



Test Assertions Part 2 - Test Assertion Markup Language Version 1.0

Committee Draft 02

30 August 2010

Specification URIs:

This Version:

<http://docs.oasis-open.org/tag/taml/v1.0/cd02/testassertionmarkuplanguage-1.0-cd-02.html>

<http://docs.oasis-open.org/tag/taml/v1.0/cd02/testassertionmarkuplanguage-1.0-cd-02.odt>

<http://docs.oasis-open.org/tag/taml/v1.0/cd02/testassertionmarkuplanguage-1.0-cd-02.pdf>

(Authoritative)

Previous Version:

<http://docs.oasis-open.org/tag/taml/v1.0/cd01/testassertionmarkuplanguage-1.0-cd-01.html>

<http://docs.oasis-open.org/tag/taml/v1.0/cd01/testassertionmarkuplanguage-1.0-cd-01.odt>

<http://docs.oasis-open.org/tag/taml/v1.0/cd01/testassertionmarkuplanguage-1.0-cd-01.pdf>

(Authoritative)

Latest Version:

<http://docs.oasis-open.org/tag/taml/v1.0/testassertionmarkuplanguage-1.0.html>

<http://docs.oasis-open.org/tag/taml/v1.0/testassertionmarkuplanguage-1.0.odt>

<http://docs.oasis-open.org/tag/taml/v1.0/testassertionmarkuplanguage-1.0.pdf> (Authoritative)

Technical Committee:

[OASIS Test Assertions Guidelines \(TAG\)](#)

Chair(s):

Patrick Curran

Jacques Durand

Editor(s):

Stephen D Green

Related Work:

This specification is related to:

[OASIS TAG TC - Test Assertions Model - Version 1.0](#)

[OASIS TAG TC - Test Assertions Guidelines - Version 1.0](#)

Declared XML Namespace(s):

<http://docs.oasis-open.org/tag/ns/taml/201002/>

Abstract:

This defines a markup for writing test assertions.

Status:

This document was last revised or approved by the TAG TC on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved 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/>.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/tag/ipr.php>).

The non-normative errata page for this specification is located at <http://www.oasis-open.org/committees/tag/>.

Notices

Copyright © OASIS® 2008-2010. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", "Test Assertion Markup Language", "OASIS TAML" are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

Table of Contents

1 Introduction.....	5
1.1 Terminology.....	5
1.2 Normative References.....	5
1.3 Non-normative References.....	5
2 Markup Representation of Test Assertions.....	6
2.1 Binding to Test Assertions, Part 1 Test Assertions Model	6
2.2 Conventions Used for Representingin the XML Markup and its Usage.....	7
2.3 Test Assertion.....	8
2.4 Test Assertion Set.....	19
2.5 Reserved Tag Names.....	28
3 XML Schema.....	30
4 Conformance.....	40

1 Introduction

[All text is normative unless otherwise indicated.]

1.1 Terminology

Within this specification, the key words "shall", "shall not", "should", "should not" and "may" are to be interpreted as described in Annex H of [\[ISO/IEC Directives\]](#) if they appear in bold letters.

1.2 Normative References

- [TAM]** OASIS Test Assertions, Part 1 Test Assertions Model Version 1.0 (OASIS TAG TC 2010) <http://docs.oasis-open.org/tag/model/v1.0/testassertionmodel-1.0.pdf>
- [ISO/IEC Directives]** ISO/IEC Directives, Part 2 Rules for the structure and drafting of International Standards, International Organization for Standardization, 2004
- [RFC 2119]** S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. IETF RFC 2119, March 1997. <http://www.ietf.org/rfc/rfc2119.txt>.
- [XSD1]** Henry S. Thompson, David Beech, Murray Maloney, et. al., editors. XML Schema Part 1: Structures. <http://www.w3.org/TR/xmlschema-1/> . World Wide Web Consortium, 2000.
- [XSD2]** Paul V. Biron and Ashok Malhotra, editors. XML Schema Part 2: Datatypes. <http://www.w3.org/TR/xmlschema-2/> . World Wide Web Consortium, 2000.

1.3 Non-normative References

- [XPATH1]** W3C Recommendation, "XML Path Language (XPath) Version 1.0" <http://www.w3.org/TR/xpath> . World Wide Web Consortium, 1999.

2 Markup Representation of Test Assertions

2.1 Binding to Test Assertions, Part 1 Test Assertions Model

This specification defines markup for test assertions conforming to the model defined in the OASIS TAG TC Test Assertions Part 1, Test Assertions Model [TAM] both Section 3 (Test Assertion) and Section 4 (Test Assertion Set).

Each 'class' in the Test Assertions Model is represented by an element of the same or similar name in the Test Assertion Markup Language, with exceptions as follows:

Model Name	Markup Name
Class: testAssertion	element: testAssertion
attribute: variableId	varattribute: id
attribute: language	attribute: lg
Class: normativeSource	element: normativeSource
Class: target	element: target
Class: predicate	element: predicate
Class: prerequisite	element: prerequisite
Class: tag	element: tag
Class: variable	element: var
Class: description	element: description
Class: prescription	element: prescription

There are classes in the Test Assertions Model [TAM] which are associated with the class 'shared'. These classes are suffixed with 'shared' to distinguish them from classes of the same name associated with the 'testAssertion' class. In the Test Assertion Markup Language the names of the complex types which correspond to these 'shared' classes include the 'shared' suffix while the corresponding element names do not.

All element and attributes names are given in lower camel case. Type names consist of the element name with the suffix '_type' appended.

Where the model specifies an attribute named 'content', usually with a base datatype 'string', the markup provides for this either with a base type of `xsd:string` assigned to an element's type (or a datatype derived from `xsd:string` such as `xsd:normalizedString` or `xsd:token`) or by allowing mixed content for the element's type.

Markup cardinalities are the same as those specified in the model.

Elements 'testAssertion', 'testAssertionSet' and 'testAssertionDocumentHeader' are declared as global elements and can be used as top level elements in a markup instance; all other elements are declared locally and are not valid as top level elements in a markup instance.

2.2 Conventions Used for Representing in the XML Markup and its Usage

The namespace prefix in use for the test assertion markup throughout this document is **taml**, representing the namespace: <http://docs.oasis-open.org/tag/ns/taml/201002/>.

It is recommended to use this prefix in all instances of this markup.

In many cases, the XML representation of a mandatory model element - i.e. an attribute or association of cardinality (1..1) - is optional in the markup. This is because such elements, although mandatory, may be implicitly represented and therefore not using the conventional explicit mark-up element intended for them.

Instances of this markup are intended to be used either "standalone" i.e. in documents that do not contain any other markup foreign to this specification, or "embedded", i.e. as elements inside documents the root element of which belongs to a namespace foreign to this specification. Instances of this markup are XML elements representing either test assertions, or test assertion sets.

The compact Relax NG notation is used for representing the XML definitions.

The XPath notation may be used for representing attributes or elements, relative of their containing element, e.g.: `taml:testAssertion/@id` for the attribute 'id' of the `taml:testAssertion` element.

XML Representation:

```
<example
  —id = 'xsd:normalizedString'
  —text ? = 'xsd:string'>
  —Content: Child ?, Sibling *, 'xsd:Any' *
</example>
```

In the example representation above there is an element called 'example' (not a real element, just an example to illustrate the XML representation convention used in this specification). The element has a mandatory attribute, shown without any following symbolic characters to signify that it is mandatory, called 'id' whose content type is the W3C XML Schema [XSD2] datatype 'normalizedString', shown as 'normalizedString' prefixed with the letters 'xsd:' to denote a W3C XML Schema datatype. The element called 'example' has another attribute named 'text' which is shown to be optional by the symbol '?' after the name: 'text?'. The 'example' element has element children called 'Child' and 'Sibling' listed after the label 'Content:' in italics. The '*' asterisk character signifies that the child element 'Sibling' has multiple cardinality and is optional. The '?' question mark character after the child element 'Child' signifies that this element is optional. If the 'example' element did not have elements as children but if its content had been of datatype 'integer' then it the content would have been represented as '*Content: xsd:integer*'.

Finally, the example contains in the *Content* an '*xsd:any*' which indicates that the element concerned can be extended with zero or more additional elements as long as they do not belong to the same namespace as the present markup. In the Test Assertion Markup Language these extra elements will not be validated by W3C XML Schema validation, they are skipped.

XML Representation:

```
<anotherExample
  —choice ? = 1|2
  —{any attributes with non-schema namespace . . .}>
  —Content: Child +, Next, (Sibling|Other) ?
</anotherExample>
```

In the next example, above, an element called 'anotherExample' has an optional attribute named 'choice' whose content can be either the literal value '1' or the literal value '2'. The child element named 'Child' is both mandatory and multiple cardinality, symbolized by the plus character '+' after the name.

Another child element follows the element called 'Child' and is mandatory and singular cardinality, denoted by the absence of any character following the name. There is also a choice of one of two possible other child elements named 'Sibling' and 'Other', the choice being denoted by a single vertical line character between the two elements and the group of choices being shown with round brackets surrounding these two choices. Following the brackets there is a '?' question mark character to denote that either of these is optional. The commas separating the child members of the 'Content' denote that there is a sequence. Spaces separating the child element names would denote that there is no required sequence. This is the convention used throughout this specification of the semantics and usage of the Test Assertion Markup Language.

The full definition of the syntax of the markup language is more formally stated in the XML Schema in Section 3. An XML instance or fragment conforming to this markup **shall** be valid according to the XML Schema designated as normative for this specification (see the conformance clause stated in Section 4).

The example contains the following note in the XML representation

```
{any attributes with non schema namespace . . .}>
```

and this indicates that the element concerned can be assigned any additional attributes as long as they do not belong to the same namespace as the present markup. In the Test Assertion Markup Language these extra attributes will not be validated by W3C XML Schema validation, they are skipped.

NOTE:-

Valid instances of XML representations defined here may omit optional parts as indicated with "?" and "*" (examples: testAssertion/target?, testAssertion/Predicate?). However, test assertion instances so represented **shall** comply with the normative cardinality requirements specified in the OASIS TAG TC Test Assertions Part 1, Test Assertions Model [TAM]. This actually requires¹ in some cases that the optional elements in the XML representation be specified in other ways outside of this particular XML fragment (take, for example, an "implicit" test assertion target).

2.3 Test Assertion

taml:testAssertion

The [taml:testAssertion](#) element is representing the class 'testAssertion' in the Test Assertions Model [TAM]. [Detailed semantics of this class and its elements can be found in \[TAM\].](#)

XML Representation:-

```
<testAssertion
id ? = 'xsd:normalizedString'
lg ? = 'xsd:normalizedString'
schemaVersionId ? = 'xsd:normalizedString'
{any attributes with non schema namespace . . .}>
  Content: normativeSource ?, target ?, prerequisite ?, predicate ?,
prescription ?, description ?, tag *, var *,
report *, 'xsd:Any' *
</testAssertion>
```

Compact Relax NG definition:

```
element taml:testAssertion { testAssertion_def }
testAssertion_def =
  attribute id { xsd:normalizedString }?,
  attribute lg { NCName }?,
```

¹OASIS TAG TC Test Assertions Part 1, Test Assertions Model [TAM] Section 3.1, "Implicit Test Assertion Parts"


```

attribute schemaVersionId { xsd:normalizedString }?,
attribute * { text }*,
element taml:prescription { prescription_def }?,
element taml:var { var_def }*,
element taml:normativeSource { normativeSource_def }?,
element taml:target { target_def }?,
element taml:prerequisite { logicalexpr_def }?,
element taml:predicate { logicalexpr_def }?,
element taml:prescription { prescription_def }?,
element taml:tag { tag_def }*,
element taml:report { report_def }*,
element { anyElement }*

```

~~A test assertion shall have a test assertion identifier unless it is implicit. The 'id' attribute may be implicit (by some special provision in a particular profile or implementation of the Test Assertion Markup Language, for example). If no provision is made for an implicit identifier to be assigned to a test assertion, a test assertion identifier shall be provided for every test assertion using the 'id' attribute of the 'testAssertion' element.~~

Like many of the elements in the Test Assertion Markup Language, the `testAssertion` element has a language attribute, 'lg'. This attribute is used to explicitly declare which prose or expression language is used for the logical expressions in the associated ~~structure element - in that case throughout the test assertion~~. It is possible to declare the language for an individual part of a test assertion such as the predicate or the prerequisite (discussed later). Declaring the language for the test assertion as a whole using the 'lg' attribute of the `testAssertion` element **shall** mean that every part in the test assertion uses that language for its expression. A profile **may** specify a set of language identifiers for use with this attribute.

~~An instance may have the element named 'testAssertion' as the top element. Such an instance or fragment may be a valid test assertion embedded within other markup such as the markup of a specification in which test assertions are embedded.~~

~~The role of a testAssertion element as a top level element is the reason that the testAssertion element has an attribute named 'schemaVersionId'. The testAssertion element attribute 'schemaVersionId' should be used as part of the default Test Assertion Markup Language (version 1) version methodology which assigns a version identifier to every version of the markup language published schema. The version methodology allows that several versions of the schema may use the same namespace when they are considered to be compatible with previous versions using that namespace. These versions are denoted 'minor versions' while 'major versions' of the markup schema have differing namespaces. Test Assertion Markup Language schema 'minor versions' should be distinguished in the element at the top level of an XML instance or fragment (such as a fragment embedded within another markup) by the provision of a version identifier in the 'schemaVersionId' attribute of this element when that top level element is either the 'testAssertion' or 'testAssertionSet' element.~~

~~Each test assertion declares a normative source, a target and a predicate and for this reason the testAssertion element has child elements normativeSource, target and predicate.~~ Conformance to the Test Assertions Model [TAM] requires that a test assertion **shall** have a normative source, a target and a predicate unless either or all of these are implicit. So the `normativeSource`, `target` and `predicate` elements **may** be implicit and also may be inherited from a test assertion set or document ancestor of the test assertion (specified later). ~~Each conforming implementation of the Test Assertion Markup Language should ensure that a normative source, a target and a predicate are implicitly or explicitly provided for each test assertion.~~

~~A test assertion may also declare a prerequisite (either as a single entity comprising all prerequisites together or multiple prerequisites), a prescription level and any number of tags. For this the testAssertion element has optional child elements prerequisite, prescription and tag. The 'tag' element has optional, multiple occurrence.~~ Conformance to the Test Assertions Model [TAM] requires that a test assertion **may** have prerequisite(s), prescription level and tags, either implicitly or

explicitly. It also specifies a part called a variable [represented here by taml:var](#). The markup provides an implementation of the guidelines' variable component as an element which it calls 'var'.

One additional, optional element added for convenience to the usability of the markup and for tool support is 'taml:report'. It [is not a requirement does not correspond to a part defined of in](#) the Test Assertions Model [TAM] but is allowed as a conforming extension.

Example:

[The test assertion below is addressing a requirement about XML schema Naming and Design Rules \(NDR\) from a NIEM specification. It uses XPath as expression language for several of its elements \(predicate, target\) and attributes \(target/@idscheme\).. It concerns targets that are xsd:complexType elements in an XML schema:](#)

```
<taml:testAssertion
  id="TA_R6.1"
  lg="XPath2.0"
  xmlns:taml="http://docs.oasis-open.org/tag/ns/taml/201002/">
  <taml:description>xsd:complexType/_-@mixed value check, as specified
in NIEM</taml:description>
  <taml:normativeSource>[Rule 6-1] Within the schema, an element
xsd:complexType SHALL NOT own the attribute mixed with the value true.
</taml:normativeSource>
  <taml:target type="complexType"
idscheme="fn:concat('complexType:',@name)"/>//xsd:complexType</taml:tar
get>
  <taml:predicate>not(@mixed) or @mixed ne 'true'</taml:predicate>
  <taml:prescription level="mandatory"/>
  <taml:report label="fail" message="Rule 6-1 violation"/>
</taml:testAssertion>
```

taml:normativeSource

[The taml:normativeSource element is representing the class 'normativeSource' in the Test Assertions Model \[TAM\]. Detailed semantics about this class and its elements can be found in \[TAM\].](#)

XML Representation:

```
<normativeSource
—{any attributes with non schema namespace . . .}>
—Content: comment ?, interpretation ?, refSourceItem *, textSourceItem *,
—derivedSourceItem *
—{this element also allows mixed content so text can be included directly as
part of the main element's content}
</normativeSource>
```

Compact Relax NG definition:

```
element taml: normativeSource { normativeSource_def }
normativeSource_def =
  attribute * { text }*,

  element taml:comment { comment_def }?,
  element taml:interpretation { interpretation_def }?,
  element taml:refSourceItem { refSourceItem_def }*,
  element taml:textSourceItem { textSourceItem_def }*,
  element taml:derivedSourceItem { refderivedSourceItem\_def }*,
  text ?
```

The normative source includes an element named 'refSourceItem' so that a reference **may** be used to point to the original text as it exists in the specification itself.

XML Representation:

```
<refSourceItem
  name ? = 'xsd:normalizedString'
  lg ? = 'xsd:normalizedString'
  uri ? = 'xsd:normalizedString'
  documentId ? = 'xsd:normalizedString'
  versionId ? = 'xsd:normalizedString'
  revisionId ? = 'xsd:normalizedString'
  dateString ? = 'xsd:normalizedString'
  resourceProvenanceId ? = 'xsd:normalizedString'
  resourceType ? = 'xsd:normalizedString'
  resourceTypeVersionId ? = 'xsd:normalizedString'
  resourceTypeSchemaId ? = 'xsd:normalizedString'
  resourceTypeSchemaVersionId ? = 'xsd:normalizedString'
  resourceTypeProvenanceId ? = 'xsd:normalizedString'
  {any attributes with non schema namespace . . .}>
  Content: xsd:string
</refSourceItem>
```

Compact Relax NG definition:

```
element taml:refSourceItem { refSourceItem_def }
refSourceItem_def =
  attribute name { xsd:normalizedString }?,
  attribute lg { NCName }?,
  attribute uri { xsd:normalizedString }?,
  attribute documentId { xsd:normalizedString }?,
  attribute versionId { xsd:normalizedString }?,
  attribute revisionId { xsd:normalizedString }?,
  attribute dateString { xsd:normalizedString }?,
  attribute resourceProvenanceId { xsd:normalizedString }?,
  attribute * - xsd:* { text }*,
  text
```

The `refSourceItem` element provides for metadata which **may** be used to specify the identification of a normative source item resource. The `uri` attribute **may** contain a URL or URI [or IRI](#) pointing to the location of the source item. The other metadata attributes includes information about the kind of resource involved and most appropriately its provenance (such as authorship identifiers to certify its authenticity) and version, etc. [The actual content of the `refSourceItem` element may be a string describing informally this source.](#)

An alternative to using a reference to point to the normative source in a specification is to actually quote verbatim the source item so the normative source includes an element named 'textSourceItem' which allows a direct, verbatim quote of the specification text.

XML Representation:

```
<textSourceItem
  extension ? = 'xsd:normalizedString'
  name ? = 'xsd:normalizedString'
  lg ? = 'xsd:normalizedString'
  {any attributes with non schema namespace . . .}>
  Content: xsd:string
</textSourceItem>
```

Compact Relax NG definition:

```
element taml:textSourceItem { textSourceItem_def }
textSourceItem_def =
  attribute name { xsd:normalizedString }?,
  attribute lg { NCName }?,
  attribute * - xsd:* { text }*,
  text
```

An alternative again to quoting verbatim the source item is to derive a form of words equivalent in meaning to the source item and for this the normative source includes an element named 'derivedSourceItem'. This is particularly useful when the source consists of tables, diagrams, graphs or text spread over several parts of the specification. The `derivedSourceItem` element provides for metadata which may be used to specify the identification of the normative source item resource from which the source information has been derived. The `uri` attribute may contain a URL or URI pointing to the location of the source item. The other metadata attributes includes information about the kind of resource involved and most appropriately its provenance (such as authorship identifiers to certify its authenticity) and version, etc. element has a structure similar to the `refSourceItem` element. The main difference with `refSourceItem` is that the content of the `derivedSourceItem` element shall represent the derived re-wording of the source.

XML Representation:-

```
<derivedSourceItem
  name ? = 'xsd:normalizedString'
  lg ? = 'xsd:normalizedString'
  uri ? = 'xsd:normalizedString'
  documentId ? = 'xsd:normalizedString'
  versionId ? = 'xsd:normalizedString'
  revisionId ? = 'xsd:normalizedString'
  dateString ? = 'xsd:normalizedString'
  resourceProvenanceId ? = 'xsd:normalizedString'
  resourceType ? = 'xsd:normalizedString'
  resourceTypeVersionId ? = 'xsd:normalizedString'
  resourceTypeSchemaId ? = 'xsd:normalizedString'
  resourceTypeSchemaVersionId ? = 'xsd:normalizedString'
  resourceTypeProvenanceId ? = 'xsd:normalizedString'
  {any attributes with non schema namespace . . .}>
  Content: xsd:string
</derivedSourceItem>
```

Compact Relax NG definition:

```
element taml:comment { comment_def }
comment_def =
  attribute lg { NCName }?,
  attribute * - xsd:* { text }*,
  text
```

XML Representation:-

```
<comment
  lg ? = 'xsd:normalizedString'
  {any attributes with non schema namespace . . .}>
  Content: 'xsd:normalizedString', 'xsd:Any' *
</comment>
```

The `comment` element **may** be used to simply add comments of any kind (or as further specified in a conformance profile for this markup or a customization thereof) to a normative source test assertion part.

Compact Relax NG definition:

```
element taml:interpretation { interpretation_def }
interpretation_def =
  attribute name { xsd:normalizedString }?,
  attribute lg { NCName }?,
  attribute * - xsd:* { text }*,
  text
```

XML Representation:

```
<interpretation
  lg ? = 'xsd:normalizedString'
  {any attributes with non schema namespace . . .}>
  Content: 'xsd:normalizedString'
</interpretation>
```

The `interpretation` element **may** be used to simply add an alternative description in prose of any kind (or as further specified in a conformance profile for this markup or a customization thereof) to a normative source test assertion part. This allows a prose expression to be added to improve human understanding of its logic.

taml:target

The `taml:target` element is representing the class 'target' in the Test Assertions Model [TAM] . Detailed semantics about this class and its elements can be found in [TAM].

XML Representation:

```
<target
  type ? = 'xsd:normalizedString'
  schemeRef ? = 'xsd:normalizedString'
  lg ? = 'xsd:normalizedString'
  {any attributes with non schema namespace . . .}>
  Content: 'xsd:normalizedString'
</target>
```

Compact Relax NG definition:

```
element taml:target { target_def }
target_def =
  attribute type { xsd:normalizedString }?,
  attribute idscheme { xsd:normalizedString }?,
  attribute lg { xsd:normalizedString }?,
  attribute * - xsd:* { text }*,
  text ?
```

A target can either be a specific item or a category of items. The 'target' element has a 'type' attribute which **should** be used to specify the target category, and this **mayshould** be implemented using a controlled vocabulary, ontology or other classification or taxonomy system used to identify the target category, when defined. Where the scheme for listing or categorizing these types is defined in a document, the identifier, URL or URI for this document **may** be associated with the target using the

~~attribute named 'schemeRef'. A target 'schemeRef id scheme' attribute or, for a set of test assertions, a shared target 'schemeRef id scheme' attribute (see later) **may** be used in cases where the target type scheme is defined using an expression or prose definition within the test assertion or set of test assertions to specify the identity scheme associated with this target type or category. For example, its value can be a function such as an XPath expression, that produces a unique ID for each target instance. In case the test assertion applies to a single target instance (as opposed to a category of targets), the id scheme attribute may contain the identifier of this target.~~

~~The target content is a normalized string. This **may** be an expression in a specialized formal expression language which **may should** be specified identified using the 'lg' attribute. Such an expression or function should identify the set of targets to which the test assertion applies. This content may also be a textual representation of the target instance(s) under consideration. or using a complete conformance profile for that particular use of the markup. The target **shall** be the subject of the test assertion predicate and, during implementation of the test assertion, it **should** be the subject (target) of the corresponding test(s). In cases where more than one target is relevant to a test assertion, additional targets **may** be treated as accessory objects and reference to these made using variables (see variables section below) and combined to form a compound target expression.~~

taml:prerequisite

The taml:prerequisite element is representing the class 'prerequisite' in the Test Assertions Model [TAM] . Detailed semantics about this class and its elements can be found in [TAM].

XML Representation:-

```
<prerequisite
  lg ? = 'xsd:normalizedString'
  {any attributes with non schema namespace . . .}>
  Content: 'xsd:normalizedString'
</prerequisite>
```

Compact Relax NG definition:

```
element taml:prerequisite { logicaexpr_def }
logicaexpr_def =
  attribute lg { xsd:normalizedString }?,
  attribute * - xsd:* { text }*,
  text
```

The prerequisite **may** be expressed using a specialized formal expression language which **may** be specified identified using the 'lg' attribute. The prerequisite content is stating a logical expression or statement to be evaluated (as "true" or "false") over the target, or over some collateral artifact or a set of these, e.g. identified using variables (see later), or over a combination of these.

~~or using a complete conformance profile for that particular use of the markup. The Test Assertions Model [TAM] specifies how the semantics of the test assertion depends on the value of the prerequisite. Section 3.3 states:~~

~~"With regard to a Target instance~~

- ~~• 'Target not qualified': if the Prerequisite (if any) evaluates to 'false' over a Target instance.~~
- ~~• 'Normative statement fulfilled [by the Target]': if the Prerequisite (if any) evaluates to 'true' over a Target instance, and the Predicate evaluates to "true".~~
- ~~• 'Normative statement not fulfilled [by the Target]': if the Prerequisite (if any) evaluates to 'true' over a Target instance, and the Predicate evaluates to 'false'.~~

~~A test assertion predicate **shall** be worded as an assertion, not as a requirement. Any 'MUST' or 'shall' keyword **shall** be absent from the predicate but reflected in the prescription level. The predicate has a clear Boolean value: Either the statement is true, or it is false for a particular target. "These semantics require that the prerequisite evaluates to a boolean logical true or false. The prerequisite element is~~

optional in a test assertion (including where a test assertion prerequisite is inherited from an ancestor test assertion set), unless otherwise made mandatory using a conformance profile of a customization for the markup language. The value of the `prerequisite` element **shall** evaluate to true or false. The prerequisite expression **shall** be used to determine whether (true) or not (false) the target qualifies for the test assertion (in particular for the predicate of the test assertion). The evaluation of the test assertion **shall** conform to the semantics of the outcome of the test assertion specified normatively in the Test Assertions Model [TAM].

taml:predicate

The `taml:predicate` element is representing the class 'predicate' in the Test Assertions Model [TAM]. Detailed semantics about this class and its elements can be found in [TAM].

XML Representation:-

```
<predicate
  lg ? = 'xsd:normalizedString'
  {any attributes with non schema namespace . . .}>
  Content: 'xsd:normalizedString'
</predicate>
```

Compact Relax NG definition:

```
element taml:predicate { logicaexpr_def }
logicaexpr_def =
  attribute lg { xsd:normalizedString }?,
  attribute * - xsd:* { text }*,
  text
```

The predicate **may** be expressed using a specialized formal expression language which **may** be specified using the 'lg' attribute. The predicate content is stating a logical expression or statement to be evaluated (as "true" or "false") over the target, and optionally over a set of collateral artifacts, e.g. identified using variables (see later).

A predicate **shall** be specified for every test assertion. The manner of its expression **shall** include where a test assertion predicate is inherited from an ancestor test assertion set. Therefore the actual predicate element shall be optional as a direct child of a `testAssertion` element (as specified in the markup XML schema) unless otherwise made mandatory using a conformance profile of a customization for the markup language. The value of the `predicate` element **shall** evaluate to true or false. The predicate expression **shall** be used to determine whether (true) or not (false) the target passes the test assertion. As already stated (see 'prerequisite' above), the evaluation of the test assertion **shall** conform to the semantics of the outcome of the test assertion specified normatively in the Test Assertions Model [TAM] or using a complete conformance profile for that particular use of the markup. The Test Assertions Model [TAM] specifies how the semantics of the test assertion depends on the value of the predicate (see 'prerequisite' section above). These semantics require that the predicate evaluates to a boolean logical true or false.

taml:prescription

The `taml:prescription` element is representing the class 'prescription' in the Test Assertions Model [TAM]. Detailed semantics about this class and its elements can be found in [TAM].

XML Representation:-

```
<prescription
  level ? = mandatory|preferred|permitted
  {any attributes with non schema namespace . . .}>
```

~~Content: 'xsd:normalizedString'~~
~~</prescription>~~

Compact Relax NG definition:

```
element taml:prescription { prescription_def }
prescription_def =
  attribute level { "mandatory" | "preferred" | "permitted" }?,
  attribute * - xsd:* { text }*,
  text ?
```

The allowable values for the attribute 'level' of the element prescription **may** be extended beyond the built-in predefined values of mandatory, preferred and permitted. ~~Custom values may be ignored by an implementation.~~

~~The prescription values correspond to the terms used in a specification to denote conformance requirements. [RFC 2119] terms conveying mandatory nature of a statement such as 'MUST' and 'MUST NOT' and in Annex H of [ISO/IEC Directives] terms 'shall', etc **shall** correspond to the prescription level value 'mandatory'. RFC2119 terms conveying optionality with preference such as 'SHOULD' and 'SHOULD NOT', 'RECOMMENDED', etc and ISO/IEC Directive terms 'should', etc **shall** correspond to the prescription level value 'preferred'. RFC2119 terms conveying optionality without preference 'MAY' and ISO/IEC Directive terms 'may', etc **shall** correspond to the prescription level value 'permitted'.~~

~~The RFC2119 terms for preference do not permit non-conformance without a reason and usually the same 'preferred' prescription level is acceptable but in some cases implementers **may** wish to make a distinction by making use of the extension facility and specify further enumeration values.~~

The base datatype of any custom extended enumerations for prescription levels **shall** be W3C XML Schema [XSD2] datatype 'Qname'. Custom enumerations **should** be prefixed with a namespace prefix associated with a namespace declared in the markup. Default namespaces (without a prefix) **shall not** be used.

~~The prescription **shall not** affect the outcome semantics of the test assertion but **may** determine how the outcome affects conformance or otherwise of the implementation to a conformance profile or to the conformance clause of the specification.~~

Besides the use of the 'level' attribute, the element content (xsd:normalizedString) **may** be used to express further or more detailed information regarding the prescription level using prose or as a logical expression.

taml:description

The taml:description element is representing the class 'description' in the Test Assertions Model [TAM]. Detailed semantics about this class and its elements can be found in [TAM].

XML Representation:-

```
<description
  lg ? = 'xsd:normalizedString'
  {any attributes with non schema namespace . . .}>
Content: 'xsd:normalizedString'
</description>
```

Compact Relax NG definition:

```
element taml:description { description_def }
description_def =
  attribute lg { xsd:normalizedString }?,
  attribute * - xsd:* { text }*,
  text
```


The `description` element may be used to add a description in prose of any kind (or as further specified in a conformance profile for this markup or a customization thereof) to a test assertion.

~~or set of test assertions. This may be especially useful when a test assertion is otherwise expressed purely in a specialized, formal, logical language which might not be intended for legibility to human readers; the description allows a prose expression to be added to such a test assertion to improve human understanding of its logic. It may also be used to explain the expressions used in a test assertion and to add comments.~~

taml:tag

The `taml:tag` element is representing the class 'tag' in the Test Assertions Model [TAM]. Detailed semantics about this class and its elements can be found in [TAM].

XML Representation:-

```
<tag
  name = 'xsd:normalizedString'
  lg ? = 'xsd:normalizedString'
  {any attributes with non schema namespace . . .}>
  Content: 'xsd:normalizedString'
</tag>
```

Compact Relax NG definition:

```
element taml:tag { tag_def }
tag_def =
  attribute name { xsd:normalizedString }?,
  attribute lg { xsd:normalizedString }?,
  attribute * - xsd:* { text }*,
  text
```

The content of the `taml:tag` element is representing the "content" attribute of the corresponding class 'tag' in the model.

The `tag/@name` attribute **shall** be used in a tag element and have a non-empty value.

Test Assertions Model [TAM] specifies the use of 'tags' assigned to a test assertion as a means to assign metadata and data of a similar nature. Special examples are to indicate to which versions of a specification the test assertion or set of test assertions applies and to specify that a test assertion or set of test assertions exist to define a particular normative property. The `tag` element of this markup **may** be used to attach such data to a test assertion or test assertion set. See below for the two special normative uses of this element.

taml:var

The `taml:var` element is representing the class 'var' in the Test Assertions Model [TAM]. Detailed semantics about this class and its elements can be found in [TAM].

XML Representation:-

```
<var
  name = 'xsd:normalizedString'
  lg ? = 'xsd:normalizedString'
  {any attributes with non schema namespace . . .}>
  Content: 'xsd:normalizedString'
</var>
```

Compact Relax NG definition:

```

element taml:var { var_def }
var_def =
  attribute name { xsd:normalizedString }?,
  attribute lg { xsd:normalizedString }?,
  attribute * - xsd:* { text }*,
  text

```

The content of the taml:tag element is representing the "content" attribute of the corresponding class 'tag' in the model.[TAM].

The Test Assertions Model [TAM] specifies the use of variables to allow several parts of a test assertion or indeed several test assertions within a set to share a value between them. It also specifies how such variables may have their values supplied as parameters at a stage subsequent to the authoring of the test assertions. The `var` element of this markup **may** be used to implement variables as specified in the Test Assertions Model [TAM].

When declared inside a test assertion, the scope of a variable includes all the parts of this test assertion. Their values **may** be declared with scope across specific test assertions by declaring the variables in the shared part of a set of test assertions of which all the test assertions in scope are members (either by reference to the test assertions or by their inclusion explicitly as descendants of the set or as otherwise specified for test assertion sets (see testAssertionSet section later, below). The variable may be referred to, `var`, value **may** be used within in any part of a test assertion part expression e.g. using a notation such as '\$variable1' where the corresponding variable is named 'variable1'. ~~This is one method but others may be used instead.~~

taml:report

The taml:report element is not representing any class in the Test Assertions Model [TAM]. It is added for convenience when test assertions are expected to contain reporting information to be used by test cases derived from these test assertions.

XML Representation:-

```

<report
  label ? = 'xsd:normalizedString'
  message ? = 'xsd:normalizedString'
  when ? = 'xsd:normalizedString'
  {any attributes with non-schema namespace . . .}>
  Content: 'xsd:normalizedString', 'xsd:Any' *
  {this element also allows mixed content so text can be included directly as
  part of the main element's content}
</report>

```

Compact Relax NG definition:

```

element taml:report { report_def }
report_def =
  attribute label { xsd:normalizedStringNCName }?,
  attribute message { text }?,
  attribute when { text }?,
  attribute lg { xsd:normalizedString }?,
  attribute * - xsd:* { text }*,
  text

```

The `report` element **may** be used to specify what messages and labels are included in any reports generated from any test cases based on the test assertion. It **may** also be used to specify which the outcomes of the test cases result in which reports and under which condition(s) each outcome is to be produced, so the `report` element is optional and of multiple cardinality in the `testAssertion` element.

The combination of allowing both mixed content (text can be interspersed with the XML tags) and extra elements from other namespaces ('xsd:any') means that the content of this element can be a mixture of text and, say, HTML or other simple formatting markup.

The attribute `label` **shall** allow values 'fail', 'pass' and 'notQualified' as content, corresponding to the standard possible outcomes defined in the Test Assertions Model [TAM] as follows:-

- notQualified corresponds to the outcome "Target not qualified".
- pass corresponds to the outcome "Normative statement fulfilled [by the Target]":
- fail corresponds to the outcome "Normative statement not fulfilled [by the Target]":

Further values - e.g. 'warning', 'undetermined' - **may** be added which may be defined in a conformance profile.

The optional `when` attribute **may** be used to state the condition that must be satisfied in order for the outcome described by the `label` attribute, to apply. This attribute is useful when defining new outcomes (values for `label`) beyond the standard possible outcomes ('fail', 'pass' and 'notQualified'), or when a standard outcome uses an interpretation different from the default interpretation, which is:

- outcome `notQualified` is conditioned by the `taml:prerequisite` evaluating to 'false'.
- outcome `pass` is conditioned by the `taml:predicate` evaluating to 'true', and the `taml:prerequisite` (if any) evaluating to 'true'.
- outcome `fail` is conditioned by the `taml:predicate` evaluating to 'false', and the `taml:prerequisite` (if any) evaluating to 'true'.

-

The content of optional attribute `message` **shall** describe the meaning of the assertion outcome, e.g. stated in the attribute **and this message may be used to** provide a report with an error message. A more detailed diagnostic message **may** be provided in the content of the `report` element. Further attributes **may** be defined for the `report` element in a conformance profile.

The optional `when` attribute **may** be used to restrict, beyond the value `label` attribute, the circumstances which **shall** be true for the report to be generated. It **shall** contain an expression which evaluates to true or false.

2.4 Test Assertion Set

taml:testAssertionSet

The `taml:testAssertionSet` element is representing the class 'testAssertionSet' in the Test Assertions Model [TAM]. Detailed semantics about this class and its elements can be found in [TAM].

Compact Relax NG definition:

```
element taml:testAssertionSet { testAssertionSet_def }
testAssertionSet_def =
  attribute id { xsd:normalizedString }?,
  attribute lg { NCName }?,
  attribute schemaVersionId { xsd:normalizedString }?,
  attribute * _taml:*_{ text }*,
  element taml:-testAssertionDocumentHeader { testAssertionDocumentHeader_def }?,
  element taml:shared { shared_def }?,
```

```

element taml:testAssertion { testAssertion_def }*,
element taml:testAssertionRef { testAssertionRef_def }*,
element taml:testAssertionSet { testAssertionSet_def }*,
element taml:testAssertionSelection { testAssertionSelection_def }*,
element taml:testAssertionList {
  element taml:testAssertion { testAssertion_def }* }?,
element { anyElement }*

```

XML Representation:-

```

<testAssertionSet
  id ? = 'xsd:normalizedString'
  lg ? = 'xsd:normalizedString'
  schemaVersionId ? = 'xsd:normalizedString'
  date ? = 'xsd:date'
  time ? = 'xsd:time'
  {any attributes with non schema namespace . . .}>
  Content: testAssertionDocumentHeader ?, shared ?,
  testAssertion *, testAssertionRef *, testAssertionSet *,
  testAssertionSelector ?, 'xsd:Any' *
</testAssertion>

```

The `testAssertionSet` element **may** be used to group together test assertions either by direct inclusion of the test assertions within the test assertion set, using the `taml:testAssertionList`, container or by references to constructs (possibly defined externally) that contain these test assertions, such as `taml:testAssertionRef`, (recursively) `taml:testAssertionSet` or `taml:testAssertionSelection`. their `ids`.

An instance **may** have this as the top element. A test assertion set **may** include any number of other test assertion sets. Care **shall** be taken to avoid infinite recursion: A test assertion set **shall not** include itself as an ancestor. The `testAssertionSet` **may** be assigned an identifier using the 'id' attribute. This allows a drill down from the test assertion set through any ancestor test assertion sets to individual ancestor test assertions when referencing a test assertion or test assertion set.

A test assertion set **may** be used to wrap together all the test assertions in a document. In this case the `testAssertionDocumentHeader` **may** be used once within a document either on its own (i.e. outside the `testAssertionSet` element) or as a direct child of the outermost `testAssertionSet` element. See section later on `testAssertionDocumentHeader`.

Another purpose of the test assertion set is that it **may** be used to provide a set of shared test assertion parts and their values in the same way to more than one test assertion (either to limit repetition or to ensure that the values correspond or to provide scope for variables across such test assertions). (See the section on the 'shared' element below.)

The `testAssertionSelection` child element **may** be used when test assertions are **to be imported from some other source(s), yet only a subset is of interest based on a selection criterion, contained within a document but outside of the test assertion set and these test assertions may be** The `testAssertionSelection` element allows for providing this selection criterion while preserving the reference to the original set of test assertions. The selection criterion may either be a list of test assertion `ids`, or a logical condition e.g. defined by an XPath expression appropriate to the markup such as `'//taml:testAssertion[...]` which identifies for association with the test assertion set all current document test assertions written in the Test Assertion Markup Language or a subset of these. One case where this **may** be used is where such test assertions are distributed throughout a physical or logical document written in another kind of markup.

```

<testAssertionSelector
  lg ? = 'xsd:normalizedString'

```

```
{any attributes with non-schema namespace . . .}>
Content: xsd:string
</testAssertionSelector>
```

taml:shared

The taml:shared element is representing the class 'shared' in the Test Assertions Model [TAM]. [Detailed semantics about this class and its elements can be found in \[TAM\].](#)

Compact Relax NG definition:

```
element taml:shared { shared_def }
shared_def =
  attribute * - taml:* { text }*,
  element taml:prescription { prescription_def }?,
  element taml:var { var_def }*,
  element taml:normativeSource { normativeSource_def }?,
  element taml:target { target_def }?,
  element taml:prerequisite { logicalexpr_def }?,
  element taml:predicate { logicalexpr_def }?,
  element taml:prescription { prescription_def }?,
  element taml:tag { tag_def }*,
  element taml:report { report_def }*,
  element { anyElement }*
```

XML Representation:

```
<shared>
Content: normativeSource ?, target ?, prerequisite ?, predicate ?,
prescription ?, description ?, tag *, var *,
report *, 'xsd:Any' *
</shared>
```

The child element named 'shared' of the testAssertionSet element **may** be used to provide one or more test assertion parts either as [default values for missing parts in the test assertions defined for this set, or as overrides](#) (either overridden by or overriding any corresponding parts of [the same-kind-of](#) test assertions [defined](#) within the set) or as composites (composing as either conjunctions or disjunctions with any corresponding parts of the [same-kind-of](#) test assertions within the set) to all the [descendant](#) test assertions of the test assertion set.

The 'normativeSource', 'target', 'predicate', 'prerequisite', 'prescription', ~~'comment',~~ 'interpretation', the identically named 'tag' elements, the identically named 'var' elements and the 'report', elements, when they are children of this 'shared' element, are extended with a 'conflict' attribute [which can take values](#) as follows:

[shared/normativeSource/@conflict = { conjunction | overriding | overridden }](#)

[shared/target/@conflict = { conjunction | disjunction | overriding | overridden }](#)

[shared/prerequisite/@conflict = { conjunction | disjunction | overriding | overridden }](#)

[shared/predicate/@conflict = { conjunction | disjunction | overriding | overridden }](#)

[shared/prescription/@conflict = { overriding | overridden }](#)

[shared/description/@conflict = { conjunction | disjunction | overriding | overridden }](#)

[shared/tag/@conflict = { conjunction | disjunction | overriding | overridden }](#)

~~shared/var/@conflict = { conjunction | disjunction | overriding | overridden }~~

~~shared/report/@conflict = { conjunction | disjunction | overriding | overridden }~~

~~overriding|overridden->~~

~~—Content: comment ?, interpretation ?, refSourceItem *, textSourceItem *,
——derivedSourceItem *~~

~~—{this element also allows mixed content so text can be included directly as
part of the main element's content}~~

~~</normativeSource>disjunction|shared/normativeSource~~

XML Representation:

~~<normativeSource~~

~~—conflict ? = —conjunction|shared/predicate~~

XML Representation:

~~<predicate~~

~~—lg ? = 'xsd:normalizedString'~~

~~—conflict ? = —conjunction|disjunction|overriding|overridden~~

~~—{any attributes with non schema namespace . . .}>~~

~~—Content: 'xsd:normalizedString'~~

~~</predicate>shared/prerequisite~~

XML Representation:

~~<prerequisite~~

~~—lg ? = 'xsd:normalizedString'~~

~~—conflict ? = —conjunction|disjunction~~

~~—{any attributes with non schema namespace . . .}>~~

~~—Content: 'xsd:normalizedString'~~

~~</prerequisite>shared/target~~

XML Representation:

~~<target~~

~~—type ? = 'xsd:normalizedString'~~

~~—lg ? = 'xsd:normalizedString'~~

~~—schemeRef ? = 'xsd:normalizedString'~~

~~—conflict ? = —conjunction|disjunction|overriding|overridden~~

~~—{any attributes with non schema namespace . . .}>~~

~~—Content: 'xsd:normalizedString'~~

~~</target>shared/prescription~~

XML Representation:

~~<prescription~~

~~—level ? = mandatory|preferred|permitted~~

~~—conflict ? = overriding|overridden~~

~~—{any attributes with non schema namespace . . .}>~~

~~—Content: 'xsd:normalizedString'~~

~~</prescription>~~

~~shared/description~~

XML Representation:

~~<description~~

~~—lg ? = 'xsd:normalizedString'~~

~~—conflict ? = conjunction|disjunction|overriding|overridden~~

~~—{any attributes with non schema namespace . . .}>~~

~~—Content: 'xsd:normalizedString'~~

~~</description>shared/report~~

XML Representation:

~~<report~~

~~—label ? = 'xsd:normalizedString'~~

~~—message ? = 'xsd:normalizedString'~~

```

—when ? = 'xsd:normalizedString'
—conflict ? = conjunction|disjunction|overriding|overridden
{any attributes with non schema namespace . . .}>
—Content: 'xsd:normalizedString', 'xsd:Any' *
—{this element also allows mixed content so text can be included directly as
part of the main element's content}
</report>shared/tag

```

XML Representation:-

```

<tag
—name = 'xsd:normalizedString'
—lg ? = 'xsd:normalizedString'
—conflict ? = conjunction|disjunction|overriding|overridden
{any attributes with non schema namespace . . .}>
—Content: 'xsd:normalizedString'
</tag>

```

shared/var

XML Representation:-

```

<var
—name = 'xsd:normalizedString'
—lg ? = 'xsd:normalizedString'
—conflict ? = conjunction|disjunction|overriding|overridden
{any attributes with non schema namespace . . .}>
—Content: 'xsd:normalizedString'
</var>

```

Whether these test assertion parts compose, with conjunction or disjunction (that is, combine using a logical 'AND' or 'OR' respectively), or override or are overridden by any corresponding test assertion parts of the same kind (and, in the case of 'tag' and 'var', with the same 'name' attribute value) within the test assertion set **shall** depend on the corresponding values of the 'conflict' attribute.

Note that the part elements can each have different sets of allowed values for the 'conflict' attribute.

The values of the 'conflict' attribute **may** be extended. Custom values **may** be ignored by an implementation. The base datatype of the custom extended enumeration for the 'conflict' attribute is W3C XML Schema [XSD2] datatype 'Qname'. Custom enumerations **should** be prefixed with a namespace prefix associated with a namespace declared in the markup. Default namespaces (without a prefix) **shall not** be used.

taml:testAssertionSelection

The taml:testAssertionSelection element is representing the class 'testAssertionSelection' in the Test Assertions Model [TAM]. Detailed semantics about this class and its elements can be found in [TAM].

Compact Relax NG definition:

```

element taml:testAssertionSelection { testAssertionSelection_def }
testAssertionSelection_def =
  attribute name { xsd:normalizedString }?,
  attribute lg { xsd:normalizedString }?,
  attribute expr { xsd:normalizedString }?,
  attribute lg { xsd:normalizedString }?,
  attribute * - xsd:* { text }*,
  element taml:testAssertionSet { testAssertionSet_def }*,
  element taml:testAssertionIdList {
    element taml:testAssertionId { xsd:normalizedString }*_}?

```

text

The attribute 'expr' contains the selection criterion that corresponds to the 'content' attribute in the test assertion model.

The attribute 'lg' corresponds to the 'language' attribute in the test assertion model.

The element `taml:testAssertionSelection/taml:testAssertionIdList` identifies a list of test assertions by their `taml:testAssertion/@id` attribute value.

The element `taml:testAssertionSet` identifies the source set of test assertions, a subset of which must be selected, either by specifying a list of `taml:testAssertionId` instances, or by using the logical expression in `@expr`, or both.

taml:testAssertionRef

The `taml:testAssertionRef` element is representing the class 'testAssertionRef' in the Test Assertions Model [TAM]. Detailed semantics about this class and its elements can be found in [TAM].

Compact Relax NG definition:

```
element taml:testAssertionRef { testAssertionRef_def }
testAssertionRef_def =
  attribute name { xsd:normalizedString }?,
  attribute lg { xsd:normalizedString }?,
  attribute * - xsd:* { text }*,
  element taml:testAssertionResource { testAssertionResource_def }*,
  element taml:testAssertionSetId { xsd:normalizedString }*,
  element taml:testAssertionIdList {
    element taml:testAssertionId { xsd:normalizedString }* }?
  text
```

XML Representation:

```
<testAssertionRef
  lg ? = 'xsd:normalizedString'
  name ? = 'xsd:normalizedString'
  {any attributes with non-schema namespace . . .}>
  Content: testAssertionResource ?, testAssertionSetId ?, testAssertionId ?,
  'xsd:Any' *
</testAssertionRef>
```

A test assertion set may refer to one or more test assertions by their test assertion identifiers or by other means to locate them in external resources, rather than include the test assertions literally within the set. A test assertion set in which references are made to other test assertions outside of the set (whether in the same document or other documents) shall use the `testAssertionRef` child element to do so.

The element `taml:testAssertionRef/taml:testAssertionIdList` identifies a list of referred test assertion by their `taml:testAssertion/@id` attribute value.

The element `taml:testAssertionRef/taml:testAssertionSetId` identifies a referred set of test assertion by its `taml:testAssertionSet/@id` attribute value. The structure of this element allows for the possibility that test assertions may be contained in another document in another location by inclusion of a child element `testAssertionResource`. Other child elements `testAssertionSetId` and `testAssertionId` allow for the possibilities that the test assertion may be within one or more layers of test assertion sets and might only be uniquely identifiable by nesting the identifiers of these sets around the test assertion identifier itself. The `testAssertionSetId` elements can be nested and the

~~testAssertionId nested within the innermost testAssertionSetId. At the same time they also allow for the possibility that the test assertion has an identifier sufficiently unique to only require that test assertion identifier itself.~~

~~The testAssertionRef and the testAssertionResource can each include the testAssertionId as a direct child without the need for further identifiers when they are inappropriate.~~

The testAssertionRef **may** be used to refer to a test assertion set as a whole, rather than a reference to each test assertion individually.

taml:testAssertionResource

The taml:testAssertionResource element is representing the class 'testAssertionResource' in the Test Assertions Model [TAM]. Detailed semantics about this class and its elements can be found in [TAM].

Compact Relax NG definition:

```
element taml:testAssertionResource { testAssertionResource_def }
testAssertionResource_def =
  attribute name { xsd:normalizedString }?,
  attribute lg { xsd:normalizedString }?,
  attribute uri { xsd:normalizedString }?,
  attribute documentId { xsd:normalizedString }?,
  attribute * - xsd:* { text }*,
  text
```

The taml:testAssertionResource/@name attribute allows for giving a name to the external resource.

The content of the taml:testAssertionResource element allows for an informal description of the resource.

XML Representation:

```
<testAssertionResource
  lg ? = 'xsd:normalizedString'
  uri ? = 'xsd:normalizedString'
  documentId ? = 'xsd:normalizedString'
  versionId ? = 'xsd:normalizedString'
  revisionId ? = 'xsd:normalizedString'
  dateString ? = 'xsd:normalizedString'
  resourceProvenanceId ? = 'xsd:normalizedString'
  resourceType ? = 'xsd:normalizedString'
  resourceTypeVersionId ? = 'xsd:normalizedString'
  resourceTypeSchemaId ? = 'xsd:normalizedString'
  resourceTypeSchemaVersionId ? = 'xsd:normalizedString'
  resourceTypeProvenanceId ? = 'xsd:normalizedString'
  {any attributes with non schema namespace . . .}>
  Content: testAssertionId ?, testAssertionSetId ?
</testAssertionResource>
```

~~Here is the metadata which may be used to specify the identification of a resource containing test assertions. The uri attribute **may** contain data to help locate the resource but the expected implementation is one where an identifier or URI is used to point to a repository of some kind which is a more appropriate container for the specific information needed to make the external test assertions available.~~

~~The other metadata attributes include information about the kind of resource involved and most appropriately its provenance (such as authorship identifiers to certify its authenticity) and version, etc.testAssertionId~~

XML Representation:-

```
<testAssertionId
  lg ? = 'xsd:normalizedString'
  ref ? = 'xsd:normalizedString'
  {any attributes with non-schema namespace . . .}>
</testAssertionId>
```

This is a pointer to a test assertion identifier. It is used as part of a reference to a test assertion within a test assertion set. The `ref` attribute **shall** be used to contain the test assertion identifier itself. `testAssertionSetId`

XML Representation:-

```
<testSetId
  lg ? = 'xsd:normalizedString'
  section ? = 'xsd:normalizedString'
  ref ? = 'xsd:normalizedString'
  {any attributes with non-schema namespace . . .}>
  Content: testAssertionId ?, testAssertionSetId ?
</testSetId>
```

This is a pointer to a test assertion set identifier. It is used as part of a reference to a test assertion within a test assertion set or to a test assertion set within another test assertion set.

The `ref` attribute **shall** be used to contain the test assertion set identifier. A section within that test assertion set **may** also be specified where appropriate using the `section` attribute, bearing in mind that the test assertion set (perhaps called by another term) might not be written using this markup.

testAssertionDocumentHeader

The `taml:testAssertionDocumentHeader` element is representing the class `'testAssertionDocumentHeader'` in the Test Assertions Model [TAM]. [Detailed semantics about this class and its elements can be found in \[TAM\].](#)

Compact Relax NG definition:

```
element taml:testAssertionDocumentHeader { testAssertionDocumentHeader_def }
testAssertionDocumentHeader_def =
  attribute * - xsd:* { text }*,
  element taml:common { common_def }
```

XML Representation:-

```
<testAssertionDocumentHeader>
  Content: common, 'xsd:Any' *
</testAssertionDocumentHeader>
```

The `testAssertionDocumentHeader` element **may** be used to provide metadata (author, location, etc) about the specification to which test assertions are associated when such test assertions are interspersed within a document written with a markup other than Test Assertion Markup Language.

The `testAssertionDocumentHeader` element **may**, alternatively, provide a container for metadata about the specification in the outermost `testAssertionSet` of a test assertion document or where an implementation only allows one test assertion set for each document.

An instance **may** have this as the top element.

There **shall** be no more than one `testAssertionDocumentHeader` element used in any given document.

common

The `taml:common` element is representing the class 'common' in the Test Assertions Model [TAM]. Detailed semantics about this class and its elements can be found in [TAM].

Compact Relax NG definition:

```
element taml:testAssertionDocumentHeader { testAssertionDocumentHeader_def }
testAssertionDocumentHeader_def =
  attribute * - xsd:* { text }*,
  element taml:common { common_def }
common_def =
  element taml:sourceDocument { sourceDocument_def } ?,
  element taml:authors { authors_def } ?,
  element taml:location { location_def } ?
sourceDocument_def =
  ( element { anyElement }* | text )
authors_def =
  ( element { anyElement }* | text )
location_def =
  ( element { anyElement }* | text )
```

~~Here some of the XML Representation:-~~

```
<common>
—Content: sourceDocument ?, authors ?, location ?, namespaces ?—
</common>
```

XML Representation:-

```
<sourceDocument
—revision ? = 'xsd:normalizedString'
—version ? = 'xsd:normalizedString'
{any attributes with non schema namespace . . .}>
Content: xsd:string
</sourceDocument>XML Representation:-
<uri>
—Content: 'xsd:normalizedString'
</uri>
```

~~See section on the namespaces element, above.XML Representation:-~~

```
<namespace>
—Content: prefix, uri
{this element also allows mixed content so text can be included directly as
part of the main element's content}
</namespace>
```

~~See section on the namespaces element, above.~~

XML Representation:-

```
<prefix>
—Content: 'xsd:token'
</prefix>
```

~~See section on the namespaces element, above.metadata about the source document to which the test assertions relate is assigned to the document containing those test assertions. The content **should** be the name or other identifier for the specification. The attributes specify its version information.~~

XML Representation:-

```
<namespaces
—revision ? = 'xsd:normalizedString'
```

```

—version ? = 'xsd:normalizedString'>
—Content: namespace*
—{this element also allows mixed content so text can be included directly as
part of the main element's content}
</namespaces>

```

~~The namespaces element caters for a special case of usage of the markup. Here the content **may** be the set of namespaces used when the target implementation is itself XML. The namespaces may be included as the content of the element or the prefixes and URIs of the namespaces may be split into separate elements within the child element called namespace (singular). Other kinds of implementation of the markup **may** omit implementation of this feature.~~

2.5 Reserved Tag Names

~~DefinesNormativeProperty and~~ NormativeProperty

A test assertion **may** be tagged to show that it is a property test assertion used in defining a "property" of an implementation (e.g. a conformance profile) using two the reserved word tag names-

~~DefinesNormativeProperty and~~ NormativeProperty.

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: ~~N~~ormative~~P~~roperty = medium-sized

The Test Assertion Markup Language allows this to be represented as follows

```

<testAssertion id="widget-TA104-2">
. . .
    <predicate> [the widget] is from LENGTH-A to LENGTH-B long in its
longer dimension</predicate>
. . .
    <tag name="DefinesNormativeProperty">true</tag>
    <tag name="NormativeProperty">medium-sized</tag>
</testAssertion>

```

~~A test assertion having a reserved word property tag DefinesNormativeProperty or NormativeProperty **may** have an absence of the 'prescription' element.~~

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 **shall** 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 **shall** be valid in all versions of the specification prior to version Y. If both `VersionAdd` and `VersionDrop` tags exist, the test assertion **shall** be valid in version X and all subsequent versions up to but not including version Y. Based on these rules, the set of test assertions that apply to a specific version of the specification can be determined.

3 XML Schema

```
<xs:schema xmlns="http://docs.oasis-open.org/tag/ns/taml/201002/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://docs.oasis-open.org/tag/ns/taml/201002/"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified" version="1.0">

  <xs:element name="testAssertion" type="testAssertion_type"/>
  <xs:element name="common" type="common_type"/>
  <xs:element name="testAssertionDocumentHeader"
type="testAssertionDocumentHeader_type"/>
  <xs:element name="testAssertionSet" type="testAssertionSet_type"/>

  <xs:simpleType name="codeExtension_type">
    <xs:restriction base="xs:QName">
      <xs:pattern value="\c-[:]]+:\c-[:]]+"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="comment_type">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="lg" type="xs:normalizedString"/>
        <xs:anyAttribute namespace="##any" processContents="skip"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="common_type">
    <xs:sequence>
      <xs:element name="sourceDocument" type="sourceDocument_type"
minOccurs="0"/>
      <xs:element name="authors" type="authors_type" minOccurs="0"/>
      <xs:element name="location" type="location_type" minOccurs="0"/>
      <xs:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="description_type">
    <xs:simpleContent>
      <xs:extension base="xs:normalizedString">
        <xs:attribute name="lg" type="xs:normalizedString"/>
        <xs:anyAttribute namespace="##any" processContents="skip"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:simpleType name="descriptionConflictBaseCode_type">
    <xs:restriction base="xs:normalizedString">
      <xs:enumeration value="overriding"/>
      <xs:enumeration value="overridden"/>
      <xs:enumeration value="conjunction"/>
      <xs:enumeration value="disjunction"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="descriptionConflictCode_type">
```

```

    <xs:union memberTypes="descriptionConflictBaseCode_type
codeExtension_type"/>
  </xs:simpleType>

  <xs:complexType name="descriptionShared_type">
    <xs:simpleContent>
      <xs:extension base="description_type">
        <xs:attribute name="conflict" type="descriptionConflictCode_type"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="interpretation_type">
    <xs:simpleContent>
      <xs:extension base="xs:normalizedString">
        <xs:attribute name="lg" type="xs:normalizedString"/>
        <xs:anyAttribute namespace="##any" processContents="skip"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="namespace_type">
    <xs:sequence>
      <xs:element name="prefix" type="xs:token"/>
      <xs:element name="uri" type="xs:anyURI"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="namespaces_type" mixed="true">
    <xs:sequence>
      <xs:element name="namespace" type="namespace_type" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="normativeSource_type" mixed="true">
    <xs:sequence>
      <xs:element name="comment" type="comment_type" minOccurs="0"/>
      <xs:element name="interpretation" type="interpretation_type"
minOccurs="0"/>
      <xs:element name="refSourceItem" type="refSourceItem_type" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="textSourceItem" type="textSourceItem_type"
minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="derivedSourceItem" type="refSourceItem_type"
minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="skip"/>
  </xs:complexType>

  <xs:simpleType name="normativeSourceConflictBaseCode_type">
    <xs:restriction base="xs:normalizedString">
      <xs:enumeration value="overriding"/>
      <xs:enumeration value="overridden"/>
      <xs:enumeration value="conjunction"/>
      <xs:enumeration value="disjunction"/>
    </xs:restriction>
  </xs:simpleType>

```

```

<xs:simpleType name="normativeSourceConflictCode_type">
  <xs:union memberTypes="normativeSourceConflictBaseCode_type
codeExtension_type"/>
</xs:simpleType>

<xs:complexType name="normativeSourceShared_type" mixed="true">
  <xs:complexContent>
    <xs:extension base="normativeSource_type">
      <xs:attribute name="conflict" type="normativeSourceConflictCode_type"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="predicate_type">
  <xs:simpleContent>
    <xs:extension base="xs:normalizedString">
      <xs:attribute name="lg" type="xs:normalizedString"/>
      <xs:anyAttribute namespace="##any" processContents="skip"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="predicateConflictBaseCode_type">
  <xs:restriction base="xs:normalizedString">
    <xs:enumeration value="overriding"/>
    <xs:enumeration value="overridden"/>
    <xs:enumeration value="conjunction"/>
    <xs:enumeration value="disjunction"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="predicateConflictCode_type">
  <xs:union memberTypes="predicateConflictBaseCode_type
codeExtension_type"/>
</xs:simpleType>

<xs:complexType name="predicateShared_type">
  <xs:simpleContent>
    <xs:extension base="predicate_type">
      <xs:attribute name="conflict" type="predicateConflictCode_type"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="prerequisite_type">
  <xs:simpleContent>
    <xs:extension base="xs:normalizedString">
      <xs:attribute name="lg" type="xs:normalizedString"/>
      <xs:anyAttribute namespace="##any" processContents="skip"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="prerequisiteConflictBaseCode_type">
  <xs:restriction base="xs:normalizedString">
    <xs:enumeration value="conjunction"/>
    <xs:enumeration value="disjunction"/>
  </xs:restriction>
</xs:simpleType>

```



```

<xs:simpleType name="prerequisiteConflictCode_type">
  <xs:union memberTypes="prerequisiteConflictBaseCode_type
codeExtension_type"/>
</xs:simpleType>

<xs:complexType name="prerequisiteShared_type">
  <xs:simpleContent>
    <xs:extension base="prerequisite_type">
      <xs:attribute name="conflict" type="prerequisiteConflictCode_type"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="prescription_type">
  <xs:simpleContent>
    <xs:extension base="xs:normalizedString">
      <xs:attribute name="level" type="prescriptionLevelCode_type"/>
      <xs:anyAttribute namespace="##any" processContents="skip"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="prescriptionConflictBaseCode_type">
  <xs:restriction base="xs:normalizedString">
    <xs:enumeration value="overriding"/>
    <xs:enumeration value="overridden"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="prescriptionConflictCode_type">
  <xs:union memberTypes="prescriptionConflictBaseCode_type
codeExtension_type"/>
</xs:simpleType>

<xs:complexType name="prescriptionShared_type">
  <xs:simpleContent>
    <xs:extension base="prescription_type">
      <xs:attribute name="conflict" type="prescriptionConflictCode_type"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="prescriptionLevelCode_type">
  <xs:union memberTypes="prescriptionLevelBaseCode_type
codeExtension_type"/>
</xs:simpleType>

<xs:simpleType name="prescriptionLevelBaseCode_type">
  <xs:restriction base="xs:normalizedString">
    <xs:enumeration value="mandatory"/>
    <xs:enumeration value="permitted"/>
    <xs:enumeration value="preferred"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="refSourceItem_type">
  <xs:simpleContent>
    <xs:extension base="xs:normalizedString">
      <xs:attribute name="lg" type="xs:normalizedString"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

```

```

    <xs:attributeGroup ref="resource_attributeGroup"/>
    <xs:anyAttribute namespace="##any" processContents="skip"/>
  </xs:extension>
</xs:simpleContent>
</xs:complexType>

<xs:complexType name="report_type" mixed="true">
  <xs:sequence>
    <xs:any namespace="##any" processContents="skip" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="label" type="xs:normalizedString"/>
  <xs:attribute name="message" type="xs:normalizedString"/>
  <xs:attribute name="when" type="xs:normalizedString"/>
  <xs:anyAttribute namespace="##any" processContents="skip"/>
</xs:complexType>

<xs:simpleType name="reportConflictBaseCode_type">
  <xs:restriction base="xs:normalizedString">
    <xs:enumeration value="conjunction"/>
    <xs:enumeration value="disjunction"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="reportConflictCode_type">
  <xs:union memberTypes="reportConflictBaseCode_type codeExtension_type"/>
</xs:simpleType>

<xs:complexType name="reportShared_type" mixed="true">
  <xs:complexContent>
    <xs:extension base="report_type">
      <xs:attribute name="conflict" type="reportConflictCode_type"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:attributeGroup name="resource_attributeGroup">
  <xs:attribute name="name" type="xs:normalizedString"/>
  <xs:attribute name="uri" type="xs:normalizedString"/>
  <xs:attribute name="documentId" type="xs:normalizedString"/>
  <xs:attribute name="versionId" type="xs:normalizedString"/>
  <xs:attribute name="revisionId" type="xs:normalizedString"/>
  <xs:attribute name="resourceProvenanceId" type="xs:normalizedString"/>
</xs:attributeGroup>

<xs:complexType name="shared_type">
  <xs:sequence>
    <xs:element name="normativeSource" type="normativeSourceShared_type"
minOccurs="0"/>
    <xs:element name="target" type="targetShared_type" minOccurs="0"/>
    <xs:element name="prerequisite" type="prerequisiteShared_type"
minOccurs="0"/>
    <xs:element name="predicate" type="predicateShared_type" minOccurs="0"/>
    <xs:element name="prescription" type="prescriptionShared_type"
minOccurs="0"/>
    <xs:element name="description" type="descriptionShared_type"
minOccurs="0"/>
    <xs:element name="tag" type="tagShared_type" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

```

```

    <xs:element name="var" type="varShared_type" minOccurs="0"
maxOccurs="unbounded"/>
    <xs:element name="report" type="reportShared_type" minOccurs="0"
maxOccurs="unbounded"/>
    <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="sourceDocument_type" mixed="true">
  <xs:sequence>
    <xs:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="revision" type="xs:normalizedString"/>
  <xs:attribute name="version" type="xs:normalizedString"/>
  <xs:anyAttribute namespace="##any" processContents="skip"/>
</xs:complexType>

<xs:complexType name="tag_type">
  <xs:simpleContent>
    <xs:extension base="xs:normalizedString">
      <xs:attribute name="name" type="xs:normalizedString"/>
      <xs:attribute name="lg" type="xs:normalizedString"/>
      <xs:anyAttribute namespace="##any" processContents="skip"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="tagConflictBaseCode_type">
  <xs:restriction base="xs:normalizedString">
    <xs:enumeration value="overriding"/>
    <xs:enumeration value="overridden"/>
    <xs:enumeration value="conjunction"/>
    <xs:enumeration value="disjunction"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="tagConflictCode_type">
  <xs:union memberTypes="tagConflictBaseCode_type codeExtension_type"/>
</xs:simpleType>

<xs:complexType name="tagShared_type">
  <xs:simpleContent>
    <xs:extension base="tag_type">
      <xs:attribute name="conflict" type="tagConflictCode_type"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="target_type">
  <xs:simpleContent>
    <xs:extension base="xs:normalizedString">
      <xs:attribute name="type" type="xs:normalizedString"/>
      <xs:attribute name="lg" type="xs:normalizedString"/>
      <xs:attribute name="idscheme" type="xs:normalizedString"/>
      <xs:anyAttribute namespace="##any" processContents="skip"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

```

```

<xs:simpleType name="targetConflictBaseCode_type">
  <xs:restriction base="xs:normalizedString">
    <xs:enumeration value="overriding"/>
    <xs:enumeration value="overridden"/>
    <xs:enumeration value="conjunction"/>
    <xs:enumeration value="disjunction"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="targetConflictCode_type">
  <xs:union memberTypes="targetConflictBaseCode_type codeExtension_type"/>
</xs:simpleType>

<xs:complexType name="targetShared_type" mixed="true">
  <xs:simpleContent>
    <xs:extension base="target_type">
      <xs:attribute name="conflict" type="targetConflictCode_type"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="testAssertion_type">
  <xs:sequence>
    <xs:element name="description" type="description_type" minOccurs="0"/>
    <xs:element name="normativeSource" type="normativeSource_type"
minOccurs="0"/>
    <xs:element name="var" type="var_type" minOccurs="0"
maxOccurs="unbounded"/>
    <xs:element name="target" type="target_type" minOccurs="0"/>
    <xs:element name="prerequisite" type="prerequisite_type" minOccurs="0"/>
    <xs:element name="predicate" type="predicate_type" minOccurs="0"/>
    <xs:element name="prescription" type="prescription_type" minOccurs="0"/>
    <xs:element name="report" type="report_type" minOccurs="0"
maxOccurs="unbounded"/>
    <xs:element name="tag" type="tag_type" minOccurs="0"
maxOccurs="unbounded"/>
    <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:normalizedString"/>
  <xs:attribute name="lg" type="xs:normalizedString"/>
  <xs:attribute name="schemaVersionId" type="xs:normalizedString"/>
  <xs:anyAttribute namespace="##any" processContents="skip"/>
</xs:complexType>

<xs:complexType name="testAssertionDocumentHeader_type">
  <xs:sequence>
    <xs:element name="common" type="common_type"/>
    <xs:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="testAssertionRef_type">
  <xs:sequence>
    <xs:choice>
      <xs:element name="testAssertionResource" type="testAssertionResource_type"
minOccurs="0" maxOccurs="unbounded"/>
      <xs:element maxOccurs="unbounded" name="testAssertionSetId"

```

```

        type="xs:normalizedString" minOccurs="0"/>
<xs:element minOccurs="0" name="testAssertionIdList">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" name="testAssertionId"
        type="xs:normalizedString" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:choice>
  <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
</xs:sequence>
  <xs:attribute name="lg" type="xs:normalizedString"/>
  <xs:attribute name="name" type="xs:normalizedString"/>
  <xs:anyAttribute namespace="##any" processContents="skip"/>
</xs:complexType>

<xs:complexType name="testAssertionResource_type">
  <xs:attribute name="lg" type="xs:normalizedString"/>
  <xs:attributeGroup ref="resource_attributeGroup"/>
  <xs:anyAttribute namespace="##any" processContents="skip"/>
</xs:complexType>

<xs:complexType name="testAssertionSelection_type">
  <xs:sequence>
    <xs:choice>
      <xs:element maxOccurs="unbounded" name="testAssertionSetId"
        type="xs:normalizedString" minOccurs="0"/>
      <xs:element minOccurs="0" name="testAssertionIdList">
        <xs:complexType>
          <xs:sequence>
            <xs:element maxOccurs="unbounded" name="testAssertionId"
              type="xs:normalizedString" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
    <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:normalizedString"/>
  <xs:attribute name="lg" type="xs:normalizedString"/>
  <xs:attribute name="expr" type="xs:normalizedString"/>
  <xs:anyAttribute namespace="##any" processContents="skip"/>
</xs:complexType>

<xs:complexType name="testAssertionSet_type">
  <xs:sequence>
    <xs:element ref="testAssertionDocumentHeader" minOccurs="0"/>
    <xs:element name="shared" type="shared_type" minOccurs="0"/>
    <xs:element name="testAssertionRef" type="testAssertionRef_type"
minOccurs="0"
  maxOccurs="unbounded"/>
    <xs:element ref="testAssertionSet" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="testAssertionSelection" minOccurs="0"
      type="testAssertionSelection_type" maxOccurs="unbounded"/>
    <xs:element minOccurs="0" name="testAssertionList">
      <xs:complexType>
        <xs:sequence>

```

```

        <xs:element ref="testAssertion" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="id" type="xs:normalizedString"/>
<xs:attribute name="lg" type="xs:normalizedString"/>
<xs:attribute name="schemaVersionId" type="xs:normalizedString"/>
<xs:attribute name="time" type="xs:time"/>
<xs:attribute name="date" type="xs:date"/>
<xs:anyAttribute namespace="##any" processContents="skip"/>
</xs:complexType>

<xs:complexType name="textSourceItem_type">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute name="extension" type="xs:boolean"/>
            <xs:attribute name="name" type="xs:normalizedString"/>
            <xs:attribute name="lg" type="xs:normalizedString"/>
            <xs:anyAttribute namespace="##any" processContents="skip"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="var_type">
    <xs:simpleContent>
        <xs:extension base="xs:normalizedString">
            <xs:attribute name="name" type="xs:normalizedString"/>
            <xs:attribute name="lg" type="xs:normalizedString"/>
            <xs:anyAttribute namespace="##any" processContents="skip"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="varConflictBaseCode_type">
    <xs:restriction base="xs:normalizedString">
        <xs:enumeration value="overriding"/>
        <xs:enumeration value="overridden"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="varConflictCode_type">
    <xs:union memberTypes="varConflictBaseCode_type codeExtension_type"/>
</xs:simpleType>

<xs:complexType name="varShared_type">
    <xs:simpleContent>
        <xs:extension base="var_type">
            <xs:attribute name="conflict" type="varConflictCode_type"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="document_type">
    <xs:simpleContent>
        <xs:extension base="xs:normalizedString">
            <xs:attribute name="label"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

```

```
</xs:complexType>
<xs:complexType name="authors_type" mixed="true">
  <xs:sequence>
    <xs:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##any" processContents="skip"/>
</xs:complexType>
<xs:complexType name="location_type" mixed="true">
  <xs:sequence>
    <xs:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##any" processContents="skip"/>
</xs:complexType>
</xs:schema>
```

4 Conformance

A Conforming Test Assertion is a Test Assertion Markup Language testAssertion element that:

- is valid according to the XML Schema (Section 3)
- satisfies all normative [mandatory](#) provisions ("[must](#)", "[must not](#)", "[shall](#)", "[shall not](#)" keywords) in Sections 2.3 Test Assertion and 2.5 Reserved Tag Names.
- uses the mark-up in compliance with the general semantics of a test assertion and its parts as described in the Test Assertions Model [\[TAM\]](#) specification.

A Conforming Test Assertion Set is a Test Assertion Markup Language testAssertionSet element that:

- is valid according to the XML Schema (Section 3)
- satisfies all normative [mandatory](#) provisions ("[must](#)", "[must not](#)", "[shall](#)", "[shall not](#)" keywords) in Sections 2.3 Test Assertion, 2.4 Test Assertion Set, and 2.5 Reserved Tag Names.
- uses the mark-up in compliance with the general semantics of a test assertion set as described in the Test Assertions Model [\[TAM\]](#) specification.

Appendix A. Acknowledgments

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

Participants:

- David Pawson, Royal National Institute for the Blind
- Dennis Hamilton, Individual
- 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, Korea TAG forum

Appendix B. Revision History

Rev	Date	By Whom	What
CD 1	02/10/10	Stephen Green	CD 1 draft for PR #1
CD 2	08/10/10	Jacques Durand	CD 2 draft for PR #2