# Test Assertions Guidelines Version 1.0

## Committee Note Draft 02

## 03 May 2013

### Work Product URIs

**This version:**

http://docs.oasis-open.org/tag/guidelines/v1.0/cnd02/guidelines-v1.0-cnd02.pdf (Authoritative)

http://docs.oasis-open.org/tag/guidelines/v1.0/cnd02/guidelines-v1.0-cnd02.html

http://docs.oasis-open.org/tag/guidelines/v1.0/cnd02/guidelines-v1.0-cnd02.odt

**Previous version:**

http://docs.oasis-open.org/tag/guidelines/v1.0/cn01/guidelines-v1.0-cn01.pdf (Authoritative)

http://docs.oasis-open.org/tag/guidelines/v1.0/cn01/guidelines-v1.0-cn01.html

http://docs.oasis-open.org/tag/guidelines/v1.0/cn01/guidelines-v1.0-cn01.odt

**Latest version:**

http://docs.oasis-open.org/tag/guidelines/v1.0/guidelines-v1.0.pdf (Authoritative)

http://docs.oasis-open.org/tag/guidelines/v1.0/guidelines-v1.0.html

http://docs.oasis-open.org/tag/guidelines/v1.0/guidelines-v1.0.odt

**Technical Committee:**

OASIS Test Assertions Guidelines (TAG) TC

**Chairs:**

Jacques Durand (jdurand@us.fujitsu.com), Fujitsu Limited

Patrick Curran (Patrick.Curran@oracle.com), Oracle

**Editors:**

Stephen D. Green (stephengreenubl@gmail.com), Individual

Dmitry Kostovarov (Dmitry.Kostovarov@oracle.com), Oracle

### Related work:

This document is related to:

- *Test Assertions Part 1 - Test Assertions Model Version 1.0*. Latest version. http://docs.oasis-open.org/tag/model/v1.0/testassertionsmodel-1.0.html
- *Test Assertions Part 2 - Test Assertion Markup Language Version 1.0*. Latest version. http://docs.oasis-open.org/tag/taml/v1.0/testassertionmarkuplanguage-1.0.html

### Abstract:

This document provides guidelines and best practices for writing test assertions aligned with the Test Assertions Model.

### Status:

This document was last revised or approved by the OASIS Test Assertions Guidelines (TAG) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/tag/.

### Citation format:

When referencing this Work Product the following citation format should be used:

**[Guidelines-v1.0]**

*Test Assertions Guidelines Version 1.0*. 03 May 2013. OASIS Committee Note Draft 02. http://docs.oasis-open.org/tag/guidelines/v1.0/cnd02/guidelines-v1.0-cnd02.html.

# Table of Contents



# 1 Introduction

This document is a non-normative guidelines document that provides best practices in using the test assertions model defined in [TAM], as well as the test assertions XML mark-up language [TAML]. However, this document alone is sufficient for the most common test assertion design so that the reader does not need to be familiar with the above mentioned Test Assertion model and mark-up specifications. These referred specifications are only necessary for a deeper understanding of test assertion semantics or when using a formal representation of test assertions.

## 1.1 References

**[TAM]** *Test Assertions, Part 1 Test Assertions Model Version 1.0*, May 2011, OASIS Committee Specification Draft 02. http://docs.oasis-open.org/tag/model/v1.0/testassertionsmodel-1.0.pdf

**[TAML]** *Test Assertions, Part 2 Test Assertions Markup Language Version 1.0*, May 2011, OASIS Committee Specification Draft 02. http://docs.oasis-open.org/tag/taml/v1.0/testassertionmarkuplanguage-1.0.pdf

**[ISO/IEC Directives]** *ISO/IEC Directives, Part 2 Rules for the structure and drafting of International Standards*, 2004, International Organization for Standardization.

**[RFC 2119]** S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997, IETF RFC 2119. http://www.ietf.org/rfc/rfc2119.txt.

**[CONF1]** *Conformance Requirements for Specifications v1.0*, March 2002, OASIS Committee Specification 01. http://www.oasis-open.org/committees/download.php/305/conformance_requirements-v1.pdf

**[CONF2]** *Conformance testing and Certification Framework*, June 2001, OASIS white paper. http://www.oasis-open.org/committees/download.php/309/testing_and_certification_framework.pdf

**[CONFCLAUSE]** *Guidelines to Writing Conformance Clauses*, September 2007, OASIS guideline.
http://docs.oasis-open.org/templates/TCHandbook/ConformanceGuidelines.html

**[TESTDEV]** *Test Development FAQ*, 2005, W3C WG note .
http://www.w3.org/QA/WG/2005/01/test-faq

**[VAR]** *Variability in Specifications*, 2005, W3C WG note.
see http://www.w3.org/TR/2005/NOTE-spec-variability-20050831/

**[TMD]** *Test Metadata*, QA Interest Group note, September 2005, W3C.
http://www.w3.org/TR/2005/NOTE-test-metadata-20050914/

## 1.2 About this Document

This document is a guide to test assertions and describes best practices in applying a general test assertion model defined in [TAM]. Its purpose is to help the reader understand what test assertions are, their benefits, and most importantly how they are created. As you will discover test assertions can be an important and useful tool in promoting the quality of specifications, test suites and implementations of specifications. You will learn that there are many ways to create test assertions.

By following the guidelines in this document you will learn how to develop well-defined test assertions that can have useful purposes and applications such as the starting point for a conformance test suite [CONF2] for a specification. Experiences in developing test assertions will be shared, along with lessons learned, helpful tricks and tools, hazards to avoid, and other knowledge that may be helpful in crafting test assertions.

Organization of the document:

**Section 2** describes the rationale for test assertions
**Section 3** describes basic design principles sufficient for simple cases of test assertions
**Section 4** explains the advanced features related to test assertions
**Section 5** provides a worked example culminating in a brief look at the use of test assertion 'markup'
**Section 6** provides a glossary of important terms
**Appendices** provide a listing of related reading material.

## 1.3 Intended Audience

The primary audience for this document is the authors of specifications and the writers of test suites as these groups are most likely to be involved in writing or in understanding test assertions.

## 1.4 Scope

These guidelines are intended to apply to any technology or business field. However, some parts of the "Advanced Features" section take their inspiration from software engineering. The examples describe an arbitrary mechanical device in order to ensure a general understanding of the concepts, whatever the background of the reader. This document is limited to the essentials of test assertions.

# 2 Rationale

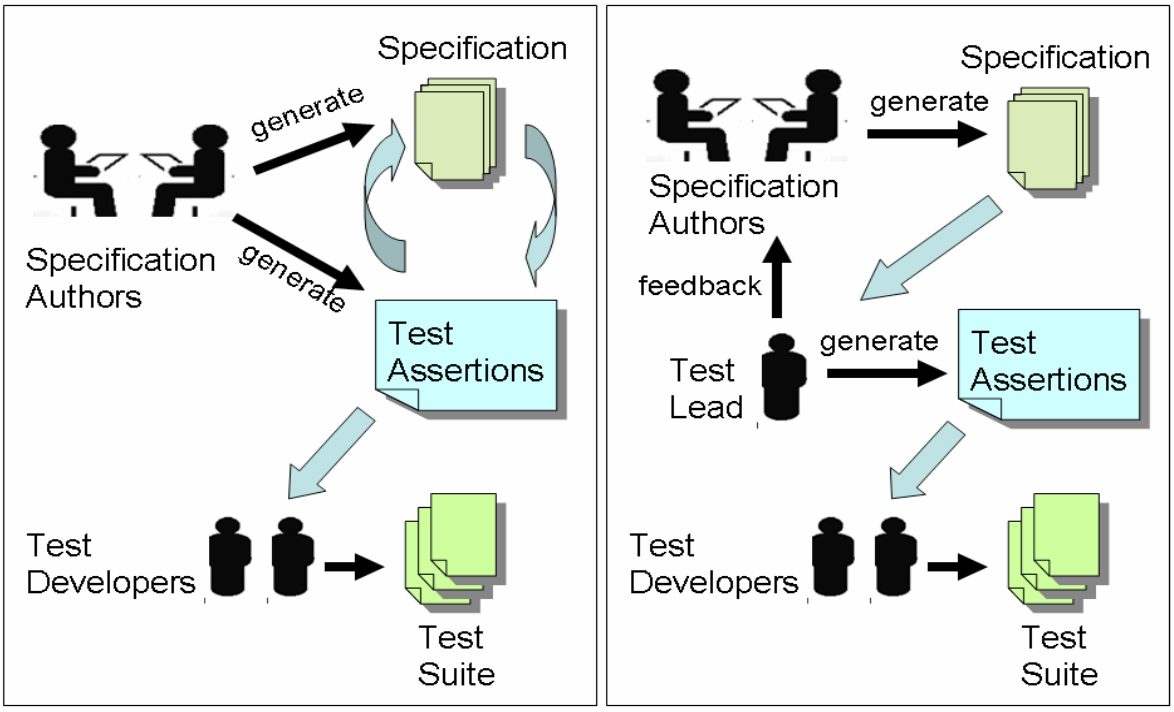
## 2.1 Benefits of test Assertions

**Improving the Specification**

Test assertions may help provide a tighter specification: Any ambiguities, gaps, contradictions and statements which require excessive or impractical resources for testing can be noted as they become apparent during test assertion creation. If there is still an opportunity to correct or improve the specification, these notes can be the basis of comments to the specification authors. If not developed by the specification authors, test assertions should be reviewed and approved by them which will improve both the quality and time-to-deployment of the specification. Therefore, best results are achieved when assertions are developed in parallel with the specification. An alternative is to have the leader of the team that is writing test suites write the test assertions as well and to provide feedback to the specification authors.

Figure 1 illustrates two ways test assertions are being developed. In the first case, test assertions are developed by specification authors concurrently with the target specification. In the second case, they are developed by another party after the specification is complete.

Fig 1. Role of Test Assertions in two examples of usage workflows.



### Facilitating Testing

Test assertions provide a starting point for writing a conformance test suite or an interoperability test suite for a specification that can be used during implementation [TESTDEV]. They simplify the distribution of the test development effort between different organizations while maintaining consistent test quality. By tying test output to specification statements, test assertions improve confidence in the resulting test and provide a basis for coverage analysis (estimating the extent to which the specification is tested).

### Aligning Implementations

Test assertions provide explicit guidance for implementers of a specification, by stating more concretely and practically the conditions to fulfill in order to conform. Unlike test suites which can only be exercised once the implementation work is done, test assertions are usable early on during the implementation work.

