TCK User’s Guide for Technology Implementors

Table of Contents

Eclipse Foundation

Technology Compatibility Kit User's Guide for Jakarta Debugging Support for Other Languages

Release 2.0 for Jakarta EE

September 2020

Technology Compatibility Kit User's Guide for Jakarta Debugging Support for Other Languages, Release 2.0 for Jakarta EE

References in this document to DSOL refer to the Jakarta Debugging Support for Other Languages 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 Debugging Support for Other Languages (Debugging Support for Other Languages 2.0) technology.

The Debugging Support for Other Languages TCK is a portable, configurable automated test suite for verifying the compatibility of a vendor's implementation of the Debugging Support for Other Languages 2.0 Specification (hereafter referred to as the vendor implementation or VI).

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 jsp-dev@eclipse.org. If you have questions about this TCK, you may send inquiries to jakartaee-tck-dev@eclipse.org.

Who Should Use This Book

This guide is for vendors that implement the Debugging Support for Other Languages 2.0 technology to assist them in running the test suite that verifies compatibility of their implementation of the Debugging Support for Other Languages 2.0 Specification.

Before You Read This Book

You should be familiar with the Debugging Support for Other Languages 2.0, version 2.0 Specification, which can be found at https://jakarta.ee/specifications/debugging/2.0/.

Typographic Conventions

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

Convention	Meaning	Example

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

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

Shell Prompts in Command Examples

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

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

1 Introduction

This chapter provides an overview of the principles that apply generally to all Technology Compatibility Kits (TCKs) and describes the Jakarta Debugging Support for Other Languages TCK (Debugging Support for Other Languages 2.0 TCK). It also includes a high level listing of what is needed to get up and running with the Debugging Support for Other Languages TCK.

This chapter includes the following topics:

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 Debugging Support for Other Languages TCK 2.0 is designed as a portable, configurable, automated test suite for verifying the compatibility of a vendor's implementation of the Debugging Support for Other Languages 2.0 Specification.

The Debugging Support for Other Languages does not define APIs, but instead defines a data format and process. As a result, the TCK is different than most, it verifies the data format, and thus indirectly the process. The input to the process is source code in an arbitrary language, and thus the process cannot be directly tested by the TCK.

1.2.1 TCK Specifications and Requirements

This section lists the applicable requirements and specifications.

- Specification Requirements: Software requirements for a Debugging Support for Other Languages implementation are described in detail in the Debugging Support for Other Languages 2.0 Specification. Links to the Debugging Support for Other Languages specification and other product information can be found at https://jakarta.ee/specifications/debugging/2.0/.
- Debugging Support for Other Languages Version: The Debugging Support for Other Languages 2.0 TCK is based on the Debugging Support for Other Languages Specification, Version 2.0.
- Compatible Implementation: One Debugging Support for Other Languages 2.0 Compatible Implementation, Eclipse Glassfish 6.0 is available from the Eclipse EE4J project (https://projects.eclipse.org/projects/ee4j). See the CI documentation page at http://javaee.github.io/glassfish for more information.

See the Debugging Support for Other Languages TCK Release Notes for more specific information about Java SE version requirements, supported platforms, restrictions, and so on.

2 Procedure for Certification

This chapter describes the compatibility testing procedure and compatibility requirements for Jakarta Debugging Support for Other Languages. This chapter contains the following sections:

2.1 Certification Overview

The certification process for Debugging Support for Other Languages 2.0 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 Debugging Support for Other Languages 2.0 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	A piece of hardware or software that may vary in quantity, existence, or version, which may be required to exist in a minimum quantity and/or at a specific or minimum revision level so as to satisfy the requirements of the Test Suite. Examples of computational resources that may vary in quantity are RAM and file descriptors. Examples of computational resources that may vary in existence (that is, may or may not exist) are graphics cards and device drivers. Examples of computational resources that may vary in version are operating systems and device drivers.
Configuration Descriptor	Any file whose format is well defined by a specification and which contains configuration information for a set of Java classes, archive, or other feature defined in the specification.
Conformance Tests	All tests in the Test Suite for an indicated Technology Under Test, as 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	The class libraries, as specified through the Jakarta EE Specification Process (JESP), for the Technology Under Test. The Libraries for Jakarta Debugging Support for Other Languages are listed at the end of this chapter.

Term	Definition
Location Resource	A location of classes or native libraries that are components of the test tools or tests, such that these classes or libraries may be required to exist in a certain location in order to satisfy the requirements of the test suite. For example, classes may be required to exist in directories named in a CLASSPATH variable, or native libraries may be required to exist in directories named in a PATH variable.
Maintenance Lead	The 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.
Operating Mode	Any Documented option of a Product that can be changed by a user in order to modify the behavior of the Product. For example, an Operating Mode can be binary (enable/disable optimization), an enumeration (select from a list of protocols), or a range (set the maximum number of active threads). Note that an Operating Mode may be selected by a command line switch, an environment variable, a GUI user interface element, a configuration or control file, etc.
Product	A vendor's product in which the Technology Under Test is implemented or incorporated, and that is subject to compatibility testing.
Product Configuration	A specific setting or instantiation of an Operating Mode. For example, a Product supporting an Operating Mode that permits user selection of an external encryption package may have a Product Configuration that links the Product to that encryption package.
Rebuildable Tests	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.
Resource	A Computational Resource, a Location Resource, or a Security Resource.
Rules	These definitions and rules in this Compatibility Requirements section of this User's Guide.
Runtime	The Containers specified in the Specifications.

Term	Definition
Security Resource	A security privilege or policy necessary for the proper execution of the Test Suite. For example, the user executing the Test Suite will need the privilege to access the files and network resources necessary for use of the Product.
Specifications	The documents produced through the Jakarta EE Specification Process (JESP) that define a particular Version of a Technology. The Specifications for the Technology Under Test are referenced later in this chapter.
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 Debugging Support for Other Languages Version 2.0.
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). Unresolved directive in rules.adoc - include::defns.inc[]

2.2.2 Rules for Jakarta Debugging Support for Other Languages Products

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

DSOL1 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.

DSOL1.1 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.

DSOL1.2 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.

DSOL1.3 An API Definition Product is exempt from all functional testing requirements defined here, except the signature tests.

DSOL2 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.

DSOL3 The testing tools supplied as part of the Test Suite or as updated by the Maintenance Lead must be used to certify compliance.

DSOL4 The Exclude List associated with the Test Suite cannot be modified.

DSOL5 The Maintenance Lead can define exceptions to these Rules. Such exceptions would be made available as above, and will apply to all vendor implementations.

DSOL6 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.

DSOL7 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.

DSOL7.1 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.

DSOL8 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.

DSOL9 The functional programmatic behavior of any binary class or interface must be that defined by the Specifications.

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 Debugging Support for Other Languages 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 Debugging Support for Other Languages 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 Debugging Support for Other Languages TCK.

2.3.3 TCK Test Appeals Steps

1. Challenges should be filed via the Jakarta Debugging Support for Other Languages 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 with be closed. If the appeal is rejected, the `challenge-appeal` issue should 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 Debugging Support for Other Languages

The Jakarta Debugging Support for Other Languages specification is available from the specification project web-site: https://jakarta.ee/specifications/debugging/2.0/.

2.5 Libraries for Jakarta Debugging Support for Other Languages

The following is a list of the packages comprising the required class libraries for Debugging Support for Other Languages 2.0:

- None

For the latest list of packages, also see:

https://jakarta.ee/specifications/debugging/2.0/

3 Installation

This chapter explains how to install the Jakarta Debugging Support for Other Languages 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

Not required.

3.2 Installing the Software

Before you can run the Debugging Support for Other Languages TCK tests, you must install and set up the following software components:

- Java SE 8
- Debugging Support for Other Languages TCK version 2.0, which includes:
  - VerifySMAP
- The Debugging Support for Other Languages 2.0 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 Debugging Support for Other Languages TCK 2.0 software.
   1. Copy or download the Debugging Support for Other Languages TCK software to your local system.
      You can obtain the Debugging Support for Other Languages TCK software from the Jakarta EE site https://jakarta.ee/specifications/debugging/2.0/.
   2. Use the `unzip` command to extract the bundle in the directory of your choice:
      `unzip jakarta-debugging-tck-2.0.0.zip`
      This creates the TCK directory. The TCK is the test suite home, `<TS_HOME>`.
3. Install the Debugging Support for Other Languages VI to be tested.
   Follow the installation instructions for the particular VI under test.

4 Setup and Configuration

This chapter describes how to set up the Debugging Support for Other Languages TCK. 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 Debugging Support for Other Languages TCK.

4.1 Generating the SMAPs to be Tested

The input to the test is a set of SMAPs. The testing party must generate these SMAPs and they must be generated according to the following procedures. There are two forms of SMAP: an unresolved SMAP in an SMAP file and a resolved SMAP embedded in the SourceDebugExtension attribute of a class file. If unresolved SMAPs are exposed, this SMAP form must be tested. If SMAPs are embedded into class files, this SMAP form (class files containing a resolved SMAP) must be tested. If both forms are exposed, the tests must be repeated with each form.

The Product must be used to create the set of SMAPs to be tested. Generally, the Product is a translator. In this case, a set of test source programs must first be written — see “Generating the SMAPs from Test Source” on page 16. If the Product has more than one input language or more than one output language, the test must be repeated for each combination of input and output langauge. If the Product has no input language, an SMAP for each type of output must be used.

4.1.1 Generating the SMAPs from Test Source

Let us call the input language of the translator LI. A set of test source programs in LI must be written.

The set of test source programs in LI must exercise all control structures in LI, all subroutine invocation mechanisms in LI and all source inclusion mechanisms in LI.. Any of these which do not exist in LI are, of course, excepted.

For each test source program, the Product must be used to generate the output program and its corresponding SMAP. These SMAPs will then be submitted to the TCK test.

4.2 Using the Debugging Support for Other Languages TCK to Test a Product

The following test is applied, one SMAP at a time, to each SMAP generated by the procedures above. The test is executed by launching the Java programming language class VerifySMAP (in dsol-tck.jar)

with the SMAP as an argument:

```
java VerifySMAP -classpath TCK_DIRECTORY/dsol-tck.jar path_to_the_smap
```

For example, to test an unresolved SMAP file pass it to the test:
`java VerifySMAP my.smap` For example, to test a class file with an embedded SMAP pass it to the test:
`java VerifySMAP my.class`

If a test fails an exception will be thrown. If the test of any SMAP fails, the TCK has failed.

5 Assertions

This chapter includes the following topics:

5.1 Assertions Tested with the Debugging Support for Other Languages 2.0 TCK

1. Syntax must be valid, per the grammar in the specification.
2. A resolved SMAP must specify a *DefaultStratumId*.
3. A specified `DefaultStratumId` must either be "Java" or be the `StratumId` of a `StratumSection`.
4. No `StratumSection` may have a `StratumId` of "Java".
5. A `FileSection` may only occur after a `StratumSection`.
6. There must be exactly one `FileSection` after each `StratumSection`.
7. In a `FileSection`, each `FileId` must be unique within that `FileSection`.
8. In a `FileSection`, the `FileName` must be non empty.
9. In a `FileSection`, the `AbsoluteFileName`, if specified, must be non empty.
10. A `LineSection` may only occur after a `StratumSection`.
11. There must be exactly one `LineSection` after each `StratumSection`.
12. In a `LineSection`, `RepeatCount` must be greater than or equal to one.
13. In a `LineSection`, `OutputLineIncrement` must be greater than or equal to zero.
14. In a `LineSection`, `InputStartLine` must be greater than or equal to one.
15. In a `LineSection`, `OutputStartLine` must be greater than or equal to one.
16. In a `LineSection`, `LineFileId` must be a `FileId` in the `FileSection` after the same `StratumSection`.
17. In a `VendorSection`, the `VENDORID` must be well formed, per the specification.
18. `FutureSection` must not be used until defined in the maintenance phase of the JSR.
19. There must be at least one `StratumSection`.
20. An embedded SMAP must not occur in a resolved SMAP.
21. An `OpenEmbeddedSection` must be followed by at least one SMAP, and terminated with `CloseEmbeddedSection`.
22. `StratumId` of `CloseEmbeddedSection` must match `StratumId` of `OpenEmbeddedSection`.