Test Assertions Part 1 - Test Assertions Model Version 1.0

Committee Draft 01

13 February 2010

Specification URIs:

This Version:

http://docs.oasis-open.org/tag/model/v1.0/cd01/testassertionsmodel-1.0-cd-01.html

http://docs.oasis-open.org/tag/model/v1.0/cd01/testassertionsmodel-1.0-cd-01.odt

http://docs.oasis-open.org/tag/model/v1.0/cd01/testassertionsmodel-1.0-cd-01.pdf (Authoritative)

Previous Version:

[N/A]

Latest Version:

http://docs.oasis-open.org/tag/model/v1.0/testassertionsmodel-1.0.html

http://docs.oasis-open.org/tag/model/v1.0/testassertionsmodel-1.0.odt

http://docs.oasis-open.org/tag/model/v1.0/testassertionsmodel-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 Part 2 - Test Assertion Markup Language Version 1.0
- OASIS TAG TC - Test Assertions Guidelines

Declared XML Namespace(s):

N/A

Abstract:

This document specifies mandatory and optional components of a test assertion model.

Status:

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

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

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

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" and “OASIS Test Assertions Model” 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

[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.

Data Model Formal Definition Terminology

The means of formally defining the model in this specification involves the use of terms “class”, “attribute”, “datatype” and “association”. These are terms familiar in an object oriented paradigm but **shall not** be strictly interpreted as object oriented terms. The terms are used as a means of formally defining the data structures in the model and do not specify or imply how that data is to be accessed or used. The use of the object oriented terminology **shall not** be taken to mean that the implementation is to be object oriented.

Class

The term “class” is used when the structure so modeled is a complex grouping of more than one entity (either “attributes” or “associations” or both).

Datatype

The term “datatype” is primarily used of a simple, primitive type such as a string or integer. An implementation **may** implement a datatype with another datatype such as a more restricted datatype based on the datatype specified in the model. (For example an entity specified with datatype “string” may be implemented as a URI.)

Attribute

The term “attribute” is used to specify an entity that is an instance of a primitive or simple datatype such as a string or an integer.

Association

The term “association” is used of an entity which is an instance of a class (i.e. its structure is defined by a class) and which appears as an element inside another class.

Domain terminology

Conformance

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

Conformance Clause

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

Implementation

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

Normative Statement, Normative Requirement

A statement made in the body of a specification that defines prescriptive requirements on a conformance target.

Test Case

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

Test Metadata

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

1.2 Normative References

[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.

1.3 Non-Normative References

[CONF1] OASIS Committee Specification 01, "Conformance Requirements for Specifications v1.0", 15 March 2002. http://www.oasis-open.org/committees/download.php/305/conformance_requirements-v1.pdf

[CONF2] L. Rosenthal, M. Skall, L. Carnahan, **"**Conformance testing and Certification Framework ", June 2001(OASIS, white paper)

http://www.oasis-open.org/committees/download.php/309/testing_and_certification_framework.pdf

[VAR] Variability in Specifications, WG note (W3C, 2005)

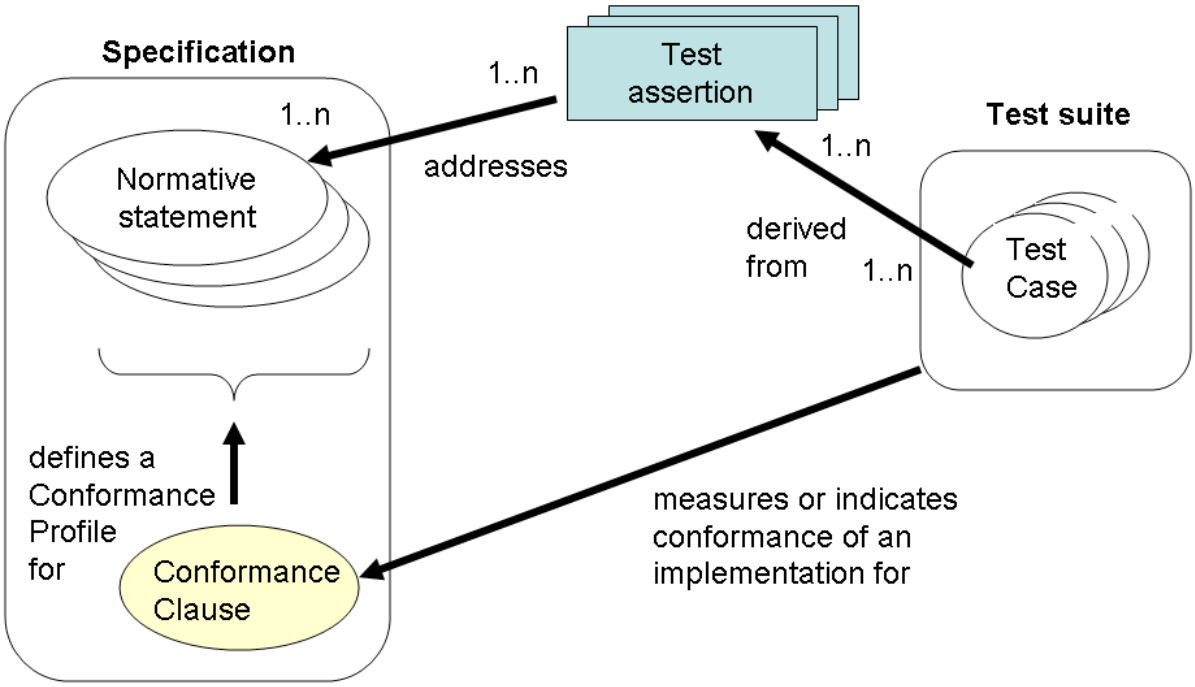
see http://www.w3.org/TR/2005/NOTE-spec-variability-20050831/

2 Definitions and Rationale

2.1 Test Assertion

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

How Test Assertions relate to Specifications and Testing



A Test Assertion should not be confused with a Conformance Clause, nor with a Test Case. The specification will often have one or more conformance clauses [[CONF1]][[CONF2]] which define (one or more) conformance profiles or levels [[VAR]] . A set of test assertions may be associated with a conformance clause in order to define more precisely what conformance entails. Test assertions lie between the specification and any suite of tests to be conducted to determine conformance. Such a test suite is typically comprised of a set of test cases. These test cases may be derived from test assertions which address the normative statements of the specification.

2.2 Test Assertion Set and Test Assertion Document

A container **may** be produced for a complete set of test assertions; often those related to all or part of a specification or conformance profile. In some cases the container is the specification itself with test

assertions included within it. Test assertions can be added to the document, removed or changed using a change and version management procedure.

2.3 Benefits of Test Assertions

Improving the Specification

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

Facilitating Testing

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

3 Test Assertion

3.1 Overview of a Test Assertion Model

Core Test Assertion Parts

A test assertion **shall** include, implicitly or explicitly:

Identifier

This unique identifier facilitates tools development and the mapping of assertions to specification statements. It is recommended that the identifier be made universally unique.[1]

Normative Source

These refer to the precise specification requirements or normative statements that the test assertion addresses.

Target

A test assertion target is the implementation or part of an implementation that is the object of a test assertion or test case. It categorizes an implementation or a part of an implementation of the referred specification.

Predicate

A predicate asserts, in the form of an expression, the feature (a behavior or a property) described in the referred specification statement(s). If the predicate is an expression which evaluates to “true” over the test assertion target, this means that the target exhibits this feature. “False” means the target does not exhibit this feature.

Optional Test Assertion Parts

In addition, a test assertion **may** optionally include:

Prescription Level

The prescription level is a keyword that indicates how imperative it is that the Normative Statement referred to in the Normative Source, be met. The test assertion defines a normative statement which may be **mandatory** (MUST / REQUIRED / SHALL), **permitted** (MAY / CAN) or **preferred** (SHOULD / RECOMMENDED). This property can be termed the test assertion’s prescription level. NOTE: in the case of the normative source including keywords 'MUST NOT' the prescription level '**mandatory**' is used and the 'NOT' included in the predicate. There are differences between various conventions of normative language [ISO/IEC Directives] [RFC 2119]and the above terms **may** be extended with more specialized terms for a particular convention and its distinct shades of meaning.

Prerequisite

A test assertion Prerequisite is a logical expression (similar to a Predicate) which further qualifies the Target for undergoing the core test (expressed by the Predicate) that addresses the Normative

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

Statement. It may include references to the outcome of other test assertions. If the Prerequisite evaluates to "false" then the Target instance is not qualified for evaluation by the Predicate.

Tag

Test assertions may be assigned 'tags' or 'keywords', which may in turn be given values. These tags provide an opportunity to categorize the test assertions. They enable the grouping of the test assertions, for example based on the type of test they assume or based on their target properties.

Variable

Test assertions **may** also include variables for convenience in storing values for reuse and shared use. Another use of a variable is as parameter or attribute employed by the writer of a test assertion to refer to a value that is not known at the time the test assertion is written, but which will be determined at some later stage, possibly as late as the middle of running a set of tests.

Implicit Test Assertion Parts

In an actual test assertion definition, the previously mentioned properties are often explicitly represented as elements of the test assertion.

A concrete representation of a test assertion **may** omit elements provided they are implicit. A common case of implicit test assertion components is the implicit target: When several test assertions relate to the same target, the latter may be described just once as part of the context where the test assertions are defined, so that it does not need to be repeated. This calls for further structural components than those described so far. The more complex structure **may** include a test assertion set whose model caters for sharing of test assertion parts among a group of test assertions.

3.2 Test Assertion Model

Table 1. Mapping Section 3.1 test assertion overview to the formal Test Assertions Model

Test Assertion Structures	Corresponding Entities in Test Assertions Model
Core Parts	
Test Assertion	Class: testAssertion
Identifier	attribute: testAssertion.id ('id' attribute of testAssertion class)
Normative Source	Class: normativeSource
Target	Class: target
Predicate	Class: predicate
Optional Parts	
Prescription Level	attribute: prescription.level ('level' attribute of prescription class)
Prerequisite	Class: prerequisite
Tag	Class: tag
Variable	Class: var

Convention Used for Formally Defining the Model

The means of formally defining the model in this specification involves the use of terms “class”, “attribute”, “datatype” and “association”. These are terms familiar in an object oriented paradigm but **shall not** be strictly interpreted as object oriented terms. The terms are used as a means of formally defining the data structures in the model and do not specify or imply how that data is to be accessed or used. The use of the object oriented terminology **shall not** be taken to mean that the implementation is to be object oriented. See section 1.2 for meanings of these terms.

Example Formal Definition:

```
example {

  content : string (1..1)
  id : string (1..1)
  Child : child (1..*)
  Sibling : sibling (0..*)

}
```

With the exception of the example above, all of the textual representations in this specification **shall** be taken as normative and authoritative. However, some classes in this specification **may** be extended either by adding further attributes or by adding further associations or both. This is indicated in the prose immediately following the representation of the class.

The class name, here called '`example`', is shown before the opening curly bracket. The attributes combine the name of the attribute in lower camel case separated by a colon from the name of the datatype on which the type of the attribute is based. The associations combine the name of the association in upper camel case separated by a colon from the name of the class which is associated and which represents the type of the association. The cardinalities are specified using the notation “(x..y)” where “x” represents the lower bound and “y” the upper bound of the cardinality. The symbol “*” represents a limitless upper bound. There are the following cardinalities in the model:

(0..1) specifies an optional, singular entity (lower bound 0, upper bound 1)

(0..*) specifies an optional, multiple entity (lower bound 0, upper bound unlimited)

(1..1) specifies a mandatory, singular entity (lower bound 1, upper bound 1)

(1..*) specifies a mandatory, multiple entity (lower bound 1, upper unlimited)

(x..y) specifies an entity lower bound x, upper y where x and y are positive integers, for example (1..2)

In the example representation above there is a class called “example” (not a real class, just an example to illustrate the representation convention used in this specification). The class has a mandatory attribute, shown with “(1..1)” to signify that it is mandatory, called “id” whose content is type “string”. The class called “example” has another attribute named “content” which is shown to be optional by the notation (0..1). The “example” class has associations to other classes called “child” and “sibling”. These are similar to attributes whose types are complex and represented in this model as classes. The (0..*) notation signifies that the entity named “sibling” has multiple cardinality and is optional. The (1..*) after the association called “child” signifies that this association is mandatory and multiple.

Any graphic images such as class diagrams included in this specification are non-normative. It is the text which shall be taken as normative. Any diagrams are to be interpreted loosely as illustrative material and in the case of any discrepancy with the text it is the text which is to be taken as authoritative.

testAssertion

Formal Definition of 'testAssertion':

```
testAssertion {

  id : string (1..1)
  language : string (0..1)
  NormativeSource : normativeSource (1..1)
  Target : target (1..1)
  Prerequisite : prerequisite (0..1)
  Predicate : predicate (1..1)
  Prescription : prescription (0..1)
  Description : description (0..1)
  Tag : tag (0..*)
  Variable : variable (0..*)

}
```

Other attributes and associations **may** be added to the `testAssertion` class.

An implementation instance **may** have the `testAssertion` as the top level class.

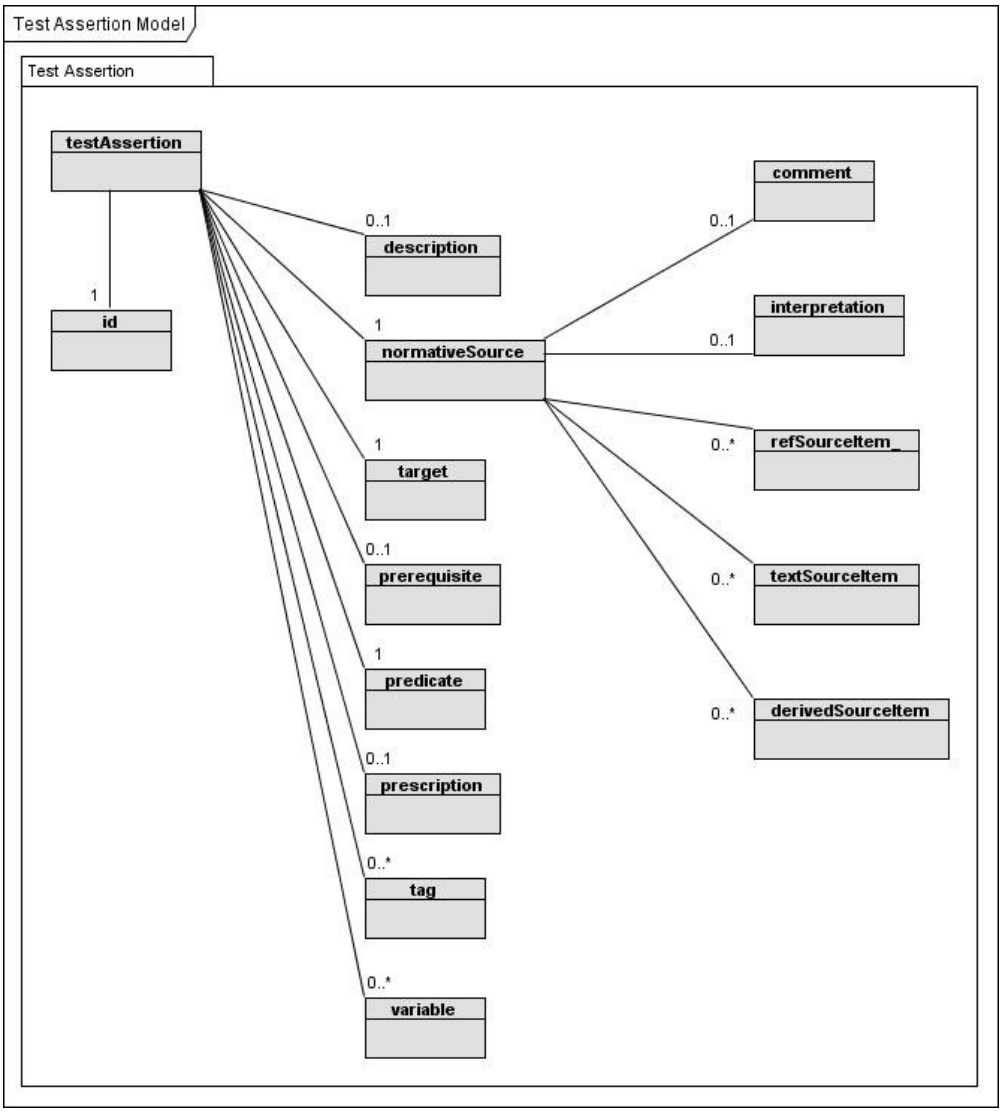
Each test assertion has an identifier and for this reason the `testAssertion` class has an '`id`' attribute. A test assertion **shall** have a test assertion identifier unless the identifier is implicit. So the '`id`' attribute **may** be implicit (by some special provision in a particular profile or implementation, perhaps using an expression to derive the identifiers, for example). For this reason the attribute is optional. 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` class.

The test assertion and most other classes in the Test Assertions Model have an optional attribute named 'language' which is used to specify the language used in the test assertion. The string datatype for this attribute **may** be further constrained using a language codelist.

Each test assertion declares a normative source, a target and a predicate and for this reason the `testAssertion` class has associations with the classes named `normativeSource`, `target` and `predicate`. A test assertion **shall** have a normative source, a target and a predicate unless either or all of these are implicit. The `normativeSource`, `target` and `predicate` associations **may** be implicit and also **may** be declared in a test assertion set (specified later). For this reason these associations are optional in the Test Assertion Model `testAssertion` class. Each conforming implementation of the Test Assertion Model **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 define (a) a prerequisite element, the content of which is a logical expression which may compose multiple simpler prerequisite conditions, (b) a prescription level and (c) any number of tags. For this the `testAssertion` class has optional associations to classes named `prerequisite`, `prescription` and `tag`. The `tag` association has optional, multiple occurrence. The model provides a 'variable' component as an association to a class which it calls `variable`.

In a concrete test assertion representation, mandatory parts could be absent provided that they are implicitly defined somewhere else (i.e. that their actual representation can be inferred, either from a container structure like a “test assertion set” or from other rules).



Test Assertion (Non-Normative UML-Style Class Diagram)

normativeSource

Formal Definition of 'normativeSource':

```
normativeSource {

  content : string (0..1)
  Comment : comment (0..1)
  Interpretation : interpretation (0..1)
  RefSourceItem : refSourceItem (0..*)
  TextSourceItem : textSourceItem (0..*)
  DerivedSourceItem : derivedSourceItem (0..*)

}
```

Other attributes **may** be added to the `normativeSource` class.

The normative source of a test assertion may be provided as a reference using the `refSourceItem` class.

Formal Definition of 'refSourceItem':

```
refSourceItem {

  content : string (0..1)
  name : string (0..1)
  language : string (0..1)
  uri : string (0..1)
  documentId : string (0..1)
  versionId : string (0..1)
  revisionId : string (0..1)
  dateString : string (0..1)
  resourceProvenanceId : string (0..1)
  resourceType : string (0..1)
  resourceTypeVersionId : string (0..1)
  resourceTypeSchemaId : string (0..1)
  resourceTypeSchemaVersionId : string (0..1)
  resourceTypeProvenanceId : string (0..1)

}
```

Other attributes **may** be added to the `refSourceItem` class.

Here there is metadata which may be used to specify the identification of a resource containing the normative source items. 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 normative source items available. 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.

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 association with a class named `textSourceItem` which allows a direct, verbatim quote of the specification text.

Formal Definition of 'textSourceItem':

```
textSourceItem {

  content : string (0..1)
  name : string (0..1)
  language : string (0..1)

}
```

Other attributes **may** be added to the `textSourceItem` class.

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 association to a class named `derivedSourceItem`. This is particularly useful when the source consists of tables, diagrams, graphs or text spread over several parts of the specification.

Formal Definition of 'derivedSourceItem':

```
derivedSourceItem {

  content : string (0..1)
  name : string (0..1)
```

```
  language : string (0..1)
  uri : string (0..1)
  documentId : string (0..1)
  versionId : string (0..1)
  revisionId : string (0..1)
  dateString : string (0..1)
  resourceProvenanceId : string (0..1)
  resourceType : string (0..1)
  resourceTypeVersionId : string (0..1)
  resourceTypeSchemaId : string (0..1)
  resourceTypeSchemaVersionId : string (0..1)
  resourceTypeProvenanceId : string (0..1)

}
```

Other attributes **may** be added to the `derivedSourceItem` class.

Here there is metadata which may be used to specify the identification of a resource containing the normative source items. 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 normative source items available. 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.

Formal Definition of 'comment':

```
comment {

  content : string (0..1)
  language : string (0..1)

}
```

Other attributes **may** be added to the `comment` class.

The `comment` class 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.

Formal Definition of 'interpretation':

```
interpretation {

  language : string (0..1)
  content : string (0..1)

}
```

Other attributes **may** be added to the `interpretation` class.

The `interpretation` class 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. It provides further information about how the predicate (or prerequisite) relates to the normative source.

target

Formal Definition of 'target':

```
target {

  content : string (0..1)
  type : string (0..1)
  schemeRef : string (0..1)
  language : string (0..1)

}
```

Other attributes **may** be added to the `target` class.

A target can either be a specific item or a category of items. The `target` class has a '`type`' attribute which **should** be used to specify the target category, and this **may** be implemented using a controlled vocabulary, ontology or other classification or taxonomy system. 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`' attribute or, for a set of test assertions, a shared target '`schemeRef`' attribute **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.

The target content is a string. This **may** be an expression in a specialized formal expression language which **may** be specified using the '`language`' attribute 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). The predicate **may** express a condition over more than one object. These objects are either parts of an implementation or external resources. A unique target object is still required by this model. In such a case, either the target is defined as the combination of objects that are expected to satisfy the predicate, or one of these objects may be selected as the target while the other objects are just considered as accessory to the test. Such objects **may** be referenced in the predicate or prerequisite using variables.

prerequisite

Formal Definition of 'prerequisite':

```
prerequisite {

  content : string (1..1)
  language : string (0..1)

}
```

Other attributes **may** be added to the `prerequisite` class.

The prerequisite **may** be expressed using a specialized formal expression language which **may** be specified using the '`language`' attribute or using a complete conformance profile for that particular use of an implementation of the model. The semantics of the test assertion depends on the value of the prerequisite (see Section 3.3). These semantics require that the prerequisite evaluates to a boolean logical true or false. The `prerequisite` class is an optional association of 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 an implementation of the test assertions model. The content of the `prerequisite` class **shall** contain an expression which evaluates 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 in Section 3.3.

predicate

Formal Definition of 'predicate':

```
predicate {

  content : string (1..1)
  language : string (0..1)

}
```

Other attributes **may** be added to the `predicate` class.

The predicate **may** be expressed using a specialized formal expression language which **may** be specified using the '`language`' attribute or using a complete conformance profile for that particular use of an implementation of the model. The semantics of the test assertion depends on the value of the predicate (see Section 3.3). These semantics require that the predicate evaluates to a boolean logical true or false.

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` class shall be optional as association of a `testAssertion` class unless otherwise made mandatory using a conformance profile of a customization. The content of the `predicate` class **shall** contain an expression which evaluates 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 in Section 3.3.

prescription

Formal Definition of 'prescription':

```
prescription {

  content : string (0..1)
  level : string (0..1) (allowed values = mandatory|preferred|permitted)

}
```

Other attributes **may** be added to the `prerequisite` class.

The allowable values for the attribute '`level`' of the class prescription **may** be extended beyond the built-in 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 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 content (string) **may** be used to express further information regarding the prescription level using prose or as a logical expression.

description

Formal Definition of 'description':

```
description {

  content : string (0..1)
  language : string (0..1)

}
```

Other attributes **may** be added to the `description` class.

The `description` class 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.

tag

Formal Definition of 'tag':

```
tag {

  content : string (0..1)
  name : string (0..1)
  language : string (0..1)

}
```

Other attributes **may** be added to the `tag` class.

The use of 'tags' assigned to a test assertion is a means to assign metadata and data of a similar nature to the test assertion. 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` class **may** be used to attach such data to a test assertion or test assertion set.

Reserved Tag Names

`DefinesNormativeProperty` and `NormativeProperty`

A test assertion **may** be tagged to show that it is a property test assertion using two reserved word tag names `DefinesNormativeProperty` and `NormativeProperty` .

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, you can easily generate the set of test assertions that apply to a specific version of the specification.

variable

Formal Definition of 'variable':

```
variable {

  content : string (0..1)
  name : string (0..1)
  language : string (0..1)

}
```

Other attributes **may** be added to the `variable` class.

The use of variables allows several parts of a test assertion or indeed several test assertions within a set to share a value between them. Such variables may have their values supplied as parameters at a stage subsequent to the authoring of the test assertions. The variable class **may** be used to implement variables. 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 Section 4 'Test Assertion Set'). The variable value **may** be used within a test assertion part expression. The notation used for these variables in the content of predicates, prerequisite, etc. is left to implementations of this model.

3.3 Semantics

As a test assertion has parts that can be evaluated over a target instance (i.e. the prerequisite, the predicate, and possibly any variables containing expressions), the following semantics **shall** apply to a test assertion:

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.

4 Test Assertion Set

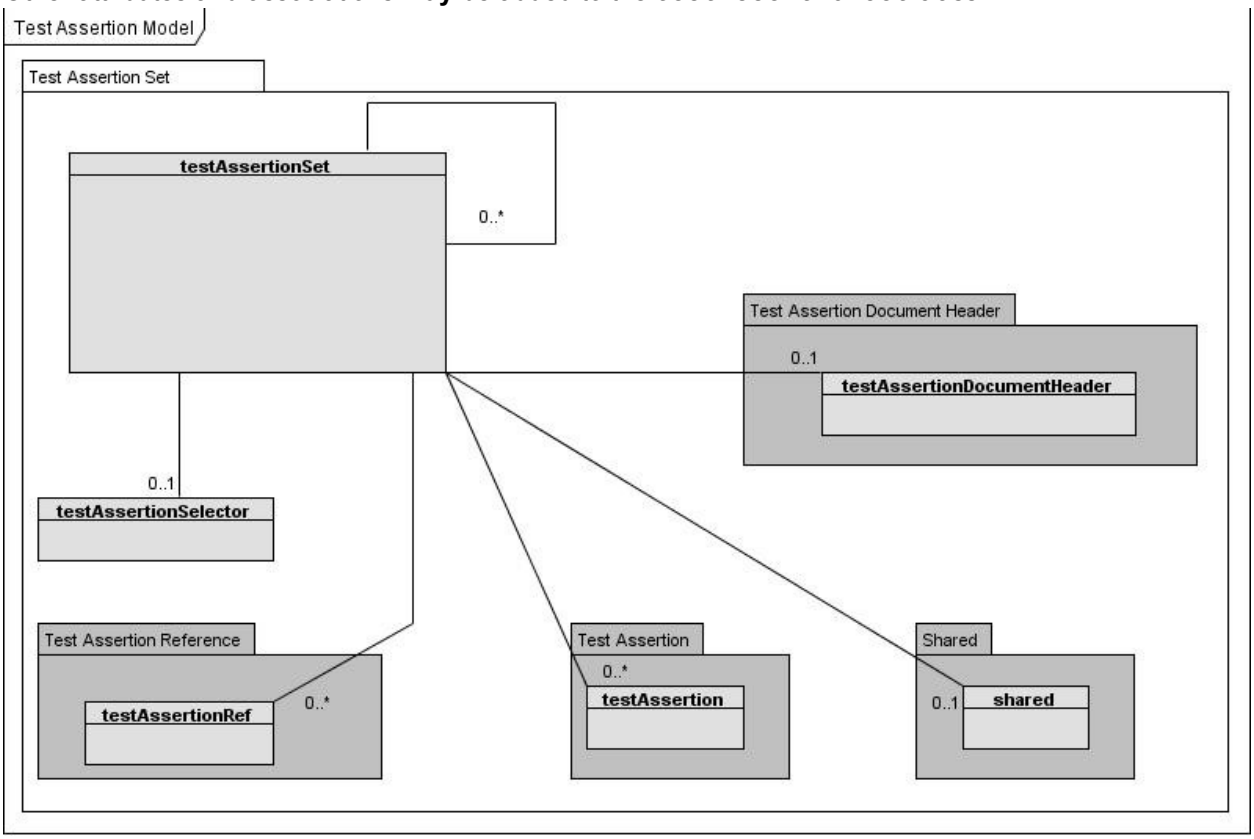
The `testAssertionSet` class **may** be used to group together test assertions either by inclusion of the test assertion within the test assertion set of by references to their Ids.

testAssertionSet

Formal Definition of 'testAssertionSet':

```
testAssertionSet {

  id : string (0..1)
  language : string (0..1)
  date : date (0..1)
  time : time (0..1)
  TestAssertionDocumentHeader : testAssertionDocumentHeader (0..1)
  Shared : shared (0..1)
  TestAssertion : testAssertion (0..*)
  TestAssertionRef : testAssertionRef (0..*)
  TestAssertionSet : testAssertionSet (0..*)
  TestAssertionSelector : testAssertionSelector (0..1)

}
```

Other attributes and associations **may** be added to the `testAssertionSet` class.



Test Assertion Set (Non-Normative UML-Style Class Diagram)

An implementation instance **may** have the `testAssertionSet` as the top level class.

An implementation conformance profile for the model **may** include the use of the feature where a testAssertionSet **may** include other test assertion sets. The default model defined in this specification allows a test assertion set to optionally 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, say, a document. In this case the `testAssertionDocumentHeader` **may** be used once within a document either on its own 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`' class below.)

The `testAssertionSelector` **may** be used when test assertions are contained within a document but outside of the test assertion set. One case where this may be used is where such test assertions are distributed throughout a physical or logical document written in some kind of markup where the `testAssertionSelector` may provide an expression which identifies all of the test assertions within that markup. The `language` attribute may be used to identify the expression language used.

Formal Definition of 'testAssertionSelector':

```
testAssertionSelector {

  content : string (0..1)
  language : string (0..1)

}
```

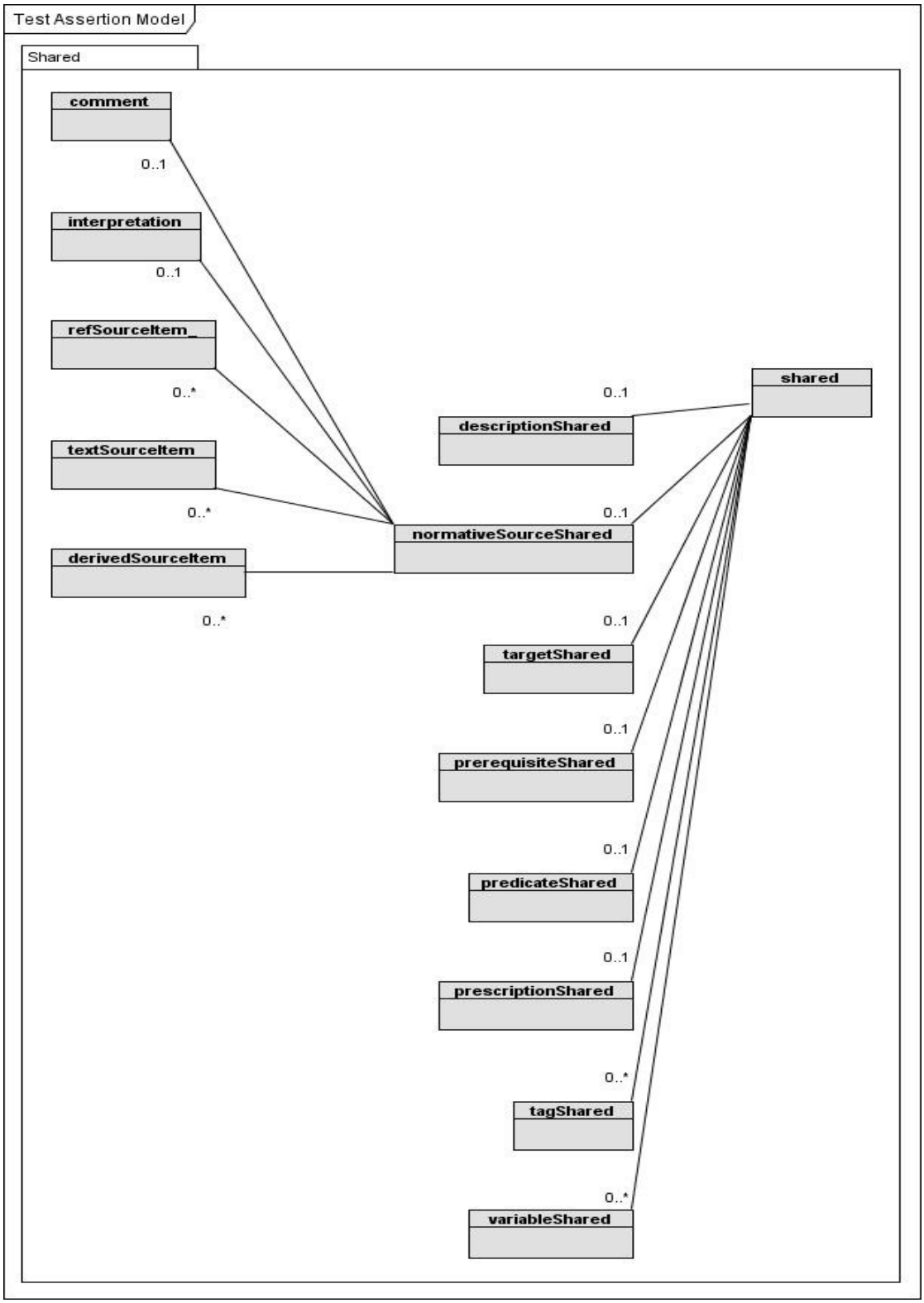
Other attributes **may** be added to the `testAssertionSelector` class.

shared

Formal Definition of 'shared':

```
shared {

  NormativeSource : normativeSourceShared (0..1)
  Target : targetShared (0..1)
  Prerequisite : prerequisiteShared (0..1)
  Predicate : predicateShared (0..1)
  Prescription : prescriptionShared (0..1)
  Description : descriptionShared (0..1)
  Tag : tagShared (0..*)
  Variable : variableShared (0..*)

}
```

Other associations **may** be added to the `shared` class.



Shared (Non-Normative UML-Style Class Diagram)

The class '`shared`', an association of the `testAssertionSet` class **may** be used to provide one or more test assertion parts either as overrides (either overridden by or overriding any corresponding parts

of the same kind of test assertions 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' ,'description', 'tag', and the 'variable' associated classes of this 'shared' element, are extended with a 'conflict' attribute. These extended classes are here given a suffix 'Shared' but other conventions **may** be used to distinguish the extended classes and associations to them.

normativeSourceShared

Formal Definition of 'normativeSourceShared':

```
normativeSourceShared {

  content : string (0..1)
  conflict : string (0..1) (allowed values = conjunction|disjunction|
overriding|overridden)
  Comment : comment (0..1)
  Interpretation : interpretation (0..1)
  RefSourceItem : refSourceItem (0..*)
  TextSourceItem : textSourceItem (0..*)
  DerivedSourceItem : derivedSourceItem (0..*)

}
```

Other attributes **may** be added to the `normativeSourceShared` class.

targetShared

Formal Definition of 'targetShared':

```
targetShared {

  content : string (0..1)
  type : string (0..1)
  schemeRef : string (0..1)
  language : string (0..1)
  conflict : string (0..1) (allowed values = conjunction|disjunction|
overriding|overridden)

}
```

Other attributes **may** be added to the `targetShared` class.

prerequisiteShared

Formal Definition of 'prerequisiteShared':

```
prerequisiteShared {

  content : string (1..1)
  language : string (0..1)
  conflict : string (0..1) (allowed values = conjunction|disjunction)

}
```

Other attributes **may** be added to the `prerequisiteShared` class.

predicateShared

Formal Definition of 'predicateShared':

```
predicateShared {
  content : string (1..1)
  language : string (0..1)
  conflict : string (0..1) (allowed values = conjunction|disjunction|
overriding|overridden)

}
```

Other attributes **may** be added to the `predicateShared` class.

prescriptionShared

Formal Definition of 'prescriptionShared':

```
prescriptionShared {

  content : string (0..1)
  level : string (0..1) (allowed values = mandatory|preferred|permitted)
  conflict : string (0..1) (allowed values = overriding|overridden)

}
```

Other attributes **may** be added to the `prescriptionShared` class.

descriptionShared

Formal Definition of 'descriptionShared':

```
descriptionShared {

  content : string (0..1)
  language : string (0..1)
  conflict : string (0..1) (allowed values = conjunction|disjunction|
overriding|overridden)

}
```

Other attributes **may** be added to the `descriptionShared` class.

tagShared

Formal Definition of 'tagShared':

```
tagShared {

  content : string (0..1)
  name : string (0..1)
  language : string (0..1)
  conflict : string (0..1) (allowed values = conjunction|disjunction|
overriding|overridden)

}
```

Other attributes **may** be added to the `tagShared` class.

variableShared

Formal Definition of 'variableShared':

```
variableShared {

  content : string (0..1)
  name : string (0..1)
  language : string (0..1)
  conflict : string (0..1) (allowed values = conjunction|disjunction|
overriding|overridden)

}
```

Other attributes **may** be added to the `variableShared` class.

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 '`variable`', 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 classes 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.

Test Assertion Reference

A test assertion set may refer to one or more test assertions by their test assertion identifiers rather than include the test assertions literally within the set.

testAssertionRef

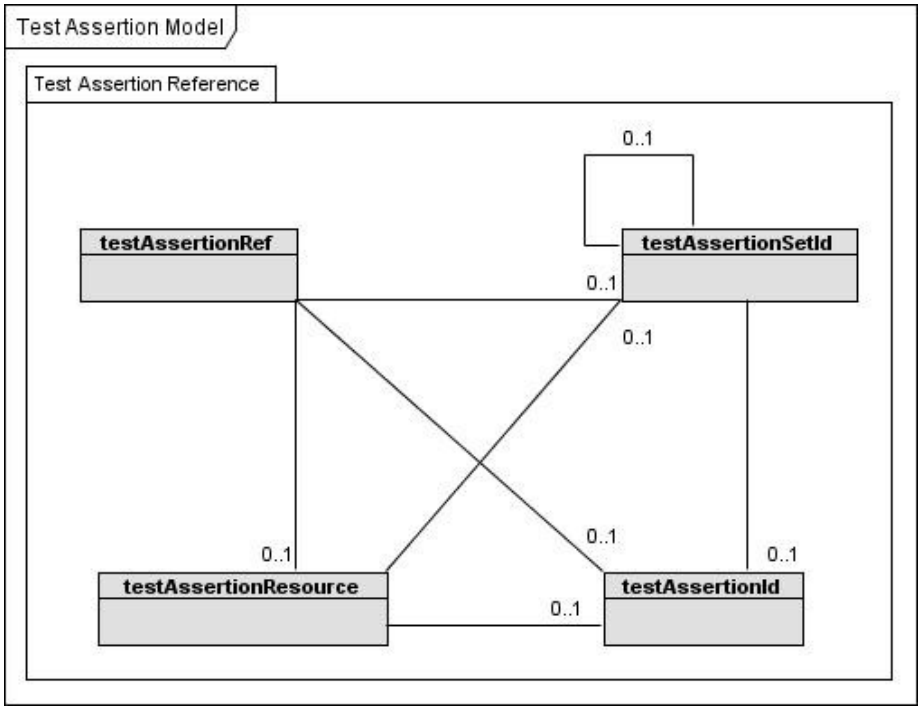
Formal Definition of 'testAssertionRef':

```
testAssertionRef {

  language : string (0..1)
  name : string (0..1)
  TestAssertionReference : testAssertionResource (0..1)
  TestAssertionSetId : testAssertionSetId (0..1)
  TestAssertionId : testAssertionId (0..1)

}
```

Other attributes and associations **may** be added to the `testAssertionRef` class.

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` class to do so. The structure of this class allows for the possibility that test assertions may be contained in another document in another location by inclusion of an association to class `testAssertionResource`. Other associations `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` associations 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.



Test Assertion Reference (Non-Normative UML-Style Class Diagram)

testAssertionResource

A test assertion resource is used when test assertions are contained in another external document.

Formal Definition of 'testAssertionResource':

```
testAssertionResource {

  language : string (0..1)
  uri : string (0..1)
  documentId : string (0..1)
  versionId : string (0..1)
  revisionId : string (0..1)
  dateString : string (0..1)
  resourceProvenanceId : string (0..1)
  resourceType : string (0..1)
  resourceTypeVersionId : string (0..1)
  resourceTypeSchemaId : string (0..1)
  resourceTypeSchemaVersionId : string (0..1)
  resourceTypeProvenanceId : string (0..1)
  TestAssertionId : testAssertionId (0..1)
  TestAssertionSetId : testAssertionSetId (0..1)

}
```

Other attributes **may** be added to the `testAssertionResource` class.

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 includes information about the kind of resource involved and most appropriately its provenance (such as authorship identifiers to certify its authenticity) and version, etc.

testAssertionId

Formal Definition of 'testAssertionId':

```
testAssertionId {

  ref : string (0..1)
  language : string (0..1)

}
```

Other attributes **may** be added to the `testAssertionId` class.

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

Formal Definition of 'testAssertionSetId':

```
testAssertionSetId {

  ref : string (0..1)
  language : string (0..1)
  section : string (0..1)
  TestAssertionId : testAssertionId (0..1)
  TestAssertionSetId : testAssertionSetId (0..1)

}
```

Other attributes **may** be added to the `testAssertionSetId` class.

The `testAssertionSetId` 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 itself. A section within that test assertion set **may** also be specified where appropriate, bearing in mind that the test assertion set (perhaps called by another term) might not be written using an implementation of this model.

testAssertionDocumentHeader

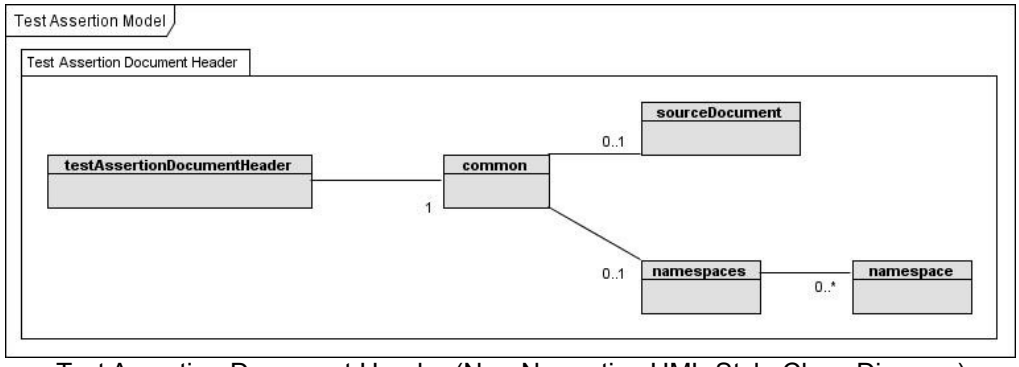
Formal Definition of 'testAssertionDocumentHeader':

```
testAssertionDocumentHeader {

  Common : common (1..1)

}
```

Other associations **may** be added to the `testAssertionDocumentHeader` class.

The `testAssertionDocumentHeader` **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. 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 level class. There **shall** be no more than one `testAssertionDocumentHeader` used in any given document implementing this model.



Test Assertion Document Header (Non-Normative UML-Style Class Diagram)

common

Formal Definition of 'common':

```
common {

  SourceDocument : sourceDocument (0..1)
  Authors : authors (0..1)
  Location : location (0..1)
  Namespaces : namespaces (0..1)

}
```

Other associations **may** be added to the `common` class.

Formal Definition of 'sourceDocument':

```
sourceDocument {

  content : string (0..1)
  revision : string (0..1)
  version : string (0..1)

}
```

Other attributes **may** be added to the `sourceDocument` class.

Here some of the 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.

Formal Definition of 'namespaces':

```
namespaces {
```

```
  content : string (0..1)
  Namespace : namespace (0..*)

}
```

Formal Definition of 'namespace':

```
namespace {

  content : string (0..1)
  Prefix : prefix (0..1)
  Uri : uri (0..1)

}
```

Formal Definition of 'prefix':

```
prefix {

  content : string (0..1)

}
```

Formal Definition of 'uri':

```
uri {

  content : string (0..1)

}
```

The `namespaces` element caters for a special case of usage of the markup. Here the content **should** 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.

5 Conformance

Implementations subject to conformance are representations of the test assertion model described in Section 3 and Section 4.

In order to conform to these guidelines, a test assertion representation:

(1) **shall** contain all test assertion parts defined in Section 3

(2) **may** contain any test assertion constructs defined in Section 4

(3) **shall** use names for these parts that are identical or can be unambiguously mapped to the definitions used in Section 3 and implemented parts of Section 4

(4) **shall** implement the normative statements for the test assertion model and its semantics in this specification.

Mandatory statements are designated by the keyword '**shall**' and '**shall not**' in bold type, as described in Annex H of [ISO/IEC Directives] .

Appendix A. Acknowledgments

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

Participants:

- 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, KIEC