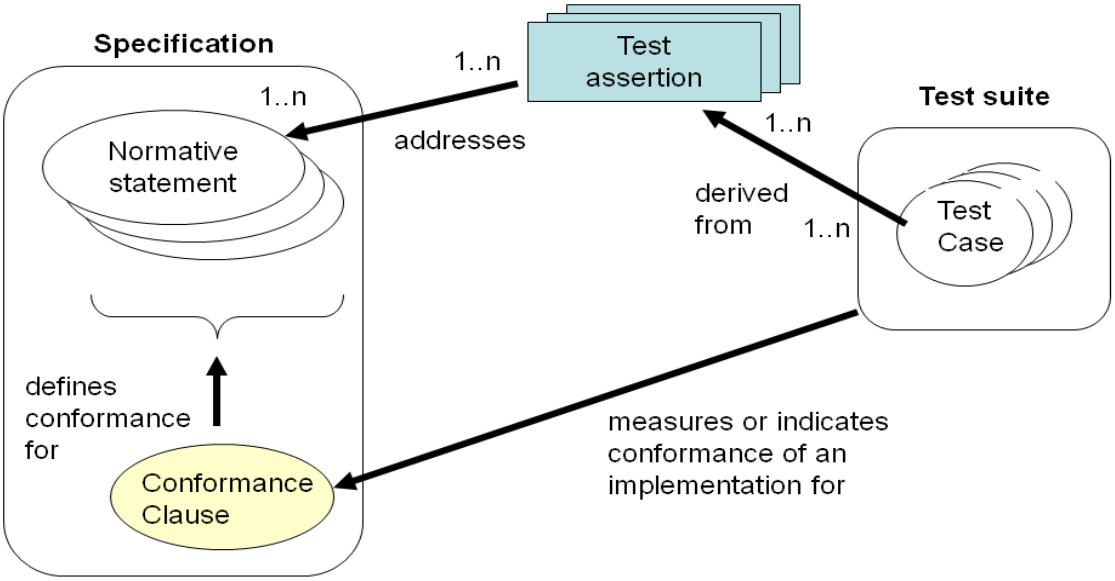
## 2.2 What is a Test Assertion?

A test assertion is a testable or measurable expression for evaluating the adherence of an implementation (or part of it) to a normative statement in a specification.

A Test Assertion should not be confused with a Conformance Clause, nor with a Test Case.

Fig.2 How Test Assertions relate to Specifications and Testing




The specification will often have one or more conformance clauses [CONFCLAUSE] [CONF1] which define various ways to conform to a specification [VAR]. A set of test assertions may be associated with a conformance clause in order to define more precisely what conformance entails for a candidate implementation. Test assertions lie between the specification and any suite of tests to be conducted to determine conformance. Such a test suite is typically comprised of a set of test cases. These test cases are derived from test assertions that address the normative statements of the specification. Figure 2 illustrates how test assertions relate with the concepts of test suite and of specification.

The Glossary, Section 6, will clarify further these related concepts: 'Conformance Clause', 'Test Assertion', 'Test Case', 'Test Metadata'.

## 2.3 A Note on Testability

Judging whether the test assertion is testable may require some knowledge about testing capabilities and resource constraints. Sometimes there is little knowledge of what actual testing conditions will be. In such cases the prime objective of writing test assertions is to provide a better understanding of what is expected from implementations in order to fulfill the requirements. In other cases, the test assertions are designed to reflect a more precise knowledge of testing conditions. Such test assertions can more easily be used as a blueprint for test suites.

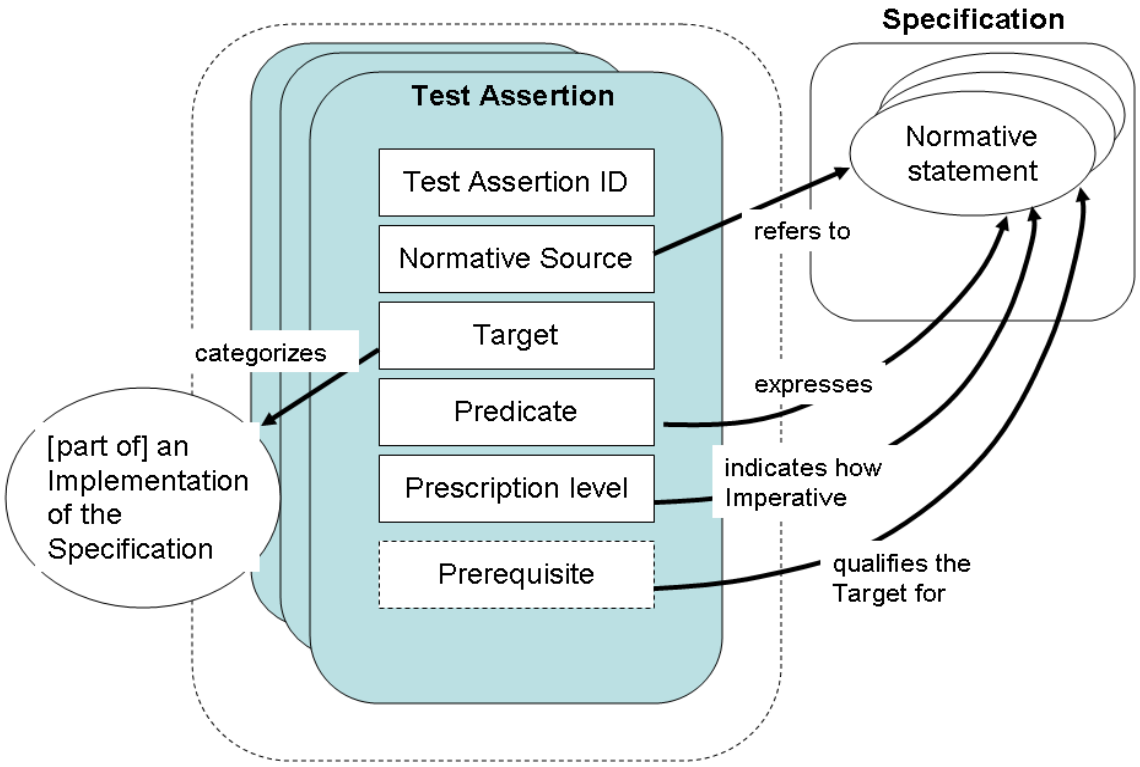
# 3 The Test Assertion Model, Overview and Best Practices

This section aims to cover the simpler aspects of test assertions. Some more complex aspects are covered later in Section 4. The model used here is formally defined in the specification OASIS TAG TC Test Assertions Part 1, Test Assertions Model [TAM].

## 3.1 The Structure of a Test Assertion

Figure 3 below shows the anatomy of a typical test assertion, and how its parts relate to the specification being addressed, as well as to the implementations under test. Some optional parts are not shown in the figure.

Fig.3 General Anatomy of a Test Assertion




Some of the elements which comprise a test assertion are considered core while others are optional.

### 3.1.1 Core Test Assertion Parts

A test assertion – as defined in [TAM] - includes, implicitly or explicitly the following parts :

**Identifier**
A unique identifier of the test assertion. It facilitates the mapping of assertions to specification statements. It is recommended that the identifier be made universally unique.[1]

**Normative Source**
The part of the test assertion that identifies the precise specification requirements or normative statements that the test assertion is addressing.

**Target**
The target – or test target - categorizes an implementation or a part of an implementation of the referred specification, that is the main object of the test assertion and of its Normative Source.

**Predicate**
The predicate is a logical expression that asserts, in a testable form , the feature (a behavior or a property) described in or referenced by the Normative Source, concerning an instance of Target. The Predicate evaluates to either "true" or "false". If the predicate evaluates to "true" over the test assertion Target, this means that the Target exhibits this feature. "False" means the Target does not exhibit this feature.

### 3.1.2 Optional Test Assertion Parts

In addition, a test assertion may optionally include:

**Description**
An informal definition of the role of the test assertion, with some optional details on some of its parts. This description must not alter the general meaning of the test assertion and its parts. This description may be used to annotate the test assertion with any information useful to its understanding. It does not need to be an exhaustive description of the test assertion.

**Prescription Level**
A label in a test assertion that states how imperative it is for a Target instance to satisfy the Predicate condition. Three levels are defined in this specification: **mandatory** (corresponding to normative keywords MUST [NOT] / REQUIRED / SHALL [NOT]), **permitted** (MAY , OPTIONAL) or **preferred** (SHOULD [NOT] / RECOMMENDED). There are differences between various conventions of normative language [ISO/IEC Directives] [RFC 2119] and the above terms may be extended with more specialized terms for a particular convention and its distinct shades of meaning.

**Prerequisite**
The test assertion Prerequisite is a logical expression (similar to a Predicate) which further qualifies the Target for undergoing the core test expressed by the Predicate). It may include references to the outcome of other test assertions. If the Prerequisite evaluates to "false" then the Target instance is not qualified for evaluation by the Predicate. If the Prerequisite

1 One way to do this is to designate a universally unique name for a set of test assertions and to include this name along with the identifier when referencing the test assertion from outside of this set.

evaluates to "true" then the Target instance is qualified for evaluation by the Predicate. Prerequisites have also been called “preconditions” in [TMD].

**Tag(s)**
A Test assertion may be assigned 'tags'. These tags provide an opportunity to categorize the test assertions. They enable the grouping of test assertions, for example based on the type of test they assume or based on their target properties. Tags may be any string – e.g. some predefined 'keywords', which may in turn be given values.

**Variable(s)**
Parameters or symbols employed when writing a test assertion used to refer to values that are not known or fixed at the time the test assertion is written, but will be determined at some later stage, possibly as late as the middle of running a set of tests. A variable is also employed to enable several assertions to share a value (set once, used by many), like a variable in other technologies.

### 3.1.3 General Semantics of a Test Assertion

As a test assertion has parts that can be evaluated over a Target instance (i.e. the Prerequisite and the Predicate), the following semantics apply to a test assertion:
With regard to a Target instance

- The "**Target”** is said **to be not qualified for the Test Assertion** if the Prerequisite (if any) evaluates to "false" over theTarget .
- The "**Target”** is said **to fulfill the Normative Statement addressed by the Test Assertion** if the Prerequisite (if any) evaluates to "true" over the Target , and the Predicate evaluates to "true".
- The "**Target”** is said **to not fulfill the Normative Statement addressed by the Test Assertion** if the Prerequisite (if any) evaluates to "true" over a Target , and the Predicate evaluates to "false".

### 3.1.4 Test Assertion Notation used in the Examples

The examples in this document have represented test assertions using a notation of the form:

Test Assertion:

```
TA_id: gizmo-TA300
Normative_Source: specification requirement 300
Target: electrical-gizmo
Prerequisite: [The gizmo] has a low-battery indicator.
Prescription_Level: mandatory
Predicate: The low-battery indicator of [the gizmo] is a red LED
that is flashing below PERCENT-CHARGE charge.
Tag: target-subcategoryof = widget
Variable: 'PERCENT-CHARGE' the percentage of charge in a battery.
```

NOTE: the optional Description part of a test assertion is not used in any example, and is not subject to the best practices and recommendations of this guidelines document.

Test Assertion List:

```
TA_List_id: A001
List_Description: all assertions describing 'Size' requirements
List_Members:  TA001, TA002, TA003, TA004a, TA004b, TA005, TA006,
TA007, TA008
```

Test Assertions Model [TAM] defines a formal model for test assertions. The following table shows how this model maps to parts in the representation of test assertions used in the examples.

| **Test Assertion as represented in Section 3.1** | **Test Assertion annotation used in the examples in this document** | **Corresponding Entities in the OASIS TAG TC Test Assertions, Part 1 Test Assertions Model** |
|---|---|---|
| Test Assertion | Test Assertion | Class: testAssertion |
| Identifier | TA_id | attribute: testAssertion.id (the 'id' attribute of the testAssertion class) |
| Normative Source | Normative_Source | Class: normativeSource |
| Target | Target | Class: target |
| Predicate | Predicate | Class: predicate |
| Description | | Class: description |
| Prescription Level | Prescription_Level | attribute: prescription.level |
| Prerequisite | Prerequisite | Class: prerequisite |
| Tag | Tag | Class: tag |
| Variable | Variable | Class: variable |
| | | |

## 3.2 Best Practices

### 3.2.1 Basic Guidelines for Test Assertions Parts

Consider the following as a requirement from a specification on “widgets” (we will build on this example throughout these guidelines):

```
[requirement 100] “A widget MUST be of rectangular shape”.
```

Here is a test assertion addressing this requirement:

```
TA id: widget-TA100-1[2]
Normative Source: "widget specification", requirement 100
Target: widget
Predicate: [the widget] is of rectangular shape
Prescription Level: mandatory
```

**About the Normative Source:**

This is where the test assertion refers explicitly to the normative statement(s) from the specification that it is addressing. Ideally, each normative statement should be uniquely identified (e.g. in above example:, "requirement 100"). Often, only section numbers or line numbers are available to identify such normative statements. In some cases it may be more reliable or more convenient to simply repeat the normative statement in the test assertion. Note that a test assertion Predicate may actually map to more than one normative statements, in which case a list of references is appropriate for Normative Source.

**About the Target:**

The Target identifies a category of implementations or of parts of an implementations – here "widget". This means that the test assertion applies to any instance of such a category, or here to any actual "widget" instance.

**About the Predicate:**

The test assertion Predicate is worded as an assertion, not as a requirement (the 'MUST' keyword is absent from the predicate. Instead, it is reflected in the prescription level). It has a clear Boolean value: either the statement is true, or it is false for a particular Target instance. The case of how to show that a test assertion conveys optionality of a specification statement (for example, a statement using keywords SHOULD, MAY, etc.) is examined later.

The Predicate can remain abstract in its expression: no indication of what kind of test procedure will be used, such as how to determine the shape of the widget, need be given. These concrete aspects are normally left to the test cases that can be derived from the test assertions. These test cases will determine the concrete criteria for evaluating a Target instance. However the wording of the Predicate may be made more practical than the wording initially used in the Normative statement, or easier to interpret for a test developer (see the "Testable Predicates" section).

**About the Prescription Level:**

The Prescription Level defines how imperative it is for a Target to satisfy the Predicate. It reflects the conformance keywords used in the narrative of the specification (e.g. MUST, SHOULD, **shall**, **should not**, etc.) There are three core values for this test assertion part, that match the conventions of normative language [ISO/IEC Directives][RFC 2119] :

2 Just as we have done with the examples, it is useful to create and follow a scheme or convention when assigning test assertion identifiers. In the examples in these guidelines, the test assertion identifier is based on a combination of broad target category and specification requirement reference number, suffixed with extra characters because it is worth remembering that there is likely to be a many-to-many relationship between specification requirements and test assertions.

- **mandatory**: this value applies for test assertions which address normative statements that include strongly imperative keywords such as MUST / REQUIRED / **shall**, but also their negative counterparts (MUST NOT, **shall not**...) as the Predicate is where the negative aspect is expressed.
- **preferred:** applies for test assertions which address normative statements that can be interpreted as recommendations, using such keywords as (SHOULD / RECOMMENDED) as well as their negative counterparts.
- **permitted:** applies for test assertions about normative statements that concern optional features about which the specification remains neutral (no recommendation), e.g. using such keywords as e.g MAY , OPTIONAL, **may, need not**) (note: be aware that the negative counterparts "NEED NOT or CAN NOT often have a **mandatory** character).

### 3.2.2 Granularity of Test Assertions

Consider the following specification requirement:

```
[requirement 101] "A widget of medium size MUST use exactly one AA
battery encased in a battery holder."
```

There are actually two requirements here that can be tested separately:

> (requirement 101, part 1) A medium-size widget MUST use exactly one AA battery.
> (requirement 101, part 2) A medium-size widget MUST have a battery holder encasing the battery.

Because of this it is possible to write two test assertions:

```
TA id: widget-TA101-1a
Normative Source: specification requirement 101, part 1
Target: medium-size widget
Predicate: [the widget] uses exactly one AA battery.
Prescription Level: mandatory
```

and

```
TA id: widget-TA101-1b
Normative Source: specification requirement 101, part 2
Target: medium-size widget
Predicate: [the widget] has a battery holder encasing the battery.
Prescription Level: mandatory
```

The granularity of a test assertion is a matter of judgment. A single test assertion instead of two could have been written here, with the predicate: "`[the widget] uses exactly one AA battery AND has a battery holder encasing the battery`". This choice may later have an impact on the outcome of a test suite written to verify the conformance of widgets.

With a single test assertion, a test case derived from this test assertion will not be expected to distinguish between the two failure cases. Using two test assertions - one for each sub-requirement - will ensure that a test suite can assess and report independently about the fulfillment of each sub-requirement. Other considerations such as the different nature of tests implied or the reuse of a test assertion in different conformance profiles [VAR], may also lead to the adoption of “fine-grained” instead of “coarse-grained” test assertions. Usage considerations will dictate the best choice.

### 3.2.3 Handling Optional Statements

Requirement 101 in the widget example in Section 3.2.1 has a mandatory character: It uses the keyword 'MUST' (or it could use 'shall' in bold type) to show that it is an absolute requirement.

Interpreting the outcome of such test assertions is straightforward. Test cases derived from such test assertions can make a clear statement of conformance to the specification for the target being tested: '(test assertion predicate = “true”)' means not only that the target exhibits the specified feature of the specification, but also that the target fulfills a specification requirement, since this feature is required.

However there might be several ways to conform to a specification, also known as dimensions of variability [VAR]. While both conforming, two implementations may not exhibit the same features. This section considers one of the most obvious cases of variability: optional features.

Consider a case where the the specification statement is optional, for example, it uses the keywords 'SHOULD' / 'RECOMMENDED' or 'MAY' / 'OPTIONAL' as follows:

```
[statement 102] “It is RECOMMENDED for a widget to be waterproof.”
[statement 103] “A widget MAY have a metallic casing.”
```

Such (normative) statements cannot be construed as formal requirements – a widget will not fail to conform to the specification if it is not waterproof, or if it has a plastic casing. However, establishing conformance is not the sole objective of test assertions. Some test suites are intended to evaluate the capabilities of an implementation, for example, which options the suite implements, regardless of conformance considerations. Even with a conformance objective in mind, a clear separation shall be made between:

(a) Describing a condition under which a target can exhibit a specified feature. This is the role of the test assertion.
(b) Deciding if a target satisfies a conformance criterion. This is the role of one or more test cases that are derived from a test assertion, the outcome of which might be interpreted according to a conformance profile.

Therefore, test assertions can be written for statements 102 and 103 by simply focusing on the specified feature and its related predicate, ignoring the prescription level in the predicate and including it instead as the separate component of that name, for which suitable values are enumerated (`'mandatory'`, `'preferred'`, `'permitted'`):

```
TA id: widget-TA102-1
Normative Source: specification statement 102
Target: widget
Predicate: [the widget] is waterproof.
Prescription Level: preferred

TA id: widget-TA103-1
Normative Source: specification statement 103
Target: widget
Predicate: [the widget] has a metallic casing.
Prescription Level: permitted
```

The negative keywords 'MUST NOT' (alternatively 'shall not' in bold type**)** and 'SHOULD NOT' are handled as follows: With 'MUST NOT' the 'MUST' determines the prescription level, in this case 'mandatory', and the 'NOT' is transferred to the predicate. For example:

```
[statement 203] "A widget MUST NOT have a painted casing."
TA id: widget-TA203-1
Normative Source: specification statement 203
Target: widget
Predicate: [the widget] does not have a painted casing.
Prescription Level: mandatory
```

Similarly with 'SHOULD NOT':

```
[statement 204] "A widget SHOULD NOT have a plastic battery holder."
```

and

```
TA id: widget-TA204-1
Normative Source: specification statement 203
Target: widget
Predicate: [the widget] does not have a plastic battery holder.
Prescription Level: preferred
```

### 3.2.4 About the Testability of Predicates

Test assertions should be written in a way that facilitates the work of test engineers who have to write test cases from them. For example, predicates may reflect some knowledge of the testing capabilities that will be available to execute a derived test suite. Or, the predicate may be written so that one does not need to be an expert in the original specification in order to understand it.

Consider the following as a requirement from a specification on "widgets" :

> [requirement 100] "A widget MUST be of cuboid shape".

Here is a test assertion addressing this requirement:

```
TA id: widget-TA100-1
Target: widget
Normative Source: "widget specification", requirement 100
Predicate: [the widget] has six facets, and each one of the [the
widget] facets is of rectangular shape.
Prescription Level: mandatory
```

The TA predicate is redefining the“cuboid” property in a more practical way. It assumes here a test environment that has the ability to count facets of a volume and to detect the shape of these facets - e.g. by optical scan. The wording of the predicate takes this knowledge into account, by restating the normative requirement in terms of "facets".

Here the wording of the predicate may also provide guidance to a test suite writer who is not expert in 3D-geometry. As illustrated in our example, the predicate wording must not however modify the semantics of the referred normative source. A domain expert can ensure that the predicate is semantically matching the normative source.

### 3.2.5 Implicit Test Assertion Parts

It was noted earlier that a concrete representation of a test assertion may omit elements provided they are implicit. A common case of implicit test assertion components is the implicit target: When several test assertions relate to the same target, the latter may be described just once as part of the context where the test assertions are defined, so that it does not need to be repeated. For example: all test assertions related to requirements about the widget power supply in a widget specification may be grouped in the section “`Widget Power Supply Requirements`”, suggesting that they share the same target.

The predicate may be implicit: In some specifications where all requirements follow a similar pattern, it is often possible to straightforwardly derive the assertion predicate from a requirement so that the predicate does not need to be explicitly stated every time. One way to do this may be to use a simple rule. Take, for example, requirement 101, part 1 "`A medium-size widget MUST use exactly one AA battery`". Compare this text with the predicate of its test assertion, widget-TA101-1a, "`[the widget] uses exactly one AA battery`". There is so much similarity between the requirement text and the test assertion predicate text that an implementation may decide there is too much overhead in writing the predicate to warrant it and the implementation may then decide to merely use a quotation of the requirement in the normative source as an implicit predicate.

# 4 Common Challenges in using Test Assertions

We have considered the five essential elements of the test assertion: identifier, reference, target, predicate and prescription level. In many cases these elements are sufficient and their use is straightforward. Other cases are more challenging, such as: :

- Normative statements which entail complex predicates
- Normative statements which are known to only be "partially testable"
- Test assertions for assessing some property of a target rather than general conformance
- Preconditions to exercising a test assertion, including the outcome of other test assertions
- Specifications where the normative statements are embedded wholly or partly in tables and diagrams
- Specifications with normative references to other specifications
- Relating test assertions to specific versions of a specification
- Reflecting inheritance and dependencies between specifications in test assertions
- Redundancy of excessively repeated test assertions elements
- Test assertion targets which are categorized differently yet related (inheritance, composition).

## 4.1 Complex Predicates

Recall the previous example requirement:

```
[requirement 101] "A widget of medium size MUST use exactly one AA
battery and be encased in a battery holder."
```

The target can be defined as "a medium-size widget" (as in section 3.2.1 ) or as just "a widget". The latter is a natural decision if the specification requirement uses the wording: "`[requirement 101] If a widget is medium size, then it MUST use exactly one AA battery and be encased in a battery holder.`" For the simplicity of this example, if the two test assertion predicates for `widget-TA101-1a` and `widget-TA101-1b` are combined into one example, one possible outcome is:

```
TA id: widget-TA101-2a
Normative Source: requirement 101
Target: widget
Predicate: if [the widget] is medium-size, then [the widget] uses
exactly one AA battery AND the battery is encased in a battery
holder.
Prescription Level: mandatory
```

The target category is broad (any widget), but the predicate part is really of interest only for a subset of this category (the medium-size widgets). Usage considerations should again drive the decision here: a test suite that is designed to verify all widgets, and does not as-

sume a prior categorization of these into small / medium / large sizes, would be improved with test assertions that only use "`widget`" as the target, such as `widget-TA101-2a` above.

*Note: As an important part of the test assertion, even when the target is implicit, not explicit, the target for each assertion must be clearly identifiable.*

A test assertion predicate may, then, be a Boolean expression - a composition of atomic predicates using logical operators AND, OR, NOT. A test assertion predicate may also be of the kind: "if (condition) then (expression)".

As mentioned before, the Predicate is worded in an abstract way and does not need to hint at a particular evaluation method. However, if a precise criterion for interpreting the battery holder requirement (in above example) is provided in an external specification, either referred to directly by the widget specification or by a related conformance clause, then a test assertion should use this criterion in its predicate. Such a test assertion should then refer not only to the specification requirement in its reference property, but also to the external specification or to the conformance clause that refers to this specification.

Another case where a predicate is more complex is when its conditional expression involves more than one part of an implementation (or implementations). In some cases it is clear which one of these objects must be considered the target, while others are just accessory objects.

Consider the following predicate: "`the [widget price tag] matches the price assigned to the widget in its [catalog entry]`", where price tags and catalog entries are both items that must follow the store policy (in effect the specification). In this case it may be reasonably assumed that the "catalog" content is authoritative over the price tag. The price tag can then be considered as the test target, while the accessory object may be identified by a variable which is then used in the predicate.

Other cases are more ambiguous. Consider the following predicate: "`the [widget price tag] matches the price that is reported on the related [item in list of today's special offers] at the store entrance`". Here it is not clear which one of these often-changing labels must be incriminated in the case of a discrepancy (although whichever is lower will likely prevail should a customer complain). The following approaches are possible:

- Consider a combined target, here a pair [`price tag and promotion item for widget X]` that is identified by the widget reference number. This combination will fail or pass the test.
- Select arbitrarily one object as the target, while the other will be an accessory, for example, identified by a variable. In a derived test case, a predicate failure will inevitably lead to the examination of both, should the accessory object be causing the failure.
- Write two or more similar test assertions using alternately one object and then the other as the target.

## 4.2 Prerequisites

An issue with the previous test assertion (`widget-TA101-2a`) is that it will apply to all widgets. Indeed, the predicate "`if [the widget] is medium-size, then...`" will be evaluated even for those widgets that are NOT medium-size: in such cases, the "`if`" part is simply false, but the entire predicate expression remains true, as the expression "`if A then B`" is logically equivalent to "`not(A) or B`", or more intuitively the predicate is verified in these two cases: case (1) = {A is true and B is also true}, case (2) = {A is false and B is either true or false}.

But clearly, case (2) is of no interest here: in this test assertion we are only interested in those widgets that are `medium-size`. One does not want to see test results reported for this test assertion for non-medium size widgets. Yet with the previous TA (widget-TA101-2a), a successful test result will be reported even when the widget is NOT medium-sized, as this case is made possible by the above predicate conditional wording.

Moreover if all widgets are categorized according to their claimed size prior to testing, then a test case implementing widget-TA101-2a will uselessly repeat the "size" test. In this situation the test assertions written in section 3.2.1 (widget-TA101-1a and widget-TA101-1b) are a better choice than widget-TA101-2a.

Assuming that the size of widgets is not a given but is subject to testing, how can we indicate that a preliminary test on the widget size must be done, and that the test assertion predicate must only apply if the widget is medium-size, meaning that otherwise the test assertion is considered "Not Relevant"? This is done by introducing a prerequisite element in the test assertion:

```
TA id: widget-TA101-2b
Normative Source: requirement 101
Target: widget
Prerequisite: [the widget] is medium-size
Predicate: [the widget] uses exactly one AA battery AND has a bat-
tery holder encasing the battery.
Prescription Level: mandatory
```

The prerequisite element is a logical expression of the same nature as the predicate, which concerns the same target instance.

When the prerequisite evaluates to false then the predicate is not evaluated: no success nor failure is reported for the target but instead a "not qualified" mention.

## 4.3 Test Assertions for Properties

Requirements addressed by test assertions may be related to specific properties of a target. Assume there are specification requirements that define under which conditions a widget qualifies as "medium-size". In other words, widgets do not come with a sticker that makes this categorization obvious by announcing small / medium / large. Instead, the size label is a property that is itself defined in the widget specification and that is subject to verification, like any other normative statement. In such a case, when writing test assertions, it is not a

good idea to consider this property as part of the definition of the target category as in the case `widget-TA101-1a` and `widget-TA101-1b`, because the category of a widget could not be identified prior to doing any test on this widget.
Assume that the following requirement defines the "medium-size" property:

> `[requirement 104] "A widget that weighs between 100g and 300g and is from 5 to 15 centimeters long in its longer dimension, is a medium-size widget."`

There is a major distinction between requirement 104 and requirement 101:

- requirement 101 uses "medium-size" as a prerequisite: its predicates only concern widgets that are already established as medium-size.
- requirement 104 defines how to qualify a test assertion as medium-sized.

The test assertions for requirement 104 can be written as:

```
TA id: widget-TA104-1
Normative Source: specification requirement 104
Target: widget
Predicate: [the widget] weighs between 100g and 300g.
Prescription Level: mandatory
Tag: normative_property = medium-sized
```

A tag, "`normative_property = medium-sized`" is assigned to convey that the test assertion evaluation relates to the property ("medium-size"). (See more about tags in Section 4.7.2 .)

```
TA id: widget-TA104-2
Normative Source: specification requirement 104
Target: widget
Predicate: [the widget] is from 5 to 15 centimeters long in its
longer dimension.
Prescription Level: mandatory
Tag: normative_property = medium-sized
```

The test assertions `widget-TA104-1` and `widget-TA104-2` will be used to derive test cases that verify if the property "medium-size" applies to some widget. A "false" outcome for their predicates is an indicator that the medium-size property does not apply. It is not indicative of a violation of the specification itself. Such test assertions are called in this document "**Property test assertions**" to distinguish them from test assertions that are used as indicators of conformance to a specification.

However, both types of test assertions are designed in the same way, with a predicate that indicates whether or not a target satisfies some feature or property. There is no mention of the "medium-size" property in the predicates of test assertions '`widget-TA104-1`' and '`widget-TA104-2`'. This is because this property is precisely what needs to be established by a test suite containing test cases that are derived from these test assertions. Only when a target (here a widget) evaluates to "true" for these two test assertions, will it be considered medium-size. These test assertions are only concerned with the nature of these tests, not with how to interpret their outcome.

## 4.4 Prerequisites Referring to Other Test Assertions

Now that there is a means to establish the "medium-size" property, we can use a more precise prerequisite element in the test assertion for the requirement 101. Because Test Assertions have been written for testing the property "medium-size", we can refer to these in the Prerequisite:

```
TA_id: widget-TA101-2c
Normative_Source: specification requirement 101
Target: widget
Prerequisite: widget-TA104-1 AND widget-TA104-2
Predicate: [the widget] uses exactly one AA battery AND has a bat-
tery holder encasing the battery.
Prescription_Level: mandatory
```

When a prerequisite element is quoting other test assertions, as seen above, with the prerequisite: (`widget-TA104-1 AND widget-TA104-2`), such references must be understood as short for: (`widget-TA104-1 outcome = 'true' AND widget-TA104-2 outcome = 'true'`).
The prerequisite could have been stated as an explicit predicate in `widget-TA101-2c`. In other words, `widget-TA101-2c` could have been rewritten as:

```
TA_id: widget-TA101-2d
Normative_Source: specification requirement 101
Target: widget
Prerequisite: [the widget] weighs between 100g and 300g AND [the
widget] is from 5 to 15 centimeters long in its longer dimension.
Predicate: [the widget] uses exactly one AA battery AND has a bat-
tery holder encasing the battery.
Prescription_Level: mandatory
```

Here `widget-TA101-2c` is semantically equivalent to `widget-TA101-2d`. However, because the notion of "medium-size" is itself specified as a property that is subject to verification and enforcement, it is useful to write test assertions for this property. It is then preferable to reuse such test assertions as prerequisites whenever this property is assumed. If the notion of medium-size evolves in future releases of the widget specification, the test assertion does not need to be altered: only its prerequisite test assertion needs to be, while all test assertions that explicitly state the prerequisite predicate would need be updated.

## 4.5 Various Normative Sources

The normative content addressed by a test assertion is not always a single, well-identified specification requirement. The normative source may include:

- Multiple (non contiguous) specification statements.
- Non-textual content: tables and diagrams.
- Normative statements that present some testability challenges.

In the previously mentioned cases, it is often useful or necessary to "derive" a new textual statement that will be the actual normative source for the test assertion, and for which the predicate outcome will unequivocally indicate fulfillment or violation. This leads to the use of the DerivedSourceItem element of the "normativeSource" class in the Test Assertions Model [TAM].
This derived statement may in turn be worded so that the Predicate is implicit (see 3.2.2). In the case of "multiple statements", although using several references is possible in the Normative Source element, it is recommended to derive a new consolidated statement.
Identifying the normative source subject to a test assertion may be a delicate exercise, as the source material is often dependent on its context for its meaning. A more complete, self-sufficient statement may be then necessary to clarify the normative source and how it should be interpreted when testing. This may be done using the Interpretation element of the "normativeSource" class in the Test Assertions Model [TAM].
Even when the normative statement is a well identified portion of text in the specification, the following cases may occur:

- The normative source is implicit, as explained in 3.2.2 . This means that there is no explicit element in the test assertion pointing at the specification part that is addressed. This may be the case when the test assertion normative source can be inferred from the location of the test assertion within the specification document itself.
- The specification document is itself expressed as a set of test assertions. This is possible by inserting in the "normative source" part of the test assertion, the normative statement itself instead of a reference to it.

## 4.6 Partially or Non Testable Normative Sources

Sometimes, the normative statement to be addressed by a test assertion can not be fully tested given the state of the art, or given the known constraints imposed to the test environment. In such cases a test assertion may only address "partially" the original normative source in the specification.

For example consider the requirement: "The widget must be of color red". Let us assume that the testability for the color "red" is known as unpractical or unfeasible given the assumed test environment. However, let us assume that testing for "blue" or "yellow" is feasible. We will then write a test assertion(s) that only partially addresses the original requirement ("red") by testing for some of its failure cases.

For this, we **interpret** the original normative statement in a way that only renders parts of its original logic. This is done by adding an interpretation statement to the `Normative Source` element:

```
TA id: widget-TA500
Normative Source: "the widget MUST be of color red". Larger interpretation:"the widget MUST not be blue or yellow."
Target: widget
Predicate: "the [widget] is not blue and is not yellow."
Prescription Level: mandatory
```

The predicate is no longer addressing the original Normative Source, but instead an **interpretation** of it (here `"the widget MUST not be blue or yellow."`). The interpretation in our example is said to be **larger** than the original source because it is accepting of a larger number of widgets than those strictly satisfying the original normative statement. Consequently:

- A predicate value "false" (i.e. concerning a widget that is either blue or yellow) still indicates that the target violates the original normative statement ("the widget MUST be of color red")
- But a predicate value "true" leads to no possible conclusion about the widget, which could or could not be "red", as long as it is not blue and not yellow

The actual normative source addressed by the above test assertion is only a "partial interpretation" of the original source; i.e. a statement that is not equivalent yet still useful for testing in some cases. When addressing a **larger interpretation** of the original normative source, testing is only conclusive in case of failure, not in case of success, hence the partial testability.

Conversely, an interpretation of a Normative Source could be **smaller** than the original source. Reusing our previous example with Normative Source "A widget must be Red", consider a test environment where only one variant of "red" can be tested, e.g. "carmine red". The following interpretation: "A widget MUST be carmine red" is said to be smaller than the original one as it identifies as conforming a smaller set of widgets. The related test assertion will be:

TA id: widget-TA500
Normative Source: "the widget MUST be of color red". **Smaller interpretation:**"the widget MUST be Carmine red."
Target: widget
Predicate: "the [widget] is Carmine red."
Prescription Level: mandatory

This test assertion is using a **smaller** (or intuitively "narrower") interpretation of the original normative source. Its Predicate is stricter than required by the original statement. Consequently:

- A predicate value "true" indicates that the target fulfills the original requirement ("the widget MUST be of color red")
- But a predicate value "false" leads to no possible conclusion about the widget, which could still be "red" even if not "Carmine red".

A clear indication in the Normative Source part that the original normative statement is being interpreted (either in a larger or smaller way) will indicate partial testability. It is a sign for test developers that a test case derived from such a test assertion cannot always produce a decisive outcome. The outcome is only decisive either in case of success (if the interpretation is **smaller**) or in case of failure (if the interpretation is **larger**).

Finally, it may be the case that the Normative Source is considered to be non-testable, to the best of the knowledge of the test assertion writer. For example, it may concern a property or behavior that is not easily observable by a test environment, or that remains privy to an implementation.

Such test assertions still serve a purpose:

- If the specification work is still ongoing, such test assertions may alert specification authors that some of the normative statements may not be testable. This may lead to refining the specification to avoid this.
- At the time an implementation is developed if not tested: they provide guidance to developers by calling out more explicitly the requirements for conformance, and also indicate what kind of feature should be made observable (and testable) when possible, during implementation.

In all cases, it is desirable that every test assertion be usable to derive executable test cases. If the test assertion predicate is to remain "un-testable," the test assertion could be tagged as "not-testable" so that test suite writers can ignore it. Note that as a test environment evolves, non-testability may be temporary.

## 4.7 Test Assertion Grouping

When writing test assertions while the specification is being analyzed, it is typical to group certain test assertions together. You can group assertions that have a special status, such as all accredited test assertions for a given specification, or those that share a particular characteristic, such as a common category of test assertion target.

A special kind of grouping is the container of all test assertions which belong to a particular specification or profile. The container may be the specification document itself if it includes the test assertions to be associated with it. One way to contain test assertions is with a Test Assertion Set . Such a construct may be used to group any number of test assertions for any purpose, not only as a container for all test assertions for the specification or profile. A test assertion set may group test assertions sharing a common test assertion part such as prerequisite.

There are two special ways to group test assertions: - explicitly listing test assertions by their identifiers (section 4.7.1 ) and a more implicit grouping by a common but not unique property such as the tag names or tag values assigned to the test assertions (section 4.7.2 ).

### 4.7.1 Lists

To explicitly identify a group of test assertions they can be listed by their unique test assertion identifiers, along with the logical reason which determines whether a test assertion is a member of that list or not to help with list maintenance.
For example:

```
TA List id: A001
List Description: all assertions describing 'Size' requirements
List Members:  TA001, TA002, TA003, TA004a, TA004b, TA005, TA006,
TA007, TA008
```

Note that if there were a large number of test assertions in this list we might have avoided enumerating each and every test assertion identifier by using an ellipsis ('…') to denote a range of identifiers, `TA001 … TA007`. There is a risk though that a test assertion be added later with identifier, say, of `TA002a` without the intention that it be part of this list. This may be less of a risk with a well thought out identifier scheme.

**Test Assertion Document**

A list of test assertions related to either conformance or interoperability testing will need special care with respect to version control and change management. Therefore, the criteria used to determine which test assertions are members of the list and which are not must be clear. The special case of a container for all test assertions related to a given specification or profile is a special example of an explicit list, although here the method used to define such

a list may involve the use of inclusion of the test assertion itself rather than just its identifier within a special document or package.

One way to create such a list is to include all such related test assertions within a document, called a 'Test Assertion Document'. Other synonymous terms might be 'Test Assertion List', 'Specification Analysis', 'Test Assertion Collection' or 'Test Assertion Set'. Note that the container of this complete set of test assertions might instead be the document of the specification or conformance profile [VAR] itself, when test assertions are included within the text of the actual specification or profile.

### 4.7.2 Tags

Another way to define a group of test assertions is to use a non-unique property of such assertions rather than just using their unique identifiers in a list or containing the test assertions in a document. To this end, test assertions may be assigned metadata in the form of non-unique 'tags' or 'labels'.
For example, test assertion '`widget-TA104-2`', already tagged to show that it is a property test assertion, might be further tagged as 'Size-Property-Description' to allow grouping by this property:

```
TA id: widget-TA104-2
Normative Source: specification requirement 104
Target: widget
Predicate: [the widget] is from 5 to 15 centimeters long in its
longer dimension.
Prescription Level: mandatory
Tag: normative_property = medium-sized
Tag: Size-Property-Description
```

Then it might be included in a list of test assertions related to 'Medium Size' requirements, along with other assertions tagged '`Size-Related`' but NOT with test assertions '`Small-Size-Related`'.

```
TA List id: A002
List Description: all assertions describing 'Medium Size Widget' re-
quirements
List Members:  All test assertions with Tag 'Size-Property-Descrip-
tion' AND Tag 'Size-Related' AND NOT Tag 'Small-Size-Related'
```

This we have called a 'List' but it is in fact defined rather more implicitly than if every member were listed by its identifier, as described in section 4.7.1 . In fact a more explicit and well-defined list might combine both tags and identifiers to group the assertions:

For example:

```
TA List id: A002
List Description: all assertions describing 'Medium Size Widget' re-
quirements
List Description: All test assertions with Tag 'Size-Property-De-
scription' AND Tag 'Size-Related' AND NOT Tag 'Small-Size-Related'
List Members:  TA001, TA002, TA003, TA004a, TA004b, TA005, TA006,
TA007, TA008
```

So a tag is a further, optional test assertion element useful in grouping test assertions. It may sometimes be useful to create tags as name-value pairs.
Note that several such filters can be applied to the same set of assertions and any given assertion can appear in more than one grouping.

Special consideration when using tags for grouping is to be given to the stages in the workflow of test assertion authoring and maintenance and subsequent use at which changes might be made to tags and their values. New tags may be added, perhaps by adding metadata which is separate from the documented test assertion. If metadata for test assertions is defined and maintained separately from the test assertions it may be subject to an entirely different set of version and change control rules and methodologies. In this case, a distinction might need to be made between tags which were part of the original test assertion and those whose list membership might be different to that which was known or expected when the list was defined.

For example, consider a list defined using tags but without explicitly listing test assertion identifiers:

```
TA List id: A002
List Description: all assertions describing 'Medium Size Widget' re-
quirements
List Description: All test assertions with Tag 'Size-Property-De-
scription' AND Tag 'Size-Related' AND NOT Tag 'Small-Size-Related'
```

If `TA004a` is originally tagged '`Size-Related`' but the workflow allows it also to be subsequently tagged '`Small-Size-Related`', then there will need to be rules which determine whether the test assertion is still a member of List '`A002`'.

## 4.8 The Case of Multiple Specifications

Modularity and succinct description within specifications can be achieved by leveraging existing specifications that are referenced by other specifications. Specification writers often create "umbrella specifications". Umbrella specifications are widely scoped specifications that delegate certain normative descriptions to other "referenced specifications".

Specification modularity may also come from a “prototypical specification”, a "base specification" from which specialized derivative specifications may define specific information for a given context.

For proper specification analysis, inclusion of test assertions from both the umbrella or base specification and all the referenced specifications should be considered.

There are three dimensions that describe how assertions from a referenced specification may be included within an umbrella specification:

- 'scope of inclusions',
- 'conditionality of inclusions',
- 'modification of inclusions'.

These relationships between specifications can be expressed using a test assertion. This form of a test assertion is a specific form of assertion that expresses some form of conformance, like a conformance clause.

Multiple dimensions can be expressed within these relationships, for example, a subset of test assertions from a reference spec may be conditionally included in an umbrella specification.

**Scope of Inclusions**

An umbrella specification usually relates to a referenced specification by assuming or requiring conformance of its implementation to this specification. These conformance requirements can be expressed in a test assertion: Indeed, instead of a particular normative statement in a specification, the test assertion can address an entire conformance statement associated with the specification. The conformance statement, whether it concerns the entire specification or just a particular conformance profile, may be expressed in either the prerequisite, or the predicate, or both.

The scope of the conformance may be determined by the expressions in these prerequisites or predicates.

The test assertion may contain a conformance statement as part of its predicate (statement is either true or false) for varying scopes of the (current) umbrella specification as follows:

- conformance to an (entire) umbrella specification
- conformance to a specific normative statement from the umbrella specification

Similarly, the logical expression used in a prerequisite may include a conformance statement (true or false) for varying scopes of the external specification as follows:

- conformance to an (entire) referenced specification
- conformance to a specific test assertion from an referenced specification

Consider the following case where the widget specification (umbrella specification) states that:

"All requirements in this section only apply to "mini" widgets, i.e. Widgets that are conforming to the Mini-Widget Small Box specification 1.2."

Then, in this 'mini widget' section:

[requirement 108] "If a mini-widget has a battery holder, then the mini-widget MUST be labelled as 'low voltage'."

the corresponding test assertion for which is:

TA id: widget-TA108-1
Target: Widget
Normative Source: specification requirement 108
Prerequisite: [the widget] conforms to the Mini-Widget Small Box Specification 1.2 AND [the widget] has a battery holder.
Predicate: [the widget] is labelled 'low voltage'.
Prescription Level: mandatory

If there are known test assertions for the referenced specification (Mini-Widget Small Box Specification 1.2) then the first part of the prerequisite expression could be replaced with a list of the external specification's test assertions. This has the added advantage of allowing a partial inclusion where a target in the umbrella specification conforms to only just a subset of the normative statements in the referenced, external specification. In that case the prerequisite's list might be a subset of the list of all the test assertions for the referenced specification.

**Conditionality of Inclusions**

This dimension of inclusion describes the condition of whether assertions in an umbrella specification conform to a referenced specification. The prerequisite of the assertion may:

(a) Require that optional portions of the referenced specification be implemented in the umbrella,
(b) Conditionally require optional portions of the referenced specification be implemented in the umbrella (for example, based on the presence of hardware or some other such support), or
(c) Make the remaining, required portions of the referenced specification optional.

**Modification of Inclusions**

This dimension of inclusion describes where an umbrella specification conforms to a referenced specification, where some subset of assertions must be modified. This means of inclusion assumes some partitioning of the unchanged assertions and modified assertions. You can use "lists of assertions" to describe in the prerequisite the subset of assertions that the umbrella specification conforms to “unchanged”. The remaining test assertions (the changed set) can be individually specified as test assertions of the umbrella specification.

Typically, assertions are modified in a referenced specification that can be strengthened in a few ways:

- strengthening the prescription level of an assertion, eg. (x) MAY do (y) => (x) MUST do (y), or
- strengthening the meaning of an assertion with additional requirements, eg. IF (x) THEN (z) => IF (x AND y) THEN (z).

## 4.9 Conformance Statements

Predicates, like prerequisites, can use conformance statements in their expression. This is often needed when referring to external specifications, as seen in the previous section about handling multiple specifications.

Conformance statements can be made abstractly without specifying the details of any test assertion that might be involved in assessing this conformance:

```
Predicate: [the widget] conforms to the mini-widget specification
```

In the above case, it is assumed that a precise conformance clause exists that defines what it means to conform to the mini-widget specification.

In other cases, the conformance statement may be defined based on an explicit set of test assertions that are combined into a logical expression. The following Predicate expression is a logical composition of the outcomes of several test assertions:

```
Predicate: TA1 and TA2 and (TA3 or TA4)
```

This form of conformance statement may be appropriate for "properties" that can be complex and involve several test assertions.

Even the property defined in section 4.3 ("medium-size" widget) could be defined by such a predicate in a "conformance stating" test assertion. In the following, the test assertion `widget-TA-mediumsize` is asserting the conditions for a widget in order to qualify as medium-size:

```
TA id: widget-TA-mediumsize
Target: Widget
Normative Source: specification requirement 104
Predicate: [the widget] satisfies widget-TA104-1 AND widget-TA104-2.
Prescription Level: mandatory
```

When an entire set of test assertions has been defined for addressing a particular conformance profile, this test assertion set can be referred to in the Predicate, for example:

```
Predicate: Test Assertion Set (xyz)
```

where there is an implicit AND composition (a logical composition of all the test assertions in the test assertion set).

## 4.10 Specification Versions

Where a specification is the basis for test assertions there needs to be consideration of how to support further versions and maybe any previous versions of that specification. One solution is to create a set of test assertions for each specification version. The references to specifications may include the identification of the precise specification version, but this may restrict that test assertion to just one version of a specification. This simple strategy is less than ideal as it results in a need to re-author the test assertions each time there is a new specification version.

Another method of dealing with multiple specification versions is to create a repository of test assertions for a specification and properly tag each test assertion for the versions of the specification where it is valid. This can be accomplished by introducing two tags, `VersionAdd` and `VersionDrop`.

`tag: VersionAdd`: the lowest numerical version to which the test assertion applies.

`tag: VersionDrop`: the lowest numerical version number to which the test assertion does NOT apply.

Both `VersionAdd` and `VersionDrop` are optional tags. The absence of both tags would mean that the test assertion is valid in all specification versions. If only a `VersionAdd` tag exists and its value is X, the test assertion will be valid in version X of the specification and all subsequent versions. If only a `VersionDrop` tag exists and its value is Y, the test assertion will be valid in all versions of the specification prior to version Y. If both `VersionAdd` and `VersionDrop` tags exist, the test assertion will be valid in version X and all subsequent versions up to but not including version Y. Based on these rules, you can easily generate the set of test assertions that apply to a specific version of the specification.

Care must be taken when going from one version of a specification to another. The test assertion author must identify all test assertions that are the same between versions, dropped from one version, added to one version, and modified between versions. Test assertions that are the same, are dropped, or are added can be handled easily with the `VersionAdd` and `VersionDrop` tags.

Test assertions that are modified are trickier to handle. One could treat the modified test assertions as two separate test assertions and tag them with the appropriate `VersionAdd` and `VersionDrop` tags. Unfortunately, this approach does not provide any indication that the updated assertion is related to the test assertion in the prior specification.
Note that in order to track the evolution of a test assertion, it is important to preserve the test assertion identifiers across revisions of the specification. This is important because it is also important to maintain the relationship between a test assertion and the tests associated with that test assertion.

## 4.11 Variables

Variables have a similar role in test assertions as in other technologies. They provide a means for consistently sharing values across multiple assertions and with other processes. The writer of a set of test assertions can use a variable to assert that all occurrences of that variable must have the same value, even if that value cannot be known at the time the test assertions are written. For example, consider a set of assertions that share a reference to the line (mains) voltage supplied by the local electric utility, where the specific location varies . By declaring a variable for this value and distinguishing the name by, for example, representing it in upper case letters (`UTILITY-VOLTAGE`), each test assertion can use it within elements and expressions.
Note: Some people prefer the term "parameter" and would say that the test assertions in question are "parameterized."

The following example also shows the use of the variable within the predicate. Here, specification requirement 130 depends on the utility voltage, expressed as variable '`UTILITY-VOLTAGE`'.

```
Variable: 'UTILITY-VOLTAGE' the AC voltage the voltage commonly
provided for hand-held electrical appliances and laptop computers.
...
```

```
TA id: widget-TA130-1

Normative Source: specification requirement 130

Target: electrical widget

Predicate:  [the electrical widget] contains an embedded AC adapter
for the UTILITY-VOLTAGE.
Prescription Level: mandatory
```

Variables can be used across elements in a single test assertion, for example, in the prerequisite (see section 3.2 ) and other parts of a test assertion. For example, requirement 130 may be restricted only to widgets that have a compliance requirement for this voltage:

```
Prerequisite: [the electrical widget] is compliant with UTIL-
ITY-VOLTAGE
```

In some cases, the variable has a value known or assigned during test assertion authoring, and is simply used to allow agile resetting of its value. In other cases, the variable can be declared in name only, leaving the value to be assigned at a subsequent stage, such as in an implementation or in a conformance clause for a profile, level or module. The value might be measured during testing and could be associated with a property (see section 4.3 ). For example, if the medium-size property is true for a widget, then the `SIZE` variable is set to the value "medium".

A particular consideration in using variables is variable scope. For example, a grouping construct might be a place to declare variables whose scope only applies to those test assertions associated with that grouping. This allows the same variable and its value to be used across several test assertions while avoiding problems with name clashes in test assertions outside of the variable's scope.

## 4.12 Target Categories

As mentioned in section 3.1 , the Target element of a test assertion generally defines a category of objects or parts of an implementation under test. For example, the test assertion target “widget” represents any object that qualifies as a widget.
It is often the case that different targets or categories are related, for example one target is a subcategory of another target . As a consequence:

- Two test assertions that have different target definitions (i.e. whose target elements define different categories), may apply to the same implementation or part of that implementation.
- In a test assertion, a prerequisite expression may refer to other test assertions. These referenced test assertions may use a target different from the target category of the referring test assertion, assuming these categories are related, e.g. one being a subtype of the other.

In order to allow for the above, target relationships should be explicitly stated. Such dependencies may be defined outside test assertions, such as in an object-oriented model. But you can also note these dependencies in the test assertion itself.

### Subcategories

Consider where a Target named "electrical-gizmo" is a sub-category of the "widget" Target. It follows that all test assertions for widget also apply to electrical-gizmo.
One way to express a dependency is to use tags. In the following example, a tag named "target-subcategoryof" will remind the reader of the test assertion that the electrical-gizmo target is also a widget target since in this case 'electrical-gizmo' is a subcategory of 'widget'; therefore it follows that any target categorized as an 'electrical-gizmo' also belongs to the category 'widget' (but not vice versa):

```
TA id: gizmo-TA300
Normative Source: specification requirement 300
Target: electrical-gizmo
Prerequisite: [The gizmo] has a low-battery indicator.
Prescription Level: mandatory
Predicate: The low-battery indicator of [the gizmo] is a red LED
that is flashing below 25% charge.
Tag: target-subcategoryof = widget
```

Another way to indicate that this target is also a widget, is to use a prefix notation such as 'widget:electrical-gizmo' (instead of just 'electrical-gizmo') for the target element in a test assertion:

```
TA id: gizmo-TA300
Target: widget:electrical-gizmo
```

Both modes of annotation (tag or target prefix) make it clear that an electrical-gizmo is also a widget. As mentioned earlier, this helps grouping test assertions based on the target to which they apply. It also helps deciding what other test assertions can be referred to in a prerequisite element.
For example, knowing that the electrical-gizmo is also a widget the following normative statement applies:

```
[statement 123] A widget MAY have a low-battery indicator.
```

Assuming there is already a test assertion (widget-TA123) written for widgets, addressing the above statement about the presence of a low-battery indicator:

```
TA id: widget-TA123
Normative Source: specification requirement 123
Target: widget
Prescription Level: mandatory
Predicate: [The widget] has a low-battery indicator.
```

then it is possible to reuse the test assertion widget-TA123 as a prerequisite for gizmo-TA300:

```
TA id: gizmo-TA300
Target: widget:electrical-gizmo
   ...
Prerequisite: widget-TA123
```

### Composition

Another kind of target relationship is the Composition relationship: Target T2 is a component of Target T1 if an instance of T1 contains an instance of T2. The second category defines something which belongs to the first category as a component of it. For example, assuming widgets always have at least one switch to control their operation, the target "switch" is a component of the target "widget". Knowledge of this composition relationship brings the same benefits as for subcategories:

- Grouping all test assertions that apply to widgets often should include the test assertions that apply to components of a widget (e.g. the switch)
- A test assertion on the switch of a widget may be used as a prerequisite to a test assertion of the widget itself. For example, addressing the requirement "`the gizmo must be able to stop when the switch is off (Requirement# 400)`" may use as prerequisite a test assertion (called here "`TA-switchworks`") that addresses the requirement "`A switch must cut its electrical circuit when in OFF position`". If satisfied, this requirement would guarantee one can rely on a well-functioning switch, when defining the test for Requirement 400. By using such a prerequisite in the test assertion for Requirement 400, we can ensure that the switch is tested first.

The test assertion below addresses the "effectiveness" requirement on the switch:

```
TA id: TA-switchworks
Normative Source: A switch must cut its electrical circuit when in
OFF position.
Target: switch
Predicate: The electrical circuit on which [the switch] is, is cut
when [the switch] is off.
Prescription Level: mandatory
```

The test assertion below will use `TA-switchworks` as prerequiste, because its own predicate expression over the gizmo target only makes sense if the target's switch satisfies `TA-switchworks`:

```
TA id: TA400
Normative Source: Widget specification requirement 400
Target: gizmo
Prerequisite: TA-switchworks(switch)
Predicate: [the gizmo] stops within five minutes when the switch is
off.
Prescription Level: mandatory
Tag: target-haspart = switch
```

In the above test assertion, the tag (`target-haspart`) is used to identify the switch element referred to by this test assertion and its prerequisite: it must be the switch that is a component of this gizmo.

# 5 Worked Example

Here we simulate a specification for our example “Widget” technology and follow through with the creation of test assertions based on this specification, completing this with a look at how to represent these test assertions using 'markup'.

## 5.1 Example Specification

We first make the examples look a little more like a typical specification extract.

```
...

Section 100

A widget MUST be of rectangular shape, as shown below.

Section 101

A widget of medium size MUST use exactly one AA battery and have a red
button on top (see below).

The mechanisms by which the widget delivers its functionality is not
subject to this specification.

Section 102

It is RECOMMENDED for a widget to be waterproof. If it is not water-
proof then it MUST have a warning label stating that it is not water-
proof.

Section 103
A widget MAY have a metallic casing. If it does have a metallic casing
it MUST have a waterproof coating.

Section 104: Localizations of Widget Size

For implementations of widgets for use in the European Union a widget
that weighs between 100g and 300g and is from 5 to 15 centimeters long
in its longer dimension, is a medium-size widget. However, in USA the
widget is medium-sized if it weighs between 4oz and 12oz and is from 2
inches to 6 inches long.
```

## 5.2 Test Assertions

Once test assertions for the specification have been crafted they can be presented in the form of a test assertion document. One section of the test assertion document, a group covering sections 100 to 104 of the example specification, could be as shown below.

### 5.2.1 Test Assertions for Examples (Sections 100 to 104)

TA id: widget-TA100-1
Normative Source: specification requirement 100
Target: widget
Predicate: [the widget] is of rectangular shape
Prescription Level: mandatory

TA id: widget-TA101-1a
Normative Source: specification requirement 101, part 1
Target: medium-size widget
Predicate: [the widget] uses exactly one AA battery.
Prescription Level: mandatory

TA id: widget-TA101-1b
Normative Source: specification requirement 101, part 2
Target: medium-size widget
Predicate: [the widget] has a red button on top.
Prescription Level: mandatory

TA id: widget-TA102-1
Normative Source: specification statement 102, part 1
Target: widget
Predicate: [the widget] is waterproof.
Prescription Level: preferred

TA id: widget-TA102-2
Normative Source: specification statement 102, part 2
Target: widget
Prerequisite: (widget-TA102-1 = false)
Predicate: [the widget] has a label warning that it is not water-proof.
Prescription Level: mandatory

TA id: widget-TA103-1
Normative Source: specification statement 103, part 1
Target: widget
Predicate: [the widget] has a metallic casing.
Prescription Level: permitted

TA id: widget-TA103-2
Normative Source: specification statement 103, part 2
Target: widget
Prerequisite: widget-TA103-1
Predicate: [the widget] has a waterproof coating over its metallic casing.

```
Prescription Level: mandatory

TA id: widget-TA104-1
Normative Source: specification requirement 104
Target: widget
Predicate: [the widget] weighs between WEIGHT-A and WEIGHT-B.
Prescription Level: mandatory
Tag:normative_property = medium-sized

TA id: widget-TA104-2
Normative Source: specification requirement 104
Target: widget
Predicate: [the widget] is from LENGTH-A to LENGTH-B long in its
longer dimension.
Prescription Level: mandatory
Tag:normative_property = medium-sized
```

### 5.2.2 Variable Scope for Localizations of Widget Sizes

The following variables apply to:

Test Assertion References:

widget-TA104-1

widget-TA104-2

```
Variable: 'GEOPOLITICAL-LOCATION' the geopolitical location of use
of the widget, allowed values being strings enumerated in country
code list ...
Variable: 'WEIGHT-A' a weight and its units. If GEOPOLITICAL-LOCA-
TION is 'US' then WEIGHT-A is 4oz. If GEOPOLITICAL-LOCATION is 'EU'
then WEIGHT-A is 100g.
Variable: 'WEIGHT-B' a weight and its units. If GEOPOLITICAL-LOCA-
TION is 'US' then WEIGHT-B is 12oz. If GEOPOLITICAL-LOCATION is 'EU'
then WEIGHT-B is 300g.
Variable: 'LENGTH-A' a length and its units. If GEOPOLITICAL-LOCA-
TION is 'US' then LENGTH-A is 2 inches. If GEOPOLITICAL-LOCATION is
'EU' then LENGTH-B is 5cm.
Variable: 'LENGTH-B' a length and its units. If GEOPOLITICAL-LOCA-
TION is 'US' then LENGTH-B is 6 inches. If GEOPOLITICAL-LOCATION is
'EU' then LENGTH-B is 15cm.
```

## 5.3 Test Assertions Markup

The examples of test assertions up until this point have been represented as labeled values. An alternative is to use 'markup' with the eXtensible Markup Language (XML). One form of markup for test assertions using XML is the **OASIS TAG TC Test Assertion Markup Language** [TAML]. This might represent the above test assertions as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<testAssertionSet xmlns="http://docs.oasis-open.org/ns/tag/taml-201002/"
          xmlns:taml="http://docs.oasis-open.org/ns/tag/taml-201002/"
    setname=”Test Assertions for Widget Specification 1.1 (Sections 100
to 104)”
    >

    <common> <!-- an extension element for parts and definitions common
to all test assertions, unless override by individual TAs -->
      <normativeSource>
        <refSourceItem>specification requirement 104</refSourceItem>
      </normativeSource>
      <target>widget</target>
      <tag tname="DefinesNormativeProperty">true</tag>
      <tag tname="NormativeProperty">medium-sized</tag>
      <var vname="GEOPOLITICAL-LOCATION"> the geopolitical location of
use of the widget, allowed values being strings enumerated in country
code list ...</var>
      <var vname="WEIGHT-A"> a weight and its units. If GEOPOLITIC-
AL-LOCATION is 'US' then WEIGHT-A is 4oz. If GEOPOLITICAL-LOCATION is
'EU' then WEIGHT-A is 100g.</var>
      <var vname="WEIGHT-B">a weight and its units. If GEOPOLITIC-
AL-LOCATION is 'US' then WEIGHT-B is 12oz. If GEOPOLITICAL-LOCATION is
'EU' then WEIGHT-B is 300g.</var>
      <var vname="LENGTH-A">a length and its units. If GEOPOLITIC-
AL-LOCATION is 'US' then LENGTH-A is 2 inches. If GEOPOLITICAL-LOCATION
is 'EU' then LENGTH-B is 5cm.</var>
      <var vname="LENGTH-B">a length and its units. If GEOPOLITIC-
AL-LOCATION is 'US' then LENGTH-B is 6 inches. If GEOPOLITICAL-LOCATION
is 'EU' then LENGTH-B is 15cm.</var>
    </common>

      <testAssertion id="widget-TA104-1">
        <predicate> [the widget] weighs between WEIGHT-A and WEIGHT-B.
</predicate>
      </testAssertion>

      <testAssertion id="widget-TA104-2">
        <predicate> [the widget] is from LENGTH-A to LENGTH-B long in
its longer dimension</predicate>
      </testAssertion>

    <testAssertion id="widget-TA100-1">
      <normativeSource>
        <refSourceItem>specification requirement 100</refSourceItem>
      </normativeSource>
      <target>widget</target>
      <predicate>[the widget] is of rectangular shape</predicate>
      <prescription level="mandatory" />
    </testAssertion>

    <testAssertion id="widget-TA101-1a">
      <normativeSource>
        <refSourceItem>specification requirement 101, part 1</ref-
SourceItem>
      </normativeSource>
      <target>medium-size widget</target>
```

```
      <predicate>[the widget] uses exactly one AA battery.</predicate>
      <prescription level="mandatory" />
    </testAssertion>

    <testAssertion id="widget-TA101-1b">
      <normativeSource>
        <refSourceItem>specification requirement 101, part 2</ref-
SourceItem>
      </normativeSource>
      <target>medium-size widget</target>
      <predicate>[the widget] has a red button on top.</predicate>
      <prescription level="mandatory" />
    </testAssertion>

    <testAssertion id="widget-TA102-1">
      <normativeSource>
        <refSourceItem>specification statement 102, part 1</ref-
SourceItem>
      </normativeSource>
      <target>widget</target>
      <predicate>[the widget] is waterproof.</predicate>
      <prescription level="preferred" />
    </testAssertion>

    <testAssertion id="widget-TA102-2">
      <normativeSource>
        <refSourceItem>specification statement 102, part 2</ref-
SourceItem>
      </normativeSource>
      <target>widget</target>
      <prerequisite>(widget-TA102-1 = false)</prerequisite>
      <predicate> [the widget] has a label warning that it is not wa-
terproof. </predicate>
      <prescription level="mandatory" />
    </testAssertion>

    <testAssertion id="widget-TA103-1">
      <normativeSource>
        <refSourceItem>specification statement 103, part 1</ref-
SourceItem>
      </normativeSource>
      <target>widget</target>
      <predicate>[the widget] has a metallic casing.</predicate>
      <prescription level="permitted" />
    </testAssertion>

    <testAssertion id="widget-TA103-2">
      <normativeSource>
        <refSourceItem>specification statement 103, part 2</ref-
SourceItem>
      </normativeSource>
      <target>widget</target>
      <prerequisite>widget-TA103-1</prerequisite>
      <predicate> [the widget] has a waterproof coating over its metal-
lic casing. </predicate>
      <prescription level="mandatory" />
    </testAssertion>
```

```
</testAssertionSet>
```

# 6 Glossary

### Conformance
The fulfillment of specified requirements by a product, document, process, or service.

### Conformance Clause
A statement in the Conformance section of a specification that provides a high-level description of what is required for an artifact to conform. The conformance clause may, in turn, refer to other parts of the specification for details. A conformance clause must reference one or more normative statements, directly or indirectly, and may refer to another conformance clause.

### Implementation
A product, document, process, or service that is the realization of a specification or part of a specification.

### Normative Source (of a test assertion)
The specification requirements or normative statements that the test assertion addresses. (See also Section 3.1 .)

### Normative Statement, or Normative Requirement
A statement made in the body of a specification that defines prescriptive properties of an implementation, or requirements about it. The prescription level is reflected by the use of normative keywords such as in [RFC 2119] or [ISO/IEC Directives].

### Predicate (of a test assertion)
An expression that asserts in a testable way the feature (a behavior or a property) expected from a test assertion Target, according to a Normative Statement. (See also Section 3.1 .)

### Prerequisite (of a test assertion)
A logical expression which - when evaluating to “true” - qualifies a particular instance of a test assertion target as relevant the test assertion. (See also Section 3.1 .)

### Prescription Level (of a test assertion)
A keyword that indicates how imperative it is that the Normative Statement referred to in the Normative Source, be met by the Target of a test assertion (See also Section 3.1 .)

### Tag (of a test assertion)
Metadata that allows the grouping of a test assertion and a means to categorize the test assertion targets. (See also Section 3.1 .)

### Test Assertion

A testable expression for evaluating the adherence of part of an implementation to a normative requirement statement in a specification. A test assertion describes the expected output or behavior for the test assertion target within specific operation conditions, in a way that can be measured or tested.

### Test Assertion Document

A container for a complete set of test assertions, often those related to all or part of a specification or conformance profile. In some cases the container is the specification itself with test assertions included within it. Test assertions can be added to the document, removed or changed using a change and version management procedure.

### Target (of a test assertion)

Implementation or part of an implementation that is the main object of a test assertion or test case. (See also Section 3.1 .)

### Test Case

A set of a test tools, software or files (data, programs, scripts, or instructions for manual operations) that verifies the adherence of a test assertion target to one or more normative statements in the specification. Typically a test case is derived from one or more test assertions. Each test case includes: (1) a description of the test purpose (what is being tested - the conditions / requirements / capabilities which are to be addressed by a particular test), (2) the pass/fail criteria, (3) traceability information to the verified normative statements, either as a reference to a test assertion, or as a direct reference to the normative statement.

### Test Metadata

Metadata that is included in test cases to facilitate automation and other processing.

### Variable (of a test assertion)

Variables are used for convenience in storing values for reuse and shared use, and also as parameters of test assertions. (See also Section 3.1 .)

# Appendix A Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

- Dennis Hamilton, Individual Member
- Dmitry Kostovarov, Oracle Corporation
- Dong-Hoon Lim, KIEC
- Hyunbo Cho, Pohang University
- Jacques Durand, Fujitsu
- Kevin Looney, Oracle Corporation
- Kyoung-Rog Yi, KIEC
- Lynne Rosenthal, NIST
- Patrick Curran, Oracle Corporation
- Paul Rank, Oracle Corporation
- Serm Kulvatunyou, NIST
- Stephen D. Green, Document Engineering Services
- Tim Boland, NIST
- Victor Rudometov, Oracle Corporation
- Youngkon Lee, KIEC

# Appendix B Related Material

## Test Assertion Methodologies

W3C work on testing methodology:

Work in W3C lead to the QA Framework: the Specification Guidelines (W3C, November 2004) provides a more general overview of all specification quality and conformance aspects, including the role of test assertions.

http://www.w3.org/TR/qaframe-spec/

A succinct Test Assertion Guide was later drafted in W3C (W3C Editors' Draft, 2006)

http://www.w3.org/2006/03/test-assertion-guide

Another general FAQ about testing from W3C, showing the role of test assertions:

http://www.w3.org/QA/WG/2005/01/test-faq

Unisoft Glossary:

Unisoft have published their own Glossary about testing, that includes a definition and succinct categorization of Assertion. Intended for POSIX standard.

http://www.unisoft.com/glossary.html

The Voting Systems Standard (EAC), (2007):

http://www.eac.gov/files/vvsg/Final-TGDC-VVSG-08312007.pdf

Shows the use of a categorization (classes and sub-classes) of test assertion Targets. The object-oriented classification is used to determine which requirements (and assertions) apply, e.g. by automatic inheritance.

Blogs:

http://pramatr.com/2008/11/06/unit-test-assertions-what-could-possibly-go-wrong/

This blog reports on common practical issues with assertion selection, configuration, and messages for unit testing. Some code examples are given.

Google Test Primer (contains section on assertions):

http://code.google.com/p/googletest/wiki/GoogleTestPrimer

A methodology guide from Google focused on a software development environment. Develops a scripting approach to test assertions. Test assertions are encoded as macros that resemble function calls. One tests a class or function by making such assertions about its behavior.

EARL (W3C), (2009):

http://www.w3.org/WAI/ER/EARL10/WD-EARL10-Guide-20090702

EARL is a W3C initiative to normalize the reporting of Test Results. EARL takes a similar approach to these Test Assertions Guidelines by encoding it's descriptions in XML. Although there is a little overlap where EARL encoded data may describe asserted behavior (Assertions), EARL primarily focuses on test results, whereas these Test Assertions Guidelines focus on describing assertions and their contexts. The two systems can be complimentary.

## Test Assertion Examples

Test Assertions from WS-Interoperability:

http://www.ws-i.org/Testing/Tools/2005/01/BP11_TAD_1-1.htm

This document shows a systematic design of Test Assertions as a basis for test suites that verify conformance to Web Service profiles. As test assertions are better understood by end-users (as opposed to test cases that are derived from them), WS-I test reports directly point at test assertions to indicate reasons for failure or success.

Test Assertions from HTML 4.01:

http://www.w3.org/MarkUp/Test/HTML401/current/assertions/assertions_toc.html

A well-rounded set of test assertions for all aspects of the HTML specification.

Generic Assertions for Manual Testing, RC3:

http://accessibility.freestandards.org/a11yspecs/kbd/kafs-gta-rc3.html

Describes minimal set of test assertions that must be developed to run on an implementation of the Keyboard Access Functional Specification - from Open A11y of Linux Foundation. Assertions are described with identifiers, titles, steps to take, and expected results.

SOAP Version 1.2 Specification Assertions and Test Collection (2003):

http://www.w3.org/TR/soap12-testcollection/

The DejaGnu Testing Framework, POSIX conforming test framework, is based on a use of test assertions

http://www.delorie.com/gnu/docs/dejagnu/dejagnu_6.html

This builds on the POSIX assertions definitions and of particular note is the analysis of outcome interpretations. The present guidelines do not give extensive coverage to this because it is considered as more relevant to test suites where outcomes can be related to the knowledge of testing methods to be used.

Test Assertions at OpenSolaris.Org:

http://opensolaris.org/os/project/zfs-crypto/phase1/testassert/

Lists outlines of several assertions related to Open Solaris. Format of each assertion is as follows: “_stc_assertion_start, ID, DESCRIPTION, STRATEGY” (with numbered steps), and "_stc_assertion_end".

OpenSolaris SCM Migration Project:

http://www.genunix2.org/wiki/index.php/SCM_Test_Assertions

**This page contains the test assertions for the OpenSolaris SCM Migration Project in wiki format. A number of assertions are listed, with edit capability (permission needed).**

# Appendix C Revision History

| Revision | Date | Editor | Changes Made |
|---|---|---|---|
| WD 1 | 12/15/08 | Stephen Green | WD 1 candidate |
| WD 2 | 02/04/09 | Stephen Green | WD 2 for PR #1 |
| WD 3 | 12/15/09 | Stephen Green | WD 3 candidate |
| WD 4 | 02/12/10 | Jacques Durand | WD 4 candidate, for Public Review #2 |
| WD 5 | 08/10/10 | Jacques Durand | WD 5 draft for Public Review #3 |
| WD 6 | 04/10/11 | Jacques Durand | CND1 candidate draft for Public Review |
| CND3 | 08/30/11 | Jacques Durand | Candidate draft for Public Review and CN.<br>-Removed some statements that could be construed as normative, added clearer references to the TA model specification. |
| WD 7 | 04/09/13 | Jacques Durand | Candidate draft for CND.<br>- 3.1.1: edits in definitions of TA parts.<br>- in 3.2 "Best Practices": new sub-section 3.2.1<br>- 3.2.4: New: "Testable Predicates"<br>- 4.6 (new subsection) Partially Testable Normative Sources, finalized.<br>- Section 6 "Glossary": updates, |
| WD 8 | 04/26/13 | Jacques Durand<br>Dennis Hamilton | Candidate draft for CND.<br>- 4.6 updates on section to handle “non-testable” case.<br>- improved reference links to bibliography. |