Date: January 2013

Test Information Interchange Format (TestIF) Specification

FTF Beta ~~1~~2

OMG Document Number: dtc/2013-xx-xx
Standard Document URL: http://www.omg.org/spec/TestIF
Machine Consumable Files:
Normative:
c4i/12-09-08 – http://www.omg.org/spec/TestIF/20121101/TestIF_UMLPIM_2012_09_24.xml
c4i/12-09-09 – http://www.omg.org/spec/TestIF/20121101/TestIF.xsd
c4i/12-09-11 – http://www.omg.org/spec/TestIF/20121101/org_omg_testif_attributes.xml
Non-Normative:
c4i/12-09-10 – http://www.omg.org/spec/TestIF/20121101/TestIF_SQL_Create_Scrips.sql
c4i/12-09-12 – http://www.omg.org/spec/TestIF/20121101/TestIF_UML_PIM_2012_09_24.mdzip

---

OMG's Issue Reporting Procedure

Contents

List of Figures

List of Tables

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:
http://www.omg.org/spec

Specifications are organized by the following categories:

Business Modeling Specifications

Middleware Specifications

- **CORBA/IIOP**
- **Data Distribution Services**
- **Specialized CORBA**

IDL/Language Mapping Specifications

Modeling and Metadata Specifications

- **UML, MOF, CWM, XMI**
- **UML Profile**

Modernization Specifications

Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

- **CORBAServices**
- **CORBAFacilities**

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier/Courier New - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

1 Scope

The scope of this response is to propose a standard for Test Information Interchange for Automating Software Test Processes for C4I and software systems. Hardware testing is not directly in the scope of the proposed standard.

Comment [MW1]: Issue 18335

1.1 Purpose

The goal is to achieve a specification that defines the format for the exchange of test information ~~between~~ among tools, applications, and systems that utilize it. The term "test information" is deliberately vague, because it includes the concepts of tests (test cases), test results, test scripts, test procedures, and other items that are normally documented as part of a software test effort.

Comment [MW2]: Issue 18360

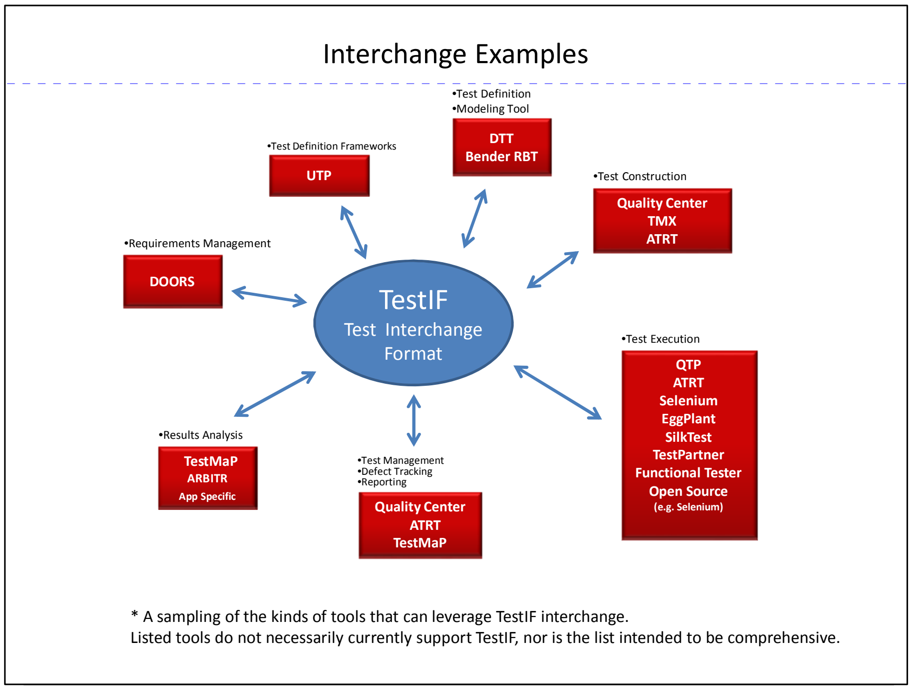
The long term goal is to standardize the exchange of all test related artifacts produced or consumed as part of the testing process, however, the current proposal is primarily focused on artifacts used or produced outside of test execution. The following are specifically in scope:

- The format of information artifacts related to testing to enable data exchange. The list of types of information is provided in the *Issues to be Discussed* section.
- Description of test specification entities including mandatory and user defined attributes
- The logical relationships ~~between~~ among the test information entities.
- Specification of a MOF compliant Platform Independent Model (expressed in UML) to cover test information data exchange.
- A simple XML schema for validation of test data being exchanged.
- Description of the process by which the standard can be extended.

Comment [MW3]: Issue 18360

Comment [MW4]: Issue 18336

Figure 1 is a notional view of the way compliant tools would access and contribute test information in a standard format. The red boxes represent instances of different types of tools that might produce or consume test information. These boxes are not intended to be prescriptive or a comprehensive capture of all of the functionality related to test data information exchange. This is merely here as an illustration of the types of tools and functionality that would utilize and benefit from this standard.



Comment [MW5]: Issue 18337

Figure 1-1: Notional Interchange Example

1.2 Usage

The following paragraphs outline the primary usage of the TestIF Standard.

1.2.1 Basic Test Information Interchange

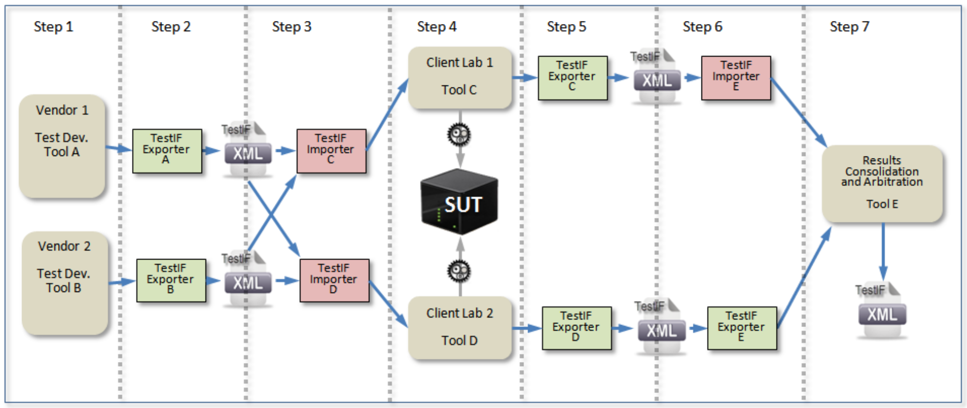


Figure 1-2: Basic Test Information Interchange

Figure 1-2 represents common scenarios on how Test Information is exchanged between several testing organizations using different testing tools. A C4I organization engages two vendors to design tests case for an application “X”. Table 1-1 details this process. The client desires to execute the tests in their own environments. The vendors export test cases to the TestIF compliant XML document using a TestIF compliant component or utility (TestIF Exporter). The test cases then are independently imported by the two different client labs using TestIF compliant component or utility (TestIF Importer). Each lab then adds the necessary tool specific execution code and environment specific attribute values to the TestIF Test Objects imported from the XML document. The tests are then executed against the application under test in their respective environments. The execution results are added to the test information and the updated test information is exported to a TestIF document for consumption by an arbitration process. Arbitration Tool E then imports the documents for consolidation, analysis and reporting. The results are then published to a TestIF document. The TestIF standard allows all of these organizations to exchange test information without regard to the other’s testing technologies.

Table 1-1: Vender Process for Test Information Interchange

Step 1	Vendors develop Tests Cases for application “X”
Step 2	Vendors export Test Cases to TestIF documents
Step 3	Client labs Import the TestIF documents into their respective test environments
Step 4	Labs “instrument” the Test Steps with appropriate executable expressions and execute the tests
Step 5	Labs export Test Information including results
Step 6	Arbiter imports TestIF documents into a consolidated repository
Step 7	Arbiter analyzes tests information and test results to roll up verdicts Arbiter publishes test information and test results in a TestIF compliant document

1.2.2 Usage – Tool Independence

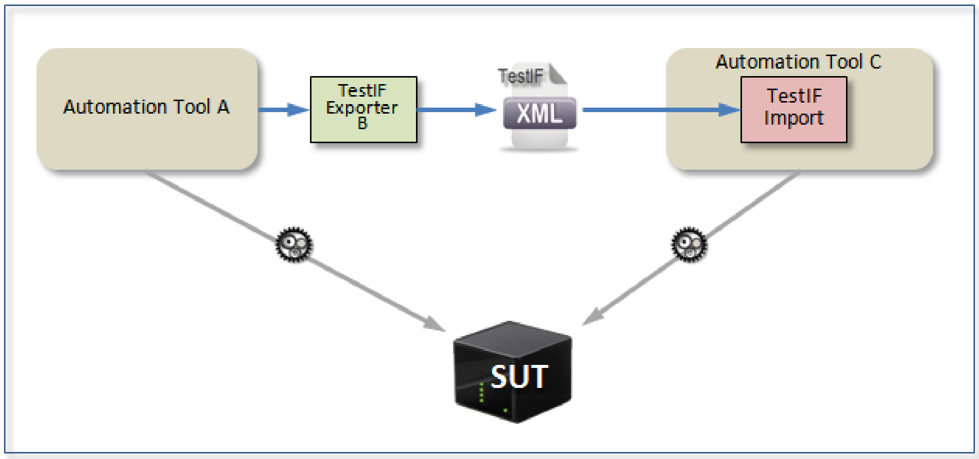


Figure 1-3: Tool Independence

A testing organization (TO) has developed a test library using Automation Tool A and has been executing test case against the System Under Test for several test cycles. Conditions have changed and the organization decides to utilize Test Automation Tool B in place of Automation Tool A. The TO acquires a tool "TestIF Exporter B" that can create a TestIF document from tests stored in Automation Tool A. Automation Tool C has the capability of importing TestIF documents directly. Test cases are imported into Automation Tool C and tests are executed against a SUT. In this exchange the Test Cases are preserved and only the contents of Test Objects that contain executable code are replaced to make the transition.

Comment [MW6]: Issue 18338

Note on TestIF Export and Import Utilities:

The specifications for tools that export or import TestIF compliant documents are outside the standard. Such utilities must simply comply with the defined TestIF document standards. It is anticipated that vendors will create integrated import/export capabilities within existing testing tools as well as new independent tools that act as servers to provide interfaces between TestIF documents and available commercial testing tools.

1.3 Examples

1.3.1 Example 1 – C4I Simple Example

Introduction
The goal of this example is to show how a very simple test is represented in the basic TestIF structures. The TestIF is very flexible in the level of expressivity of the test information. For tests that are fully automated the structures in the TestIF document would likely be highly annotated with machine readable attributes. In this example we will focus only on the underlying basic structures.

Note that the symbols used in the example figures are non-normative. They are intended for illustrative purposed only.

Comment [MW7]: Issue 18330

Example
The testing of C4I systems typically involves verification of interface contracts and verification of the outputs of processing received messages. To keep the example short we will deal with only two fairly simple requirements from a system that is processing an external message about a track's course:

Table 1-2: Sample Requirements

ID	Requirement
R1	Values less than 0 received in the course field shall be set to 0.00.
R2	Values greater than 359.99 received in the course field shall be set to 359.99.

A human readable test procedure derived from the above requirements might take the form shown in Table 1-3.

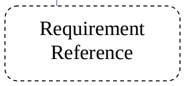
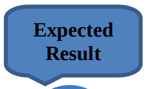
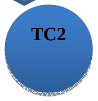
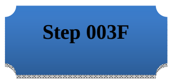
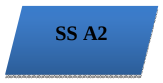
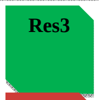
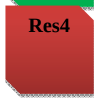
Table 1-3: Notional Human Readable Test Procedure

Test Case	Step	Requirement	Action	Expected Response
TC1	TC1_TS1		Send message with course set to less than zero.	
TC1	TC1_TS2	R1	Verify displayed value.	Displayed course equals 0.00
TC2	TC2_TS1		Send message with course set to greater than 359.99.	
TC2	TC2_TS2	R2	Verify displayed value.	Displayed course equals 359.99

TestIF representation of the Test Procedure

The diagram in Table 1-4 is a legend of the symbols used in the subsequent diagrams. Each item maps directly to a TestIF object.

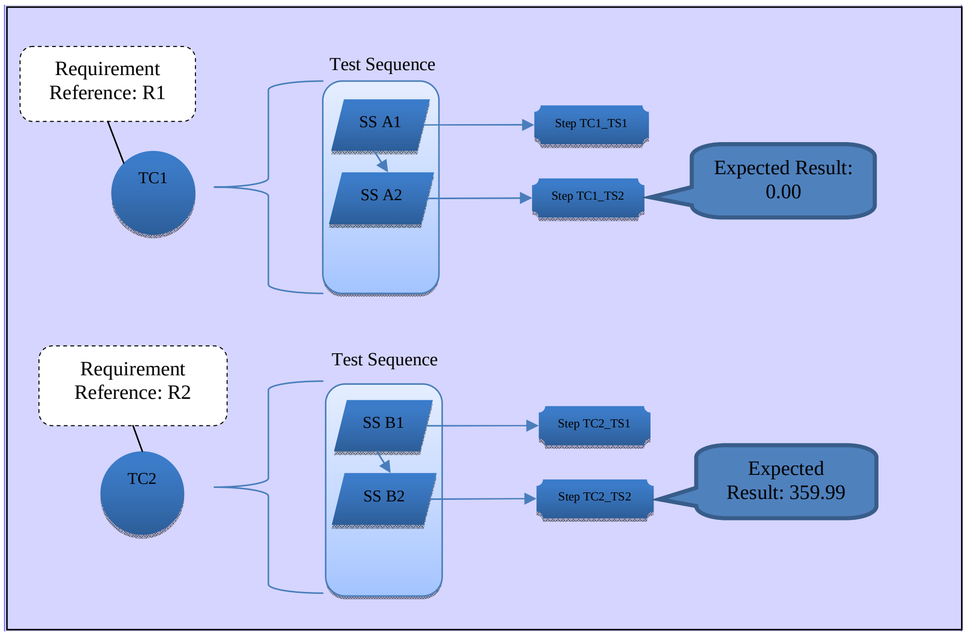
Table 1-4: Legend for Subsequent Diagrams

Symbols	Definition
	Requirement (Externally Defined)
	~~Attribute Value~~The expected result
	Test Case
	Test Step
	Sequenced Step
	Test Result – Pass
	Test Result – Fail

Formatted Table

Comment [MW8]: Issue 18331

The above Test Procedure translated into TestIF objects would take the form shown in Figure 1-4.



Comment [MW9]: Issue 18331

Figure 1-4: Test Procedure Translated into TestIF Objects

- The Test Case objects in this example act as the container for the sequence of execution of their associated steps (Test Sets and Test Steps can also contain an internal sequence of associated items).
- The Test Sequences in this Test Procedure are linear and straightforward, but TestIF's Test Sequence object and referencing approach is able to represent any directed graph. This allows TestIF to handle complex test sequences (eg. loops, parallel execution, etc.).
- Essentially all the objects in TestIF are defined once in the document and then referenced where they are needed. The blue arrows indicate this referencing mechanism. If two different Test Cases needed to reference the same Test Step(s), the Test Steps are defined once in the TestIF document and simply referenced from both locations.
- Requirement Reference Attributes are used in this example to attach the requirement to the Test Case the requirement could have been attached at the Test Step level instead if that is more expressive/correct in another scenario.
- Expected Result Attributes are used to capture the value that must be verified in the associated item in a well-defined machine readable format.

A record of an execution of the Test Cases above represented in TestIF would take the form shown in Figure 1-5.

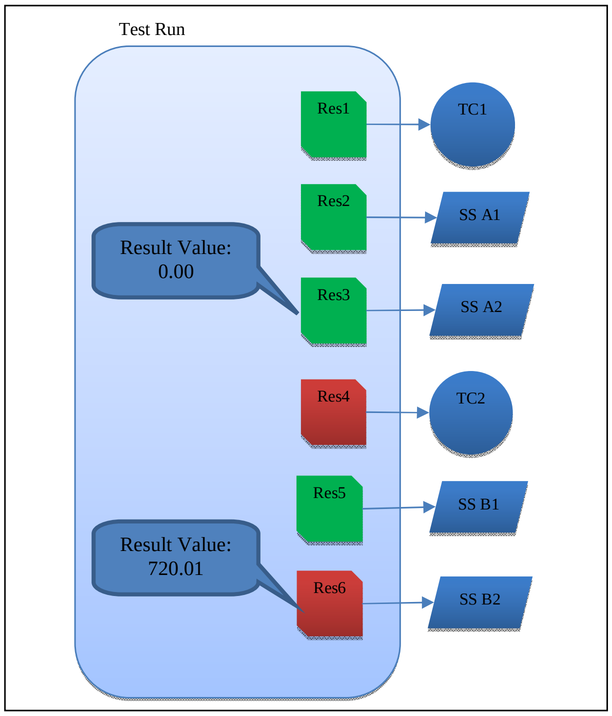


Figure 1-5: The Execution of the Test Cases Represented in TestIF

- In this example we can see that an execution of the two Test Cases above produced a set of Test Results. The first Test Case passed with correct result. The second failed due to ~~a~~ observing an incorrect value.
- The Test Run objects acts as a container for a list of Test Result objects.
- Each Test Result object references the object that it is a result for. Sequence Steps are the most common reference since they indicate exactly where in a specific Test Sequence this result was generated from. In the case where there is a Test Result that is generated external to specific Test Sequence the Test Result references the item for which the Result is associated (Test Set, Test Case, and Test Step). In this example the rollup results for each of the Test Cases was determined by a second tool after the execution of the test and then written into the TestIF document.
- Result Value Attributes are used to capture the results observed during execution.

Comment [MW10]: Issue 18360

Comment [MW11]: Issue 18360

1.3.2 Example 2 – Requirement Traceability and Results Arbitration for Cause-Effect Model-Generated Tests

Introduction

Critical Logic uses a Cause-Effect Modeling tool ~~called DTT~~to generate test cases. Cause-Effect Models represent rules for system behavior. The models automatically generate test cases for complete functional test coverage of the rules.

Comment [MW12]: Issue 18332

The model-generated test cases can be output to reports for manual test execution, imported into test management tools, or imported into automation tools for scripting. These test case definitions have potentially complex structures that allow for optimized test design, very accurate requirement traceability, and fine-tuned test result arbitration.

The goal of this example is to show how TestIF supports the unique capabilities of DTT around requirement traceability and results arbitration, allowing other tools to leverage and build on those capabilities.

Note that the symbols used in the example figures are non-normative. They are intended for illustrative purposed only.

Comment [MW13]: Issue 18330

Example
Automated Testing of C4I and other complex systems requires automation of reporting on Requirement Verification Status (RVS). The reporting requirement common creates two problems:

1. Oversimplification of test cases to be requirement-specific, resulting in incomplete test coverage
2. Inaccurate reporting of requirement status when a failed test case nonetheless successfully verified some requirement within it

Both increase risk and cost. Cause-Effect Models solve these problems by providing complete test coverage, optimizing test counts, and maintaining test-step-level requirement traceability. In so doing, Cause-Effect Models reveal deeper challenges to RVS reporting:

1. It typically takes more than one test to fully exercise a requirement.
2. A typical test exercises more than one requirement.

The present example shows how TestIF supports DTT's solution to these two problems.

Requirements
Our example starts with the requirements for user login to a secure, account-based system.

Table 1-5: Requirements

RqmtID	Requirement
R01	If the user logs in successfully and their account is Open or Pending, display the welcome screen.
R02	If the user logs in successfully and their account is Open, display Message 2.
R03	If the user logs in successfully and their account is Pending, display Message 3.

Notes:
- R01 requires two positive verifications – one for "Open" and one for "Pending".
- R02 is specific to "Open," and may occur regardless of the outcomes related to R01.
- R03 is specific to "Pending," and may occur regardless of the outcomes related to R01.

Cause-Effect Model

This Cause-Effect Model, Figure 1-6, is a complete representation of the requirements in Table 1-5.

A legend is included to the right of the model.

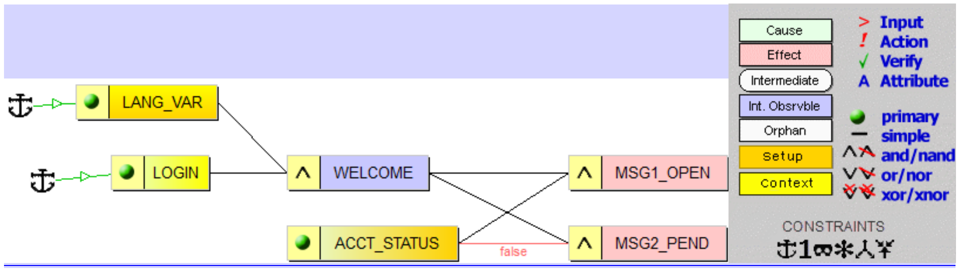
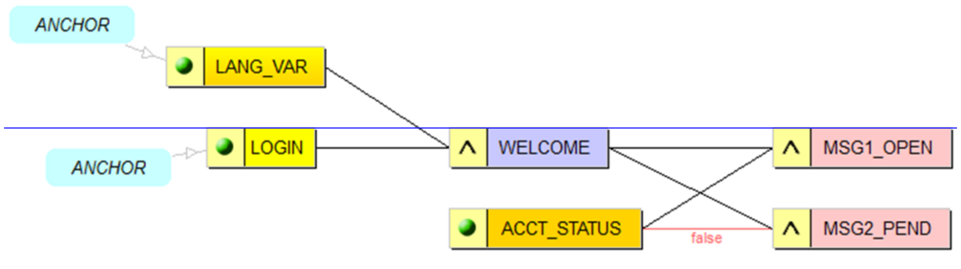


Figure 1-6: Cause-Effect Model

The above model is not normative. It is included only to support the example.

Comment [MW14]: Issue 18333



~~Figure 1-6: Cause-Effect Model~~

Test Case Definitions

Table 1-6 reflects the Test Case output generated by the Cause-Effect Model in Figure 1-6.

Table 1-6: Test Case Output Generated by a Cause-Effect Model

Test Scenario	Step	Seq	Test Description	TP1 (R01)	TP2 (R01)	TP3 (R02)	TP4 (R03)
1	001T	1	System is configured to operate in the user's language, represented by <LANG_VAR>.	**X**		**X**	
1	002T	2	User logs in successfully with UserID = <USERID> and Password = <PSSWD>.	**X**		**X**	
1	003T	3	User's Account is Open.	**X**		**X**	
1	007T	4	System displays the Welcome screen.	**X**			
1	009T	5	System displays Message ID#1 (as defined in the Conditional Message Index) for the current language (as defined by <LANG_VAR>), indicating the account is Open.			X	
2	001T	1	System is configured to operate in the user's language, represented by <LANG_VAR>.		X		X
2	002T	2	User logs in successfully with UserID = <USERID> and Password = <PSSWD>.		X		X
2	003F	3	User's Account is Pending.		X		X
2	007T	4	System displays the Welcome screen.		X		
2	008T	5	System displays Message ID#2 (as defined in the Conditional Message Index) for the current language (as				X

Test Scenario	Step	Seq	Test Description	TP1 (R01)	TP2 (R01)	TP3 (R02)	TP4 (R03)
			defined by <LANG_VAR>), indicating the account is Pending.				▨

Notes:

1. "Test Scenario" is the executable test case. There are 2 Test Scenarios.
2. "TP1," "TP2," etc. are "Test Paths." There are 4 Test Paths.
3. Test Paths are collections of steps which when executed in the same Test Scenario verify related Requirements.
4. Each Test Path traces to one or more Requirements – in this case one each. In this example, each Test Scenario exercises two Test Paths.
5. Failure of one Test Path in a Test Scenario does not necessarily imply failure of another.

In this example, Step 007T in Test Scenario 1 is returned with a Verdict of "FAIL". All other Steps pass. This gives rise to a set of questions for deriving Requirement Verification Status from this set of results.

1. What is the Status of Requirement 02? It is exercised by a single Test Path (TP3). All the steps in TP3 passed, but other unrelated steps in the Scenario that exercises TP3 failed.
2. What is the Status of Requirement 01? It is exercised by 2 Test Paths (TP1, TP2). One passed, one failed.
3. What is the status of Test Scenario 1? Part of it succeeded, part of it failed.

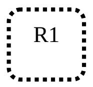
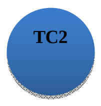
Different systems, projects, and organizations have different answers to these questions. Some may be more nuanced than others. The answers to these types of questions are what make up the rules for Test Results Arbitration. Arbitration itself is outside the scope of the TestIF standard, but the standard must support interchange of sufficient data to support external arbitration.

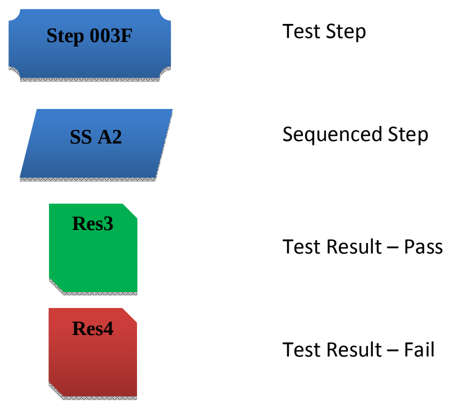
Translating to TestIF

Test Scenarios and Test Paths as Test Cases Related by Custom Attribute
TestIF does not provide a specific semantically defined structure for differentiating between and relating Test Scenarios and Test Paths as defined in this example – they are tool-specific concepts. Both concepts map well to the TestIF TestCase Test Object, because they are a sequenced set of reusable Test Steps.

The following diagrams use the example described above to show how TestIF can be extended to set up this relationship. Further, it shows how step-level execution results support test result higher-level arbitration of requirement verification status by tracing back through the Scenario-Test Path relationship to the requirement. Table 1-7 is a legend of the symbols used in the subsequent diagrams. The complete XML for this example can be found as an Appendix to this document.

Table 1-7: Legend for Subsequent Diagrams

Symbols	Definition
	Requirement (Externally Defined)
	Test Case



Test Cases and Requirement Traceability

Figure 1-7 shows how the ‘Test Paths’ from the example above are defined as Test Case objects and related to externally defined requirements using an attribute. The ‘Test Paths’ are also related to the ‘Test Scenarios’ they belong to by defining the ‘Test Scenarios’ as Test Cases, and relating the ‘Test Paths’ and ‘Test Scenarios’ using another attribute.

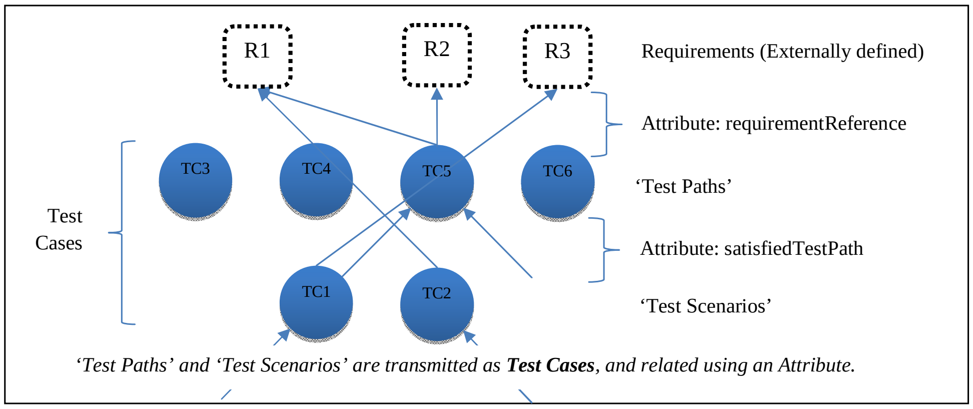
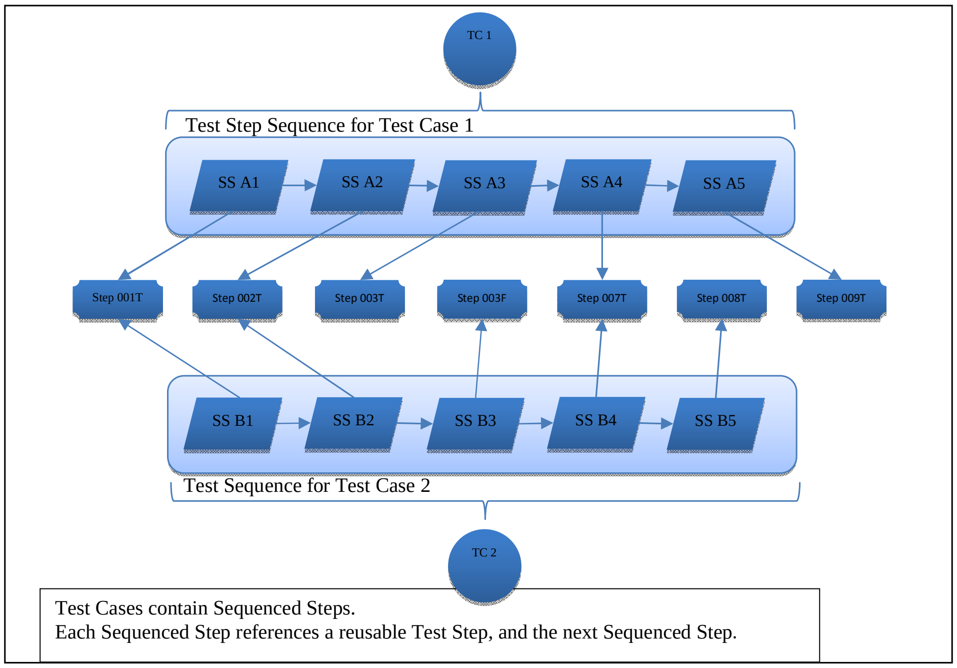

R1
R2
R3
Requirements (Externally defined)
Attribute: requirementReference
TC3
TC4
TC5
TC6
‘Test Paths’
Test Cases
Attribute: satisfiedTestPath
TC1
TC2
‘Test Scenarios’
‘Test Paths’ and ‘Test Scenarios’ are transmitted as Test Cases, and related using an Attribute.

Figure 1-7: The ‘Test Paths’ Relationship

Test Cases and Test Steps

Figure 1-8 shows how Test Steps are defined outside of Test Cases, so they can be reused in different sequences defined by different Test Cases.



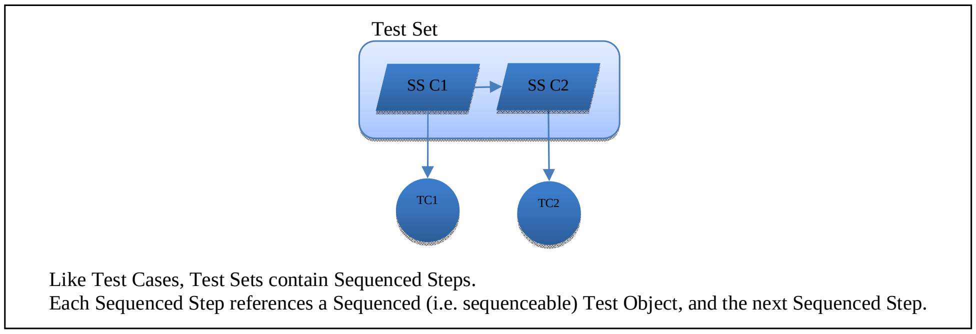
Test Cases contain Sequenced Steps.
Each Sequenced Step references a reusable Test Step, and the next Sequenced Step.

Figure 1-8: Test Steps Definition Outside of Test Cases

Executable Test Sets

Figure 1-9 shows how a Test Set is created for test execution. The Test Set contains Sequenced Steps, each of which points to a Sequenced Test Object (i.e. one of the three types of Test Objects that can be sequenced [Test Set, Test Case, Test Step]), and the next Sequenced Step.

In this example, the Test Set is simply a sequence of the two Test Cases (the two 'Test Scenarios' from the example above).



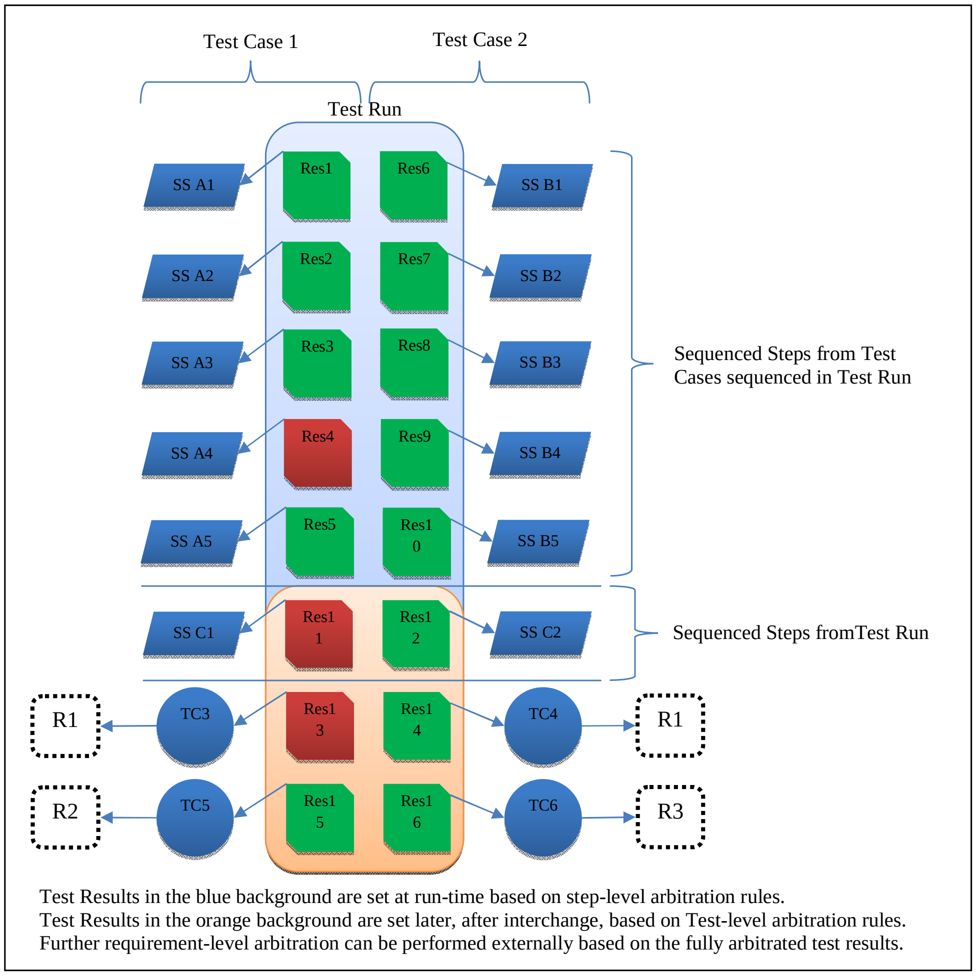
Like Test Cases, Test Sets contain Sequenced Steps.
Each Sequenced Step references a Sequenced (i.e. sequenceable) Test Object, and the next Sequenced Step.

Figure 1-9: Creation of a Test Set

Test Run, Test Results, and Post-Run Result Arbitration

Figure 1-10 shows how Test Runs contain the Test Results from executing the Test Steps that flow from the Test Set. Test Results can reference either a Sequenced Step or a Test Object.

In this example, the Test Results for Test Cases were supplied by an external arbiter after evaluating the Step-level Results from the Run, using the SatisfiedTestPaths attribute to determine which Test Cases to supply results for.



Test Results in the blue background are set at run-time based on step-level arbitration rules.
Test Results in the orange background are set later, after interchange, based on Test-level arbitration rules.
Further requirement-level arbitration can be performed externally based on the fully arbitrated test results.

Figure 1-10: Test Runs Contain the Test Results from Executing the Test Steps

Even though only two Test Cases were sequenced in the Test Set, Test Results can be derived for all 6 Test Cases, allowing for a more granular traceability of results to requirements.

By this external arbiter's rules, a failure in any Test Case step causes that Test Case to fail (Res11, Res13). If R2 had been traced to Test Case 1, its verdict would have been incorrectly set to FAIL.

This structure of combining Test Cases into a single executable set of steps that share environmental variables and sequence is analogous to the UTP concept of "Test Context".

This example demonstrates TestIF's ability to combine test cases ('Test Paths') into executable test contexts ('Test Scenarios') that share environment and control, without sacrificing granularity of requirement traceability. Tests can be combined, and unrelated failures can be disregarded in rollup of Requirement Verification Status.

TestIF supports this power of Cause-Effect Models to optimize both test efficiency and requirement traceability.

1.4 Overall Design Rationale

The following design issues and considerations were prioritized during the development of the proposed specification:

- Interchange
  - Supports interchange between homogeneous and heterogeneous systems and test tools
  - Does not presume any form or method of test authoring (tool-independence)
  - Relevant to common C4I testing scenarios
  - Support interchange of test definitions, not translation of scripting languages
  - Support all kinds of test definitions, including UML Testing Profile (UTP) compliant (or not), model-based definitions, or even a simple spreadsheet.
  - Compliant tools can process the entities they understand and ignore what they don't
  - The standard primarily assumes that it enables software-to-software interchange, as opposed to optimizing for human readability.~~Assume software-to-software interchange, with no special consideration for read/write by people~~

Comment [MW15]: Issue 18329

- Test Results Arbitration Support
  - Leverage UTP concept of separating results arbitration from test execution
  - Support interchange of results of verdict (consistent with UTP concept)
  - Nothing in standard about how verdicts are set
  - Provides for verdict attribution at any level (addresses *Issue to Be Discussed* #1)

- Expressivity vs. strong typing
  - Provide semantic definitions for very common testing terms, within flexible structure
  - Flexibility to extend the standard using simple attribution structures

- Self-Consistency
  - Assume self-consistency of interchanged data
  - E.g. Verdicts only apply to the versions of the tests included in the same file
  - No enforcement of referential conflict built into standard – up to implementers

- Compatibility With Adopted Standards
  - Leverage structural concept of attributed identifiables from ReqIF
  - Not trying to be UTP, which defines the entire test system, but consistent with it
  - Explored other standards to ensure no conflict (see also *Relationship to Adopted OMG Specifications*, below)

- Generic
  - Ensure meeting C4I needs first, but support wide adoption
  - Testing space requires interface between wide variety of tools for different purposes
  - Flexibility to support all kinds of test authoring and execution

2 Conformance

Implementations of this standard are considered to be in conformance if they fully match one or more of the language-level PSMs specified. The implementation must indicate the language PSMs that they match in their statement of conformance.

Comment [MW16]: Issue 18339

- Must follow hierarchy rules defined in the text of the PIM for allowed sequence containment (i.e., a Test Set can only contain any other SequencedTestObject, a Test Case can only contain Test Steps, and a Test Step can only contain Test Steps). See the diagram in the TestIF Package section and the diagram labeled "Conceptual Relationship between these SequencedTestObject Types".

- Identifiers are required to be unique within a TestIF document. Data that is expected to be referenced outside of the TestIF document or across TestIF documents must have a UUID. TestIF does not specify the mechanism for declaring UUIDs. There are various startegies for defining UUIDs that are acceptable. For "transient" objects, only relevant within the scope of a single TestIF document may use local identifiers. Local Identifiers must begin with "localonly." Other uses of Identifier are considered to be non-conforming.

Comment [MW17]: Issue 18341

- When creating custom AttributeDefinition objects, it is recommended to use Identifiers that provide appropriate attribution of the author (tool). The pre-defined AttributeDefinitions in TestIF are all prefixed with org.omg.TestIF.

- Tools shall use the semantic structures prescribed by the standard where applicable. Usage of the flexibility features of relatedTestObjects and Attributes to convey concepts that are already semantically covered elsewhere in the standard via direct structure or attribute is considered non-conforming.

- TestItems are provided to cover the cases where the semantic meanings of the other objects are too restrictive or do not match, but to be useful and understandable by other tools TestItems should always have attributes attached to them to define their semantics using the standard extension mechanism.

- It is not necessary for a conformant tool ~~Conformant~~ to ~~exporting~~ export ~~tools are not required~~ or ~~to~~ produce all of the object types outlined in the standard if that tool does not use all of the types. If a tool states that they can import TestIF artifact, they must preserve ~~support~~ all of the data types and features of the specification.

Comment [MW18]: Discuss in the FTF further refinement of this conformance point.

3 References

3.1 Normative

The following normative documents contain provisions which, through reference in this text, constitute provisions of this TestIF Specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

The following normative documents contain provisions that, through reference in this text, constitute provisions of this specification:

URI

- Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter, IETF RFC 2396, August 1998

  http://www.ietf.org/rfc/rfc2396.txt

XHTML 1.1 Modularization

- XHTML™ Modularization 1.1, Daniel Austin et al., eds., W3C, 8 October 2008

  http://www.w3.org/TR/xhtml-modularization/

XML 1.0 (Second Edition)

- Extensible Markup Language (XML) 1.0, Second Edition, Tim Bray et al., eds., W3C, 6 October 2000

  http://www.w3.org/TR/REC-xml

XML-Namespaces

- Namespaces in XML, Tim Bray et al., eds., W3C, 14 January 1999

  http://www.w3.org/TR/REC-xml-names

XML-Schema

The authoritative description of the Test Interchange Format exchange document structure is provided as an XML Schema. XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents.

- XML Schema Part 1: Structures, Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, W3C, 2 May 2001

  http://www.w3.org/TR/xmlschema-1//

- XML Schema Part 2: Datatypes, Paul V. Biron and Ashok Malhotra, eds., W3C, 2 May 2001

  http://www.w3.org/TR/xmlschema-2/

3.2 Relationship with other standards

No change~~s~~ to ~~UML 2.0~~ any ~~or other~~ OMG specification~~s~~ ~~are~~ is required.

3.2.1 UML Testing Profile (UTP)

UTP is a powerful framework for creating abstract test models that completely define the testing space and expected system behaviours therein. UTP strives to be a language for methodologies.

TestIF supports UTP by providing a means to communicate the information stored in UTP models to other tools that do not 'speak' UTP. This includes both model-based and non-model-based test authoring tools, as well as other kinds of tools that rely on test information.

TestIF addresses the reality that there are many kinds of tools involved with testing that use and contribute to test case information. Tools for Test Management, Automation Frameworks, Results Arbitration, and others all rely on interchange of test case information.

Adoption of UTP is not yet widespread. Other frameworks for elaborating test specifications are pervasive. UTP has the potential to become the underlying framework of a wide variety of test design tools. TestIF fills the need of every one of those tools to express an interchangeable specification for test behavior in the form of executable test cases.

Please see the table in “Responses to Issues to be Discussed,” Item #1, for a detailed treatment of the relationship between specific TestIF and UTP concepts.

UTP Home: http://utp.omg.org/

3.2.2 ReqIF

ReqIF is an XML interchange standard for exchange of software requirements. It is directly analogous to TestIF in its purpose and structure.

Where possible, TestIF leverages ReqIF concepts and patterns. These include:

- The concept of extensibility through Attributes
- General XML PSM structure
- Providing a core semantic structure to support extensibility and ease of adoption

Both TestIF and ReqIF provide an XML PSM, which allows for ease of implementation to the standard, encouraging adoption.

ReqIF Home: http://www.omg.org/spec/ReqIF/

3.2.3 Systems Assurance

Sometimes referred to as "Software Assurance," Systems Assurance is an umbrella term that covers a number of OMG standards.

While there are some analogous concepts in the various SA standards, TestIF addresses needs that are outside the scope of the SA standards. Indeed, TestIF could be extended for transmitting SA information in a variety of ways.

Systems Assurance home: http://www.omgwiki.org/SysA/doku.php

3.2.4 Semantics of Business Vocabulary and Business Rules (SBVR)

SBVR provides a framework for writing business rules (not in terms of system). Test design or authoring tools may benefit from leveraging SBVR by writing test step descriptions according to a structure specified in an SBVR framework. SBVR is thus 'upstream' from TestIF.

3.2.5 Software Assurance Evidence Metamodel (SAEM)

SAEM uses SBVR to define a vocabulary for *evidence* about software artifacts to support assurance *arguments* against *claims* (typically related to safety and security).

- Claims define the expected behavior of the system. In system design and testing, claims are conceptually analogous to requirements or test cases (depending on the methodology).
- Evidence describes anything that might be used to support an argument that the system satisfies a claim.
    - "**Evidence** can be diverse as various things may be produced as evidence, such as documents, expert testimony, ***test results***, measurement results, records related to process, product, and people, etc."

      TestIF supports transmittal of any kind of result, including external references to documents.
- Arguments are potentially more complex (see ARM, below). In TestIF, an Argument can be thought of as an assertion made by some external Arbiter based on the information transmitted in TestIF format.

TestIF can be extended to support transmittal of all kinds of information. The core concepts of the SAEM are analogous to common testing concepts. As such, TestIF may be a good choice for transmittal of SAEM information.

3.2.6 Argumentation Metamodel (ARM)

ARM provides a structure for making claims about security, in support of the SAEM.

TestIF can be extended to support transmittal of ARM information.

3.2.7 Systems Modeling Language (SysML)

SysML provides a broad set of notations and tools for describing a system. SysML includes the concept of test case, requirements, and other items which have some overlap of the concepts covered in UTP, SysML, and the concepts requested in this standard.

While some terms may be shared across the standards, TestIF provides a general semantic framework that allows for any standard or tool to communicate test information according to its own definition of a given shared term.

TestIF can be extended to reference any externally identifiable thing in the SysML model, and attach that reference to any Test Object in the TestIF structure, providing complete traceability to any part of the model.

3.2.8 SysML Requirements Management

SysML includes a graphical construct to represent text based requirements and relate them to other model elements. The requirements diagram captures requirements hierarchies and requirements derivation. The ‘satisfy’ and ‘verify’ relationships allow a modeler to relate a requirement to a model element that satisfies or verifies the requirements. The requirement diagram provides a bridge between the typical requirements management tools and the system models.

TestIF can be extended to reference any externally identifiable thing in the SysML model, and attach that reference to any Test Object in the TestIF structure, providing complete traceability to any part of the model, including Requirements and related objects.

SysML home: http://www.omgsysml.org/

3.2.9 MARTE~~S~~ (Modeling and Analysis of Real-Time and Embedded “Systems”)

MARTE~~S~~ provides facilities to annotate UML models with information required to conduct performance and schedulability analysis on real-time embedded systems. It also defines a general framework for quantitative analysis which can be leveraged to refine/specialize any other kind of analysis. The primary objectives of MARTE~~S~~:

- Provide a common way of modeling both hardware and software aspects of a RTES in order to improve communication between developers.
- Enable interoperability between development tools used for specification, design, verification, code generation, etc.
- Foster the construction of models that may be used to make quantitative predictions regarding real-time and embedded features of systems taking into account both hardware and software characteristics.

TestIF could be used for transmitting MARTES-related test info.

MARTE~~S~~ home: http://www.omgmarte.org/

Comment [MW19]: Issue 18356

3.2.10 UML Profile for DODAF/MODAF (UPDM)

The Unified Profile for DoDAF and MODAF (UPDM) supports for DoDAF and MODAF. This provides a standard means of describing DoDAF and MODAF compliant architectures using UML and SysML. UPDM significantly enhances the quality, productivity, and effectiveness associated with enterprise and system of systems architecture modeling. UPDM is used to model C4I systems at an architectural level. It is a specification for the UML/SysML/SOAML frameworks.

TestIF can be extended to reference any externally identifiable thing in the SysML model, and attach that reference to any Test Object in the TestIF structure, providing complete traceability to any part of the model.
UPDM home: http://www.omg.org/spec/UPDM/index.htm.

4 Terms and definitions

For the purposes of this specification, the following terms and definitions apply.

4.1 Terms

- TestIF – The abbreviated name for the Test Information Interchange Format standard specified in this document.
- Exchange XML Document – The XML artifact, conformant to the TestIF Standard.
- Exporting TestIF Tool – A software program that generates TestIF conformant artifacts.
- Importing TestIF Tool – A software program that consumes TestIF conformant artifacts.
- Test Authoring Tool – A software program that is used to define, model, and/or specify tests.
- Test Execution Tool – A software program that utilizes automation techniques to perform tests.
- UUID – Universally Unique Identifier.
- Requirement – A statement specifying the necessary functionality and expected performance of a system or component.
- ReqIF – Requirements Interchange Format (OMG Standard)
- C4I – A term used in military and command situations that is an abbreviation for Command, Control, Computers, Communications, and Intelligence.

4.2 General Notes on Semantics

In the testing space, many standards, tool vendors, and organizations have their own definitions of what constitutes a Test Case, and other concepts related to testing. TestIF is designed to support ALL definitions of what a “Test Case” is.

The semantics of TestIF are intended to support universal concepts that are foundational to testing, providing an extensible core for all testing tools to build around.

The variety of tools, systems, test environments, test approaches, and project needs around testing begs for an interchange standard that provides a common framework for each interested tool to document and communicate not only its test information, but also an explicit definition of how to interpret the information.

Among the highest aims of TestIF is to not be at odds with any one standard’s or tool’s definition of what a “Test Case” is.

5 Symbols (and abbreviated terms)

Table 5-1: Acronyms and Abbreviations

Acronym	Definition
AB	Architecture Board
API	Application Program Interface
ARM	Argumentation Metamodel
ATRT	Automated Test and Re-Test
AUT	Application Under Test
BMD	Ballistic Missile Defense
BoD	Board of Directors
C4I	Command, Control, Computers, Communications, and Intelligence
CCM	CORBA Component Model
CORBA	Common Object Request Broker Architecture
CWM	Common Warehouse Metamodel
DoD	Department of Defense
DTT	Direct-to-Test
HP	Hewlett Packard
HTTP	HyperText Transfer Protocol
IDL	identification
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Standardization Organization
LCS	Littoral Combat Ship
LOI	Letter of Intent
MARTES	Modeling and Analysis of Real-Time and Embedded Systems
MDA	Model Driven Architecture
MOF	Meta Object Facility
OMG	Object Management Group, Inc.
PEO IWS	Program Executive Office Integrated Warfare System
PEO SUB	Program Executive Office for Submarines
PIM	Platform Independent Model
PSM	Platform Specific Model
RVS	Requirement Verification Status
QTP	QuickTest Professional
RFI	Request for Information
RFP	Request for Proposal
RM-ODP	Reference Model of Open Distributed Processing
SAEM	Software Assurance Evidence Metamodel
SBVR	Semantics of Business Vocabulary and Business Rules

Acronym	Definition
SPEM	Software Process Engineering Metamodel
SQL	Structured Query Language
SUT	System Under Test
SysML	Systems Modeling Language
TC	Technology Committee
TF	Task Force
TMX	Test Management and Execution
TO	Testing Organization
UML	Unified Modeling Language
UPDM	UML Profile for DODAF/MODAF
URI	Uniform Resource Identifiers
URL	User Requirements Language
UTP	UML Testing Profile
UUID	Universally Unique Identifier
XHTML	Extensible Hypertext Markup Language
XMI	XML Metadata Interchange
XML	eXtensible Mark-up Language

6 Additional Information

6.1 Changes to Existing OMG Specifications

~~There are n~~No change~~s~~ to any ~~existing~~ OMG Specification~~s~~ is required as a result of this standard.

Comment [MW20]: Issue 18356

6.2 Acknowledgements

The following companies submitted this specification:

- SimVentions
- IDT

The following companies supported this specification:

* Critical Logic
* Intervise Consulting
* Lockheed Martin

7 Platform Independent Model (PIM)

The PIM consists of the following logical packages:

- org.omg.TestIF – root package of the spec containing the high level and container items of the standard
- org.omg.TestIF.Attributes – package that contains Attribute related classes
- org.omg.TestIF.Test Classes – package that contains Test Specific classes~~Package TestIF~~

Comment [MW21]: Issue 18340

7.1 Package TestIF

TestIF is the base package of the standard. The general classes and interfaces of the TestIF standard are kept in the base package. Examine the sub-packages for the definition of Attributes and specific test related classes.

All objects in TestIF contain the common attributes of being "Identifiable". Objects that require additional (and unbounded) attributes add the "AttributedIdentifiable" classification. All items that extend this abstraction have an unbound list of AttributeValues in addition to the basic characteristics of being Identifiable.

Figure 7-1 shows the relationships between these types.

Figure 7-2 provides an overview of the structure of information types and groups in TestIF.

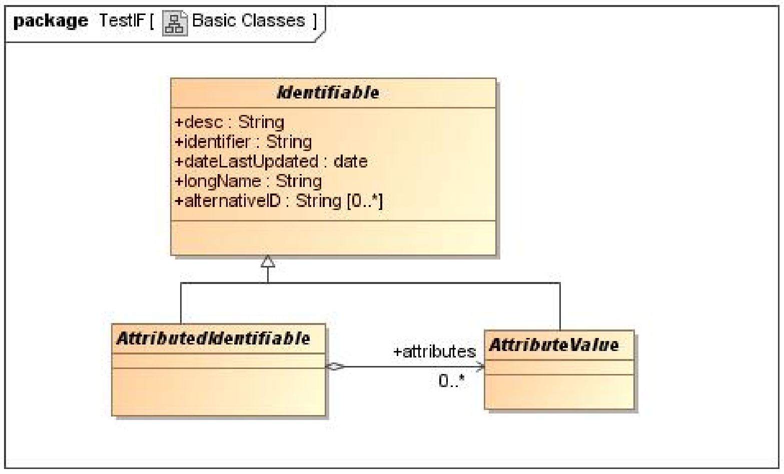


Figure 7-1: Basic Classes

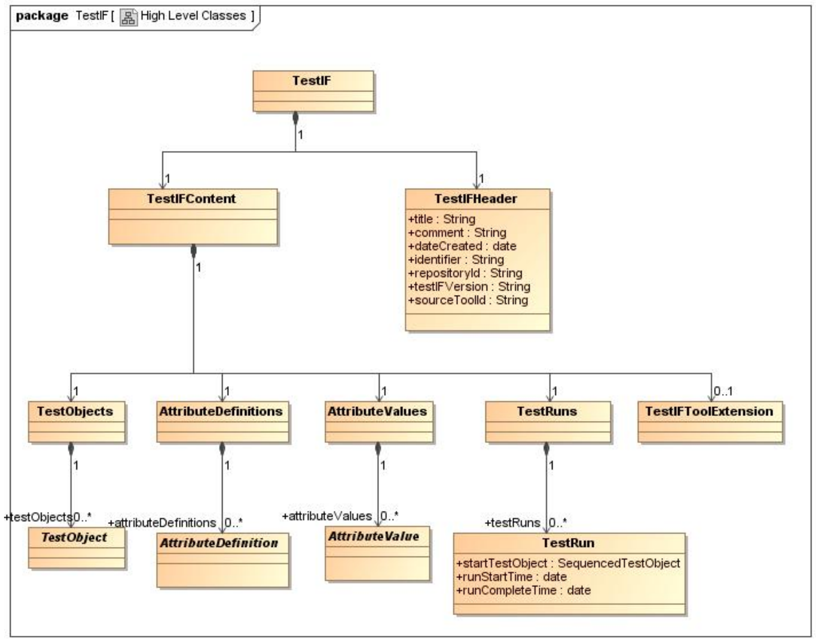


Figure 7-2: High Level Classes

7.1.1 Class AttributeDefinitions

A container for all AttributeDefinition items to be exchanged in TestIF

Name	AttributeDefinitions
Qualified Name	TestIF::AttributeDefinitions
Visibility	public
Abstract	false
Base Classifier	
Realized Interface	

7.1.1.1 Fields / Attributes

7.1.1.1.1 attributeDefinitions

Type	AttributeDefinition
Default Value	
Visibility	public
Multiplicity	0..*

7.1.2 Class AttributedIdentifiable

A Class AttributedIdentifiable is an object that has attributes.

Name	**AttributedIdentifiable**
Qualified Name	**TestIF::AttributedIdentifiable**
Visibility	**Public**
Abstract	**True**
Base Classifier	**•Identifiable**
Realized Interface	

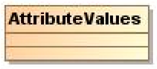
7.1.2.1 Fields / Attributes

7.1.2.1.1 attributes

Type	AttributeValue
Default Value	
Visibility	public
Multiplicity	0..*

7.1.3 Class AttributeValues

A container for all AttributeValue items to be exchanged in TestIF



Name	AttributeValues
Qualified Name	TestIF::AttributeValues

Visibility	public
Abstract	false
Base Classifier	
Realized Interface	

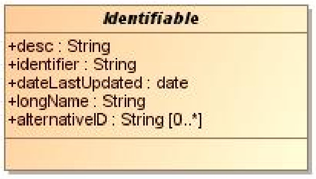
7.1.3.1 Fields / Attributes

7.1.3.1.1 attributeValues

Type	AttributeValue
Default Value	
Visibility	Public
Multiplicity	0..*

7.1.4 Class Identifiable

The Class Identifiable is the base class of identifiable objects in TestIF.



Name	Identifiable
Qualified Name	TestIF::Identifiable
Visibility	
Abstract	True
Base Classifier	
Realized Interface	

7.1.4.1 Fields / Attributes

7.1.4.1.1 alternativeID

Optional

Type	String
Default Value	
Visibility	public
Multiplicity	0..*

7.1.4.1.2 dateLastUpdated

Type	date
Default Value	
Visibility	public
Multiplicity	

7.1.4.1.3 desc

This is an optional description of the object.

Type	String
Default Value	
Visibility	public
Multiplicity	

7.1.4.1.4 identifier

This is a unique identifier for the object within the TestIF interchange file ~~document~~. If the object needs to be referenced outside ~~this document~~the TestIF interchange file, use a UUID.

Comment [MW22]: Issue 18341

Type	String
Default Value	
Visibility	public
Multiplicity	

7.1.4.1.5 longName

Optional

Type	String
Default Value	
Visibility	public
Multiplicity	

7.1.5 Class TestIF

The Class TestIF is the base container node for all TestIF Content.

Name	TestIF
Qualified Name	TestIF::TestIF
Visibility	Public
Abstract	False
Base Classifier	
Realized Interface	

7.1.5.1 Fields / Attributes

7.1.5.1.1 content

Type	TestIFContent
Default Value	
Visibility	Public
Multiplicity	1

7.1.5.1.2 header

Type	TestIFHeader
Default Value	
Visibility	public
Multiplicity	1

7.1.6 Class TestIFContent

The Class TestIFContent is the container node for the Test Objects defined in TestIF

Name	TestIFContent
Qualified Name	TestIF::TestIFContent
Visibility	public
Abstract	false
Base Classifier	
Realized Interface	

7.1.6.1 Fields / Attributes

7.1.6.1.1 extensionData

Type	TestIFToolExtension
Default Value	
Visibility	Public
Multiplicity	0..1

7.1.6.1.2 attributeDefinitions

Type	AttributeDefinitions
Default Value	
Visibility	Public
Multiplicity	1

7.1.6.1.3 attributeValues

Type	AttributeValues
Default Value	
Visibility	Private
Multiplicity	1

7.1.6.1.4 testRuns

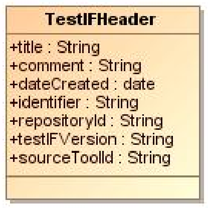
Type	TestRuns
Default Value	
Visibility	Public
Multiplicity	1

7.1.6.1.5 testObjects

Type	TestObjects
Default Value	
Visibility	Private
Multiplicity	1

7.1.7 Class TestIFHeader

This element contains metadata for the exchange file.



Name	TestIFHeader
Qualified Name	TestIF::TestIFHeader
Visibility	
Abstract	false
Base Classifier	
Realized Interface	

7.1.7.1 Fields / Attributes

7.1.7.1.1 comment

Optional

Type	String
Default Value	
Visibility	public

Multiplicity	

7.1.7.1.2 dateCreated

This is the date that the project was initially created.

Type	date
Default Value	
Visibility	public
Multiplicity	

7.1.7.1.3 identifier

Optional

Type	String
Default Value	
Visibility	public
Multiplicity	

7.1.7.1.4 repositoryId

This is an optional unique identifier of the repository containing the test definitions that have been exported, such as database ID or URL.

Type	String
Default Value	
Visibility	public
Multiplicity	

7.1.7.1.5 sourceToolId

This is an optional identifier of the exporting tool.

Type	String
Default Value	
Visibility	public
Multiplicity	

7.1.7.1.6 testIFVersion

Version of TestIF that this document supports

Type	String
Default Value	
Visibility	public

Multiplicity	

7.1.7.1.7 title

This is the title of the project that this TestIF document is capturing. This item is optional.

Type	String
Default Value	
Visibility	Public
Multiplicity	

7.1.8 Class TestIFToolExtension

This element can contain tool specific interchange information which cannot be transported in core TestIF content. This class serves as an "anchor" class where non-standard implementations can add to or extend with their own data. It provides a structural placeholder in the specification.

Comment [MW23]: Issue 18342

Name	TestIFToolExtension
Qualified Name	TestIF::TestIFToolExtension
Visibility	
Abstract	false
Base Classifier	
Realized Interface	

7.2 Package TestIF::Attributes

The Attributes package contains the classes and interfaces that comprise the Attribute definitions and values for the TestIF standard.

Each concrete attribute value that is used in TestIF needs to be valid against its related data type (AttributeDefinition). For example: the value of a "priority"-attribute may need to be an integer number, while the value for a "status"-attribute may need to be picked from a list of choices. In TestIF, each attribute value (AttributeValue element) is related to the definition object that specifies what the attribute represents and its data type. TestIF also supports the concept of Composite Attributes. This allows the user to specify complex strucutres for attributes that can be referenced by their "root" AttributeValueComposite" object.

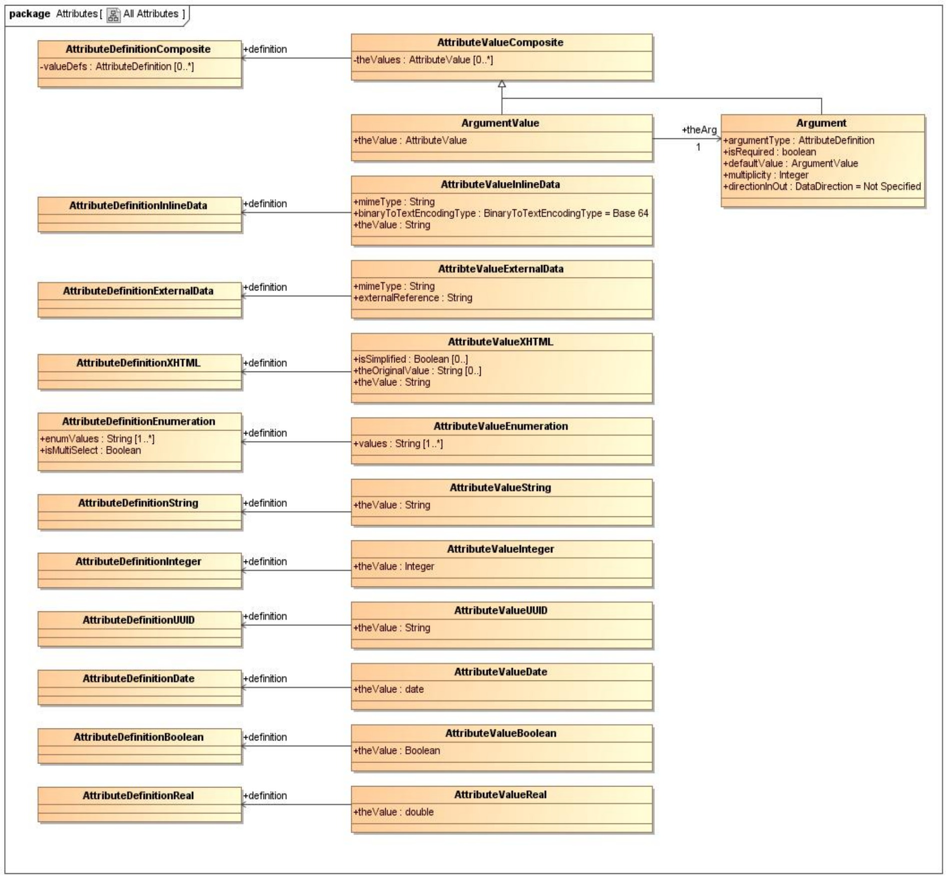
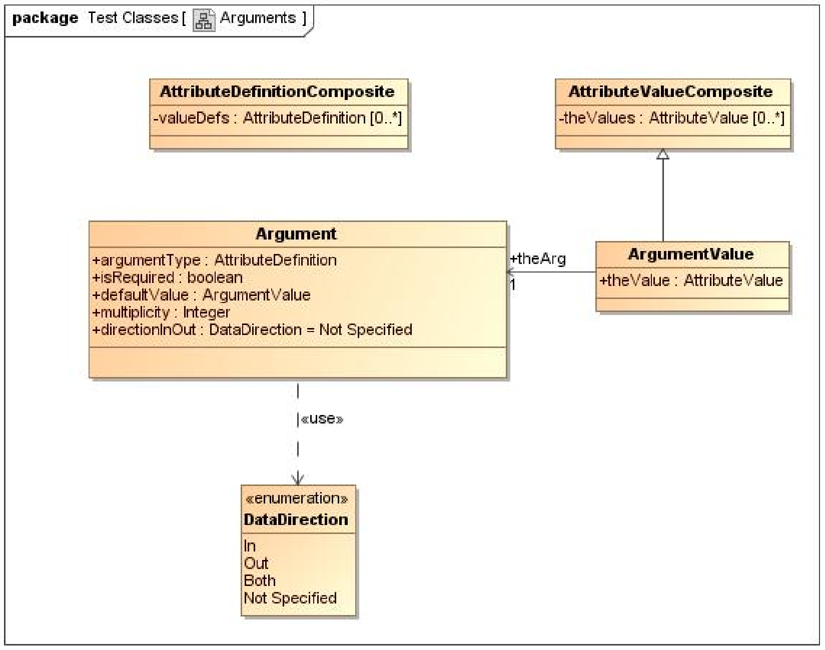


Figure 7-3: All Attributes

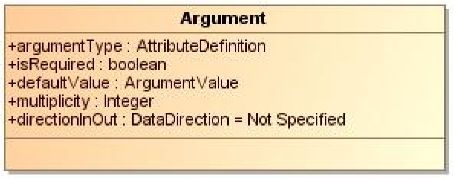
7.2.1 Class Argument

Arguments specify the parameters (in and/or out) relevant to performing any particular step. Arguments provide the mechanism for describing the inputs to a Sequenced Test Object. TestIF uses a combination of ArgumentDefinition and Argument Value to distinguish the declarative options for input values versus the “runtime” values of any particular test sequence. Arguments provide an explicit mechanism for tying test inputs to sequences beyond simply using Attributes. Figure 7-4 shows the logical relationships where Arguments are used in TestIF.



package Test Classes [Arguments]
AttributeDefinitionComposite
-valueDefs : AttributeDefinition [0..*]
AttributeValueComposite
-theValues : AttributeValue [0..*]
Argument
+argumentType : AttributeDefinition
+isRequired : boolean
+defaultValue : ArgumentValue
+multiplicity : Integer
+directionInOut : DataDirection = Not Specified
+theArg
1
ArgumentValue
+theValue : AttributeValue
«use»
«enumeration»
DataDirection
In
Out
Both
Not Specified

Figure 7-4: Arguments



Argument
+argumentType : AttributeDefinition
+isRequired : boolean
+defaultValue : ArgumentValue
+multiplicity : Integer
+directionInOut : DataDirection = Not Specified

Name	Argument
Qualified Name	TestIF::Attributes::Argument
Visibility	public
Abstract	false
Base Classifier	•AttributeValueComposite
Realized Interface	

7.2.1.1 Fields / Attributes

7.2.1.1.1 argumentType

This specifies the type of field that this argument represents

Type	AttributeDefinition
Default Value	
Visibility	public
Multiplicity	

7.2.1.1.2 defaultValue

This is the default value of the argument.

Type	ArgumentValue
Default Value	
Visibility	public
Multiplicity	

7.2.1.1.3 directionInOut

This indicates the "direction" of the argument (in, out, both, or not specified).

Type	DataDirection
Default Value	Not Specified
Visibility	public
Multiplicity	

7.2.1.1.4 isRequired

This flag is to indicate whether a value is required for the argument.

Type	boolean
Default Value	
Visibility	public
Multiplicity	

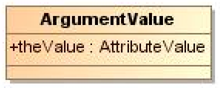
7.2.1.1.5 multiplicity

This is the permitted number of values for this argument.

Type	Integer
Default Value	
Visibility	public
Multiplicity	

7.2.2 Class ArgumentValue

Arguments specify the parameters (in and/or out) relevant to performing any particular step. Arguments provide the mechanism for describing the inputs to a Sequenced Test Object. TestIF uses a combination of ArgumentDefinition and Argument Value to distinguish the declarative options for input values versus the "runtime" values of any particular test sequence. Arguments provide an explicit mechanism for tying test inputs to sequences beyond simply using Attributes.



Name	ArgumentValue
Qualified Name	TestIF::Attributes::ArgumentValue
Visibility	public
Abstract	false
Base Classifier	•AttributeValueComposite
Realized Interface	

7.2.2.1 Fields / Attributes

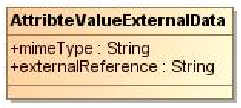
7.2.2.1.1 theArg

Type	Argument
Default Value	
Visibility	Public
Multiplicity	1

7.2.2.1.2 theValue

Type	AttributