agent?</p>	<p>By default, the JavaTest harness communicates with a passive agent using port 1908. You must change only the default if you are running more than one passive agent on a system; this might be the case if you have set up batch runs.</p> <p>If you answer No, then you will need to specify the default port number in the next interview question.</p>
Passive Port	<p>Tag: <code>jck.env.jaxb.agent.agentPassivePort</code></p> <p>Which port does the JavaTest harness use to contact the JavaTest Agent?</p>	Specify the port you want to use to communicate with the JavaTest agent.
Test Platform File Separator	<p>Tag: <code>jck.env.jaxb.agent.fileSeparator</code></p> <p>Which file separator character is used on the test platform?</p>	<p>Specify the appropriate file separator for your operating system.</p> <p>For example, for Windows, use the back slash (\) character. For Solaris OS or Linux operation system, use the forward slash (/) character.</p>
Test Platform Path Separator	<p>Tag: <code>jck.env.jaxb.agent.pathSeparator</code></p> <p>Which path separator character is used on the test platform?.</p>	<p>Specify the appropriate path separator for your operating system.</p> <p>For example, on Windows systems, specify the ; character, on Unix systems specify the : character. Other platforms may differ.</p>
Agent Map File	<p>Tag: <code>jck.env.jaxb.agent.mapArgs</code></p>	<p>Because JavaTest and the JavaTest agent run on different systems, values such as path names and variable names may differ on each system. For example, if a file is accessed over a network, it might be mounted as H:\tests\lib on the JavaTest system and as F:\tests\lib on the system running the JavaTest agent. The map file is used to associate values used on the JavaTest agent with those used by JavaTest. The map file is specified on the JavaTest agent, but the use of the map must be enabled here. For more information about map files, refer to the JavaTest User's Guide or to the JavaTest online help.</p>

Item Name	Question Tag and Text	Comments and Proposed Answers
Java Launcher	<p>Tag: <code>jck.env.jaxb.testExecute.cmdAsFile</code></p> <p>Enter the full path name of the Java launcher that runs the tests through the Java Compiler API interface.</p>	<p>This interview question appears only if you answered Yes to the Test Platform MultiJVM question of this interview.</p> <p>This is the command used on the reference system to invoke the java command.</p> <p>NOTE: Use the java command in JDK (not JRE).</p> <p>* On Solaris or Linux platforms, type: <code>\$JAVA_HOME/bin/java</code></p> <p>For example: <code>/usr/java/bin/java</code></p> <p>* On Win32 systems, type: <code>%JAVA_HOME%\bin\java.exe</code></p> <p>For example: <code>"C:\Java\jdk-9\bin\java.exe"</code></p> <p>Ensure that you supply the full file name, including the file extension.</p>
Other Options	<p>Tag: <code>jck.env.jaxb.testExecute.otherOpts</code></p> <p>Specify any additional commandline options that must be set.</p>	<p>Specify any command-line options that JavaTest should set for your compiler when it runs the tests on your system. Specify the options using name=value pairs. For example you may want to specify: <code>-Xmx256m -Xms128m</code> to change the default maximum and initial Java heap size.</p> <p>NOTE: If you use Java Launcher from JCK 8 to run the tests, specify the <code>-Djava.endorsed.dirs=<JAXB_HOME>/lib</code> option.</p>

Item Name	Question Tag and Text	Comments and Proposed Answers
Other Environment Variables	<p>Tag: jck.env.jaxb.testExecute.otherEnvVars</p> <p>Except for CLASSPATH, if there are any other environment variables that must be set for the reference Java launcher, enter them here.</p>	<p>Specify any environment variables that the JavaTest harness should use in the environment created to run the tests on your system. Specify the variables using the name=value pairs.</p> <p>For example, on Solaris systems: special_path=/home/env/config</p> <p>NOTE: You must specify both variables: JAVA_HOME and JAXB_HOME. For example, on Solaris: JAVA_HOME=/jdk-8/solarissparcv9 JAXB_HOME=/jaxb-2.3 on Windows: JAVA_HOME=d:\jdk-8\windows-i586 JAXB_HOME=d:\jaxb-2.3</p> <p>Where the path to the directory of J2SE on Solaris should be specified in JAVA_HOME, JAXB_HOME points to the directory under which the bin/, docs/, examples/, and lib/ directories reside.</p> <p>The JavaTest cannot inherit environment variables from your operating system environment. If you find that your tests run correctly outside of the JavaTest harness, but do not run correctly using the JavaTest harness, then the reason might be that a value set in your operating system environment is not available to the test. If you check your operating system environment and determine that this is the case, then set the missing values here.</p>

Item Name	Question Tag and Text	Comments and Proposed Answers
Need Extra XML Binding Classes	<p>Tag: <code>jck.env.jaxb.classes.needJaxbClasses</code></p> <p>Do you need to specify extra XML Binding classes for the Java launcher to execute tests?</p>	<p>The XML Binding classes must be available (that is, can be found and loaded) during test execution. This means that XML Binding JAR archives or directories which contain the XML Binding classes must be in the test executing Java class path.</p> <p>Answer No if the XML Binding JAR archives are in the Java default class path, for example, all jar files are stored in the <code>jdk_install_dir/jre/lib/ext</code> directory.</p> <p>Answer Yes if some directories or archives are located not in the Java default class path.</p> <p>If you answer Yes, then you will be asked to specify necessary class paths or XML Binding archives.</p>
Schema Compiler Questions	<p>Tag: <code>jck.env.jaxb.xsd_compiler.intro</code></p> <p>The following questions collect information about how to compile the source schema on your platform.</p>	None.

Item Name	Question Tag and Text	Comments and Proposed Answers
Schema Compiler Operation Mode	<p>Tag: <code>jck.env.jaxb.xsd_compiler.defaultOperationMode</code></p> <p>Would you like to run your schema compiler in default operation mode?</p>	<p>A XML Binding compatible implementation must support a default operating mode in which all the required XML Schema to Java bindings that are described in the XML Binding specification must be implemented. Errors must be reported when alternative or extension non-required features are encountered.</p> <p>A XML Binding compatible implementation may support non-default operating modes. These modes must support all the XML Schema to Java bindings that are described in XML Binding specification, but may also generate alternative or extension bindings for XML Schema constructs, which are not required by this specification.</p> <p>If you select Yes, then the schema compiler is supposed to run in default operation mode and the tests for non-required features must be passed (as negative tests with expected error messages).</p> <p>If you select No, then the schema compiler is supposed to run in non-default operation mode and the tests for non-required features will not be run. For more information on default operation mode requirements, see the Compatibility section of the XML Binding specification.</p>
Optional XML Validation Tests	<p>Tag: <code>jck.env.jaxb.xsd_compiler.skipValidationOptional</code></p> <p>Is Schema Validation implemented via mechanism different from JAXP?</p>	<p>XML validation can be implemented from JAXP or from some other mechanism. If the validation is implemented from JAXP, then validation checking tests can be skipped by deselecting them in the test interview. If you use some other tools for schema validation, then XML validation tests are mandatory.</p>

Item Name	Question Tag and Text	Comments and Proposed Answers
Schema Compiler Run Command	<p>Tag: jck.env.jaxb.xsd_compiler.testCompile.xjcCmd</p> <p>The remaining questions ask which tests you want to run and how you want them to be run.</p>	<p>This is the command used to invoke your schema compiler. For example, for the Eclipse XML Binding implementation on Solaris we provide a script <code>TCKDIR/solaris/bin/ xjc.sh</code>, which compiles schemas into class files. The command is as follows: <code>/bin/ksh solaris/bin/xjc.sh</code> For the Eclipse XML Binding implementation on Windows, we provide a script <code>TCKDIR\win32\bin\xjc.bat</code>, which compiles schemas into class files. The command is as follows: <code>win32\bin\xjc.bat</code> The xjc script supports the following mandatory command-line options: <code>xjc -p <pkg> -d <dir> <schema files></code> where:</p> <ul style="list-style-type: none"> • <code>-p <pkg></code> specifies the target package • <code>-d <dir></code> specifies the directory to store generated files.

Item Name	Question Tag and Text	Comments and Proposed Answers
Schema Compiler Class	<p>Tag: jaxb.xsd_compiler.run.compilerWrapperClass</p> <p>You must provide a custom class to invoke your compiler. This class should implement the <code>com.sun.jaxb_tck.lib.SchemaCompilerTool</code> interface.</p>	<p>When you run schema compilation through the JavaTest agent, your schema compiler will be executed in the agent's JVM. Your implementation must provide a custom class to invoke your schema compilation tool. This class should implement the <code>SchemaCompilerTool</code> interface: <code>package com.sun.jaxb_tck.lib; public interface SchemaCompilerTool { /** * @param xsdFiles - array of strings containing schema files * @param packageName - the target package * @param outDir - output directory where java file(s) will be generated * @param out - output stream for logging * @param err - error stream for logging * @return 0 if java file(s) generated successfully */ int compile(String[] xsdFiles, String packageName, File outDir, PrintStream out, PrintStream err); }</code> Schema compilation should be accomplished during invocation of <code>compile</code> method. The XML Binding TCK includes the following class that is fully compatible with Eclipse's compatible implementation and that can be used for schema compilation: <code>com.sun.jaxb_tck.lib.SchemaCompiler</code>. The realization of the method <code>compile(String[] xsdFiles, String packageName, File outDir, PrintStream out, PrintStream err)</code> compiles a schema into <code>java</code> sources. If your implementation doesn't provide such a class, then create it and specify it here.</p>
Schema Generator Questions	<p>Tag: jck.env.jaxb.schemagen.intro</p> <p>The following questions collect information about how to generate xml schema from java sources on your platform.</p>	None.
Schema Generator Presence	<p>Tag: jck.env.jaxb.schemagen.skipJ2XOptional</p> <p>Does your implementation provide schema generator which is able to generate schemas from Java sources?</p>	Java Architecture for XML Binding 2.3 specification does not require any implementation, which provides a possibility to generate the schemas from the java sources. If they are provided, then all specification requirements should be met.

Item Name	Question Tag and Text	Comments and Proposed Answers
Schema Generator Run Command	<p>Tag: jck.env.jaxb.schemagen.run.jxcCmd</p> <p>Enter the command to run the schema generator.</p>	<p>This is the command used to invoke your schema generator. For example, for the Eclipse XML Binding implementation on Solaris we provide a script TCKDIR/solaris/bin/schemagen.sh, which generates the schemas from the java files. The command is: /bin/ksh solaris/bin/schemagen.sh For the Eclipse XML Binding implementation on Windows, we provide a script TCKDIR\win32\bin\schemagen.bat, which compiles the schemas into class files. The command is: win32\bin\schemagen.bat The schemagen script supports the following mandatory command-line options: schemagen -d <dir> <java files>, where, -d <dir> specifies the directory to store the generated files.</p>

Item Name	Question Tag and Text	Comments and Proposed Answers
Schema Generator Class	<p>Tag: <code>jck.env.jaxb.schemagen.run.schemagenWrapperClass</code></p> <p>You must provide a custom class to invoke your schema generator. This class should implement the <code>com.sun.jaxb_tck.lib.SchemaGenTool</code> interface.</p>	<p>When you run schema generation through JavaTest agent, your schema generator will be executed in the agent's JVM. Your implementation must provide a custom class to invoke your schema generation tool. This class should implement the <code>SchemaGenTool</code> interface:</p> <pre>package com.sun.jaxb_tck.lib; public interface SchemaGenTool { /** * @param javaFiles - array of strings containing java files * @param outDir - output directory where schema file(s) will be generated * @param out - output stream for logging * @param err - error stream for logging * @return 0 if schema file(s) generated successfully */ int generate(String[] javaFiles, File outDir, PrintStream out, PrintStream err); }</pre> <p>Schema generations should be complete during the invocation of the <code>generate</code> method. The XML Binding TCK includes the following class that is fully compatible with Eclipse's compatible implementation and that can be used for schema generation: <code>com.sun.jaxb_tck.lib.SchemaGen</code>. The realization of the method <code>generate(String[] javaFiles, File outDir, PrintStream out, PrintStream err)</code> generates the XML schemas from the <code>java</code> files. If your implementation doesn't provide such a class, then create it and specify it here.</p>
Schema Compiler Questions	<p>Tag: <code>jck.env.jaxb.xsd_compiler.intro</code></p> <p>The following questions collect information about how to compile the source schema on your platform.</p>	None.
Standard Values	<p>Tag: <code>jck.parameters</code></p> <p>The remaining questions ask which tests you want to run and how you want them to be run.</p>	None.

Item Name	Question Tag and Text	Comments and Proposed Answers
Specify an Exclude List?	Tag: <code>jck.excludeList.needExcludeList</code> Do you wish to specify an exclude list?	Answer Yes if you want to specify an exclude list for the test run. If you select Yes, then select the location of the exclude list to use. Answer No if you want to run all tests that are not filtered out by other run filters.
Which Exclude List?	Tag: <code>jck.excludeList.excludeListType</code> Which exclude list do you wish to use?	This question appears only if you answered Yes to the previous configuration interview question. Select either the Initial option to use the exclude list provided with the test suite or the Specify option to specify the exclude list you wish to use.
Specify Exclude List Files	Tag: <code>jck.excludeList.customFiles</code> Specify the files that make up the exclude list you wish to use.	Click Add to activate a file chooser with which you select an exclude list file. Exclude list files conventionally use the extension <code>`.jtx`</code> .
Specify Keywords?	Tag: <code>jck.keywords.needKeywords</code> Do you wish to specify a keyword expression to select/reject tests based on keywords contained in each test description?	Keywords are tokens associated with specific tests. Answer Yes to run tests based on the keyword and the other run filter settings. If you select Yes, then you are asked to specify the keyword to use. Answer No to run all tests that are not filtered out by other run filters.
Specify Status?	Tag: <code>jck.priorStatus.needStatus</code> Do you wish to select tests to run based on their result in a previous run?	Answer Yes to run tests based on the status from the previous test run and the other run filter settings. If you select Yes, then you are asked for the status or statuses. Answer No to run all tests that are not filtered out by other run filters.
Concurrency	Tag: <code>jck.concurrency.concurrency</code> The JavaTest harness can run tests concurrently. Specify the maximum number of tests that should be run concurrently:	The JavaTest harness can run tests concurrently. If you are running the tests on a multiprocessor computer, then the concurrency can accelerate your test runs. The default range of values that the JavaTest harness uses is from 1 to 50. For the first test run, keep this value set to 1.

Item Name	Question Tag and Text	Comments and Proposed Answers
Time Factor	Tag: <code>jck.timeout.timeout</code> Specify a time factor that is applied to each test's default timeout. For example, specifying 2 doubles the time for each test (the default is 1):	The default 1.0 corresponds to 10 minutes. Use the floating point format specific to your locale. NOTE: Setting a proper time factor value requires consideration of such technical factors as the network connector speed.
Completed!	Tag: <code>jck.end</code> Congratulations! This completes the configuration. Click Done to save your answers and close the editor. To run the test suite, choose Run Tests → Start.	None.

4.2 Unused Section

4.3 Unused Section

4.4 Custom Configuration Handlers

Configuration handlers are used to configure and unconfigure a XML Binding 3.0 implementation during the certification process. These are similar to deployment handlers but used for configuration. A configuration handler is an Ant build file that contains at least the required targets listed below:

- `config.vi` - to configure the vendor implementation
- `clean.vi` - to unconfigure the vendor implementation

These targets are called from the `<TS_HOME>/bin/build.xml` file and call down into the implementation-specific configuration handlers.

To provide your own configuration handler, create a `config.vi.xml` file with the necessary configuration steps for your implementation and place the file under the `<TS_HOME>/bin/xml/impl/<your_impl>` directory.

For more information, you may wish to view `<TS_HOME>/bin/xml/impl/glassfish/config.vi.xml`, the configuration file for Eclipse EE4J Jakarta EE 8 Compatible Implementation, Eclipse GlassFish.

4.5 Custom Deployment Handlers

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

The XML Binding TCK provides these deployment handlers:

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

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

4.5.1 To Create a Custom Deployment Handler

To deploy tests to another XML Binding implementation, you must create a custom handler.

1. Create a new directory in the `<TS_HOME>/bin/xml/impl` directory tree. For example, create the `<TS_HOME>/bin/xml/impl/my_deployment_handler` directory. Replace `my_deployment_handler` with the value of the `impl.vi` property that you set in Step 5 of the configuration procedure described in Section 4.2, "Configuring Your Environment to Repackage and Run the TCK Against the Vendor Implementation".
2. Copy the `deploy.xml` file from the `<TS_HOME>/bin/xml/impl/none` directory to the directory that you created.
3. Modify the required targets in the `deploy.xml` file. This is what the `deploy.xml` file for the "none" deployment handler looks like.

```

<project name="No-op Deployment" default="deploy">
  <!-- No-op deployment target -->
  <target name="-deploy">
    <echo message="No deploy target implemented for this deliverable"/>
  </target>
  <target name="-undeploy">
    <echo message="No undeploy target implemented for this deliverable"/>
  </target>
  <target name="-deploy.all">
    <echo message="No deploy target implemented for this deliverable"/>
  </target>
  <target name="-undeploy.all">
    <echo message="No undeploy target implemented for this deliverable"/>
  </target>
</project>

```

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

The following Ant targets can be called from anywhere under the `<TS_HOME>/src` directory:

- `deploy`
- `undeploy`
- `deploy.all`
- `undeploy.all`

The `deploy.all` and `undeploy.all` targets can also be called from the `<TS_HOME>/bin` directory.



The targets in the `deploy.xml` file are never called directly. They are called indirectly by the targets listed above.

4.6 Using the JavaTest Harness Software

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

- Through the JavaTest GUI; if using this method, please continue on to [Section 4.7, "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."](#)

4.7 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 XML Binding TCK test environment. This section covers the following topics:

- [Configuration GUI Overview](#)
- [Starting the Configuration GUI](#)
- [To Configure the JavaTest Harness to Run the XML Binding TCK Tests](#)
- [Modifying the Default Test Configuration](#)



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."](#)

4.7.1 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 Start, then Run Tests 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.

4.7.2 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 XML Binding TCK includes an Ant script that is used to execute the JavaTest harness from the `<TS_HOME>` directory. Using this Ant script to start the JavaTest harness is part of the procedure described in [Section 4.7.3, "To Configure the JavaTest Harness to Run the TCK Tests."](#)

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

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

After you specify a work directory, you can use the Test Manager to configure and run tests as described in [Section 4.7.3, "To Configure the JavaTest Harness to Run the TCK Tests."](#)

4.7.3 To Configure the JavaTest Harness to Run the TCK 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.

Note that you only need to complete all these steps the first time you start the JavaTest test harness. After you complete these steps, you can either run all of the tests by completing the steps in [Section 5.1, "Starting JavaTest,"](#) or run a subset of the tests by completing the steps in [Section 5.2, "Running a Subset of the Tests."](#)

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

```
cd <TS_HOME>/bin
ant gui
```
2. From the File menu, click **Open Quick Start Wizard**.

The Welcome screen displays.

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

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

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

You are prompted to select a test suite.

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

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

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

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

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

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

The JavaTest harness starts running the tests.

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

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

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

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

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

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

4.7.4 Modifying the Default Test Configuration

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

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

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

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

5 Executing Tests

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

This chapter includes the following topics:

- [Starting JavaTest](#)
- [Running a Subset of the Tests](#)
- [Running the TCK Against your selected CI](#)
- [Running the TCK Against a Vendor's Implementation](#)
- [Test Reports](#)



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 XML Binding TCK using the JavaTest harness software:

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



The `ant` command referenced in the following two procedures and elsewhere in this guide is the Apache Ant build tool, which will need to be downloaded separately. The `build.xml` file in `<TS_HOME>/bin` contains the various Ant targets for the XML Binding TCK test suite.

5.1.1 To Start JavaTest in GUI Mode

Execute the following commands:

```
cd <TS_HOME>/bin
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. Start JavaTest using the following command:

```
ant runclient
```

Example 5-1 XML Binding TCK Signature Tests

To run the XML Binding TCK signature tests, enter the following commands:

```
cd <TS_HOME>/tests/api/signaturetest/jaxb
ant runclient
```

Example 5-2 Single Test Directory

To run a single test directory, enter the following commands:

```
cd <TS_HOME>/tests/api/javax_xml/bind/JAXBContext
ant runclient
```

Example 5-3 Subset of Test Directories

To run a subset of test directories, enter the following commands:

```
cd <TS_HOME>/tests/api/javax_xml/bind
ant runclient
```

5.2 Running a Subset of the Tests

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

- [Section 5.2.1, "To Run a Subset of Tests in GUI Mode"](#)
- [Section 5.2.2, "To Run a Subset of Tests in Command-Line Mode"](#)
- [Section 5.2.3, "To Run a Subset of Tests in Batch Mode Based on Prior Result Status"](#)

5.2.1 To Run a Subset of Tests in GUI Mode

1. From the JavaTest main menu, click **Configure**, then click **Change Configuration**, and then click **Tests to Run**.
The tabbed Configuration Editor dialog box is displayed.
2. Click **Specify** from the option list on the left.
3. Select the tests you want to run from the displayed test tree, and then click **Done**.
You can select entire branches of the test tree, or use **Ctrl+Click** or **Shift+Click** to select multiple tests or ranges of tests, respectively, or select just a single test.
4. Click **Save File**.
5. Click **Run Tests**, and then click **Start** to run the tests you selected.
Alternatively, you can **right-click** the test you want from the test tree in the left section of the JavaTest main window, and choose **Execute These Tests** from the menu.
6. Click **Report**, and then click **Create Report**.
7. Specify the directory in which the JavaTest test harness will write the report, and then click **OK**.
A report is created, and you are asked whether you want to view it.
8. Click **Yes** to view the report.

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

1. Change to the directory containing the tests you want to run.
2. Start the test run by executing the following command:

```
ant runclient
```

The tests in the directory and its subdirectories are run.

5.2.3 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 XML Binding tests with a status of failed and error by invoking the following commands:

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

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

5.3 Running the TCK Against another CI

Some test scenarios are designed to ensure that the configuration and deployment of all the prebuilt XML Binding TCK tests against one Compatible Implementation are successful operating with other compatible implementations, and that the TCK is ready for compatibility testing against the Vendor and Compatible Implementations.

1. Verify that you have followed the configuration instructions in [Section 4.1, "Configuring Your Environment to Run the TCK Against the Compatible Implementation."](#)
2. If required, verify that you have completed the steps in [Section 4.3.2, "Deploying the Prebuilt Archives."](#)
3. Run the tests, as described in [Section 5.1, "Starting JavaTest,"](#) and, if desired, [Section 5.2, "Running a Subset of the Tests."](#)

5.4 Running the TCK Against a Vendor's Implementation

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

1. Verify that you have followed the configuration instructions in [Section 4.2, "Configuring Your Environment to Repackage and Run the TCK Against the Vendor Implementation."](#)
2. If required, verify that you have completed the steps in [Section 4.3.3, "Deploying the Test Applications Against the Vendor Implementation."](#)
3. Run the tests, as described in [Section 5.1, "Starting JavaTest,"](#) and, if desired, [Section 5.2, "Running a Subset of the Tests."](#)

5.5 Test Reports

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

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

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

5.5.1 Creating Test Reports

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, click **Report**, then click **Create Report**.
You are prompted to specify a directory to use for your test reports.
2. Specify the directory you want to use for your reports, and then click **OK**.
Use the Filter list to specify whether you want to generate reports for the current configuration, all tests, or a custom set of tests.
You are asked whether you want to view report now.
3. Click **Yes** to display the new report in the JavaTest ReportBrowser.

5.5.1.2 To Create a Test Report in Command-Line Mode

1. 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.

2. To specify the default report directory, set the `report.dir` property in `<TS_HOME>/lib/jaxb_tck23.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_HOME>/lib/jaxb_tck23.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, click **Report**, then click **Open Report**.
You are prompted to specify the directory containing the report you want to open.
2. Select the report directory you want to open, and then click **Open**.
The selected report set is opened in the JavaTest ReportBrowser.

5.5.2.2 To View an Existing Report in Command-Line Mode

Use the Web browser of your choice to view the `report.html` file in the report directory you specified from the command line or in `<TS_HOME>/lib/jaxb_tck23.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 includes the following topics:

- [Overview](#)
- [Test Tree](#)
- [Folder Information](#)
- [Test Information](#)
- [Report Files](#)
- [Configuration Failures](#)

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. Ant test execution tasks provide command-line users with immediate test execution feedback to the display. Available JTR report files and log files can also help command-line users troubleshoot test run problems.

For every test run, the JavaTest harness creates a set of report files in the reports directory, which you specified by setting the `report.dir` property in the `<TS_HOME>/lib/jaxb_tck23.jte` file. The report files contain information about the test description, environment, messages, properties used by the test, status of the test, and test result. After a test run is completed, the JavaTest harness writes HTML reports for the test run. You can view these files in the JavaTest ReportBrowser when running in GUI mode, or in the Web browser of your choice outside the JavaTest interface. To see all of the HTML report files, enter the URL of the `report.html` file. This file is the root file that links to all of the other HTML reports.

The JavaTest harness also creates a `summary.txt` file in the report directory that you can open in any

text editor. The `summary.txt` file contains a list of all tests that were run, their test results, and their status messages.

The work directory, which you specified by setting the `work.dir` property in the `<TS_HOME>/lib/jaxb_tck23.jte` file, contains several files that were deposited there during test execution: `harness.trace`, `log.txt`, `lastRun.txt`, and `testsuite`. Most of these files provide information about the harness and environment in which the tests were executed.



You can set `harness.log.traceflag=true` in `<TS_HOME>/lib/jaxb_tck23.jte` to get more debugging information.

If a large number of tests failed, you should read [Configuration Failures](#) to see if a configuration issue is the cause of the failures.

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.



You can set `harness.log.traceflag=true` in `<TS_HOME>/lib/jaxb_tck23.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. See [Section 5.5, "Test Reports,"](#) for more information.

6.6 Configuration Failures

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

Here are few tips to verify the errors:

1. 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`.
2. Check if the `env.html` report file is generated, else, the JavaTest GUI provides the ability to view evaluated variables used by the tests. Select **Configure** and then **Show Test Environment** from the Test manager menu to view the contents of the test environment.
3. Verify if the Configuration Editor has generated a configuration question log when you complete a configuration interview. You can use the configuration question log to review your answers to all the questions in the current configuration. Select **Configure** and **Show Question Log** from the **Test**

Manager menu to view the current configuration interview.

6.7 Test Manager Properties

You can view the properties of a test manager by selecting **View** and then **Properties** from the **Test Manager** menu. The Test Manager Properties dialog box contains Test Suite, Work Directory, Configuration, and Plugin information.

See JavaTest User's Guide , Graphical User Interface, or JavaTest online help for a detailed description of the Test Manager Properties dialog box.

6.8 Test Suite Errors

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

See JavaTest User's Guide , Graphical User Interface, or JavaTest online help for a detailed description of the **Test Manager: Test Suite Errors** dialog box.

6.9 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 Results

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 cause error when something (usually a configuration problem) keeps the test from being executed as expected. Errors often indicate a systemic problem - a single configuration problem can cause many tests to fail. For example, if the path to the Java runtime is configured incorrectly, then no tests can run and will result in an error.

How Tests Are Selected For a Test Run

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 the sub-branches of the tree as a way of limiting the tests that are executed during a test run. The JavaTest harness browses through the tree starting with its 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 Procedure for Certification.
- **Keywords:** A test can be selected based on keywords specified in the Keywords field in the test description. The possible keywords are the following: schema, document, runtime, positive, negative, bindinfo, jaxb_not_required, java_to_schema.
- **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-bytest basis using information stored in result files (.jtr) written in the work directory.

Test Execution

Based on the test keywords and the `executeClass` parameter of the test description, all the tests are divided in four execution models:

- **Schema Tests:** The keyword schema indicates this test execution model. The schema tests are the Schema Compiler Tests. These tests have a schema file (*.xsd) in the test description parameter source. The schema is compiled to the work directory, and the output directory option of the XML Binding schema compiler is used.
The package specified in the test description parameter package is used to set the output package parameter to the schema compiler. If the test has the keyword negative, then the schema compiler must report an error compiling the test schema. Otherwise, the schema is positive and may have XML Binding custom binding information if the keyword bindinfo is present. + The positive schema is expected to compile successfully. After the compilation passes, the signature test (the class specified in the test description parameter executeClass) runs with arguments specified in the test description parameter executeArgs.
- **Document Tests:** The keyword document along with the class JAXBTest specified in the test description parameter executeClass indicate this test execution model. The document tests are the subset of the Content Tree Tests, with the exception of the content tree modification stage.
The last two stages are applicable only to valid XML documents. The first stage (schema compilation) is performed as for the schema tests. All stages except the first one are performed by executing appropriate test cases (unmarshal, compareContent, and marshal respectively) of the class specified in test description parameter executeClass, with arguments specified in the test description parameter executeArgs.
All parameter entries starting with \$ are resolved in the environment, but no errors occur if any of

these values are not defined. If the exclude list identifies test cases to be excluded, then these cases are added to the test arguments using the `-exclude` option. Note that all schemas used in the document tests are valid and the tests may not have the keyword `negative`. Invalid documents are indicated by option `-invalid` in the parameter `executeClass`.

- XML Binding Framework API Tests:** The XML Binding framework API tests have their own classes (other than the signature test or `JAXBTest`) specified in the test description parameter `executeClass`. The tests may or may not have schema file specified in parameter `schema`.

First, if the schema is specified, it is compiled as described for the schema tests. Then the test executes the class specified in test description parameter `executeClass` with arguments specified in the test description parameter `executeArgs`.

All parameter entries starting with `$` are resolved in the environment, but no errors occur if any of these values are not defined. If the exclude list identifies test cases to be excluded, then these cases are added to the test arguments using the `-exclude` option.
- Java -to-Schema Tests:** The keyword `/java_to_schema/` indicates this test execution model. The Java-to-Schema Tests are the same as described in Schema Compiler Tests. These tests have a Java files (*.java) in the test description parameter `source`. The generated schema is placed in the work directory, and the output directory option of the XML Binding schema generator is used. If the test has the attribute `negative`, then the schema generator must report an error processing the test Java files. Otherwise, the Java code is correctly annotated and should be processed successfully. After the generation passes, the validation test (the class specified in the test description parameter `/executeClass/`) runs with arguments specified in the test description parameter `/executeArgs/`. The validation test checks that a valid XML document, taken from the test description parameter `/executeArgs/` follows the just-generated schema, and fails to check invalid XML documents.
- XML Validation Tests:** XML Validation can be implemented through JAXP or some other mechanism. If the validation is implemented through JAXP, then validation checking tests can be skipped by deselecting them in the test interview. If you use some other tools for schema validation, then XML validation tests are mandatory.

A Frequently Asked Questions

This appendix contains the following questions.

- [Where do I start to debug a test failure?](#)
- [How do I restart a crashed test run?](#)
- [What would cause tests be added to the exclude list?](#)

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

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

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

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

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

The JavaTest exclude file (`<TS_HOME>/lib/jaxb_tck23.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 compatible 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.

Appendix B is not used for the XML Binding TCK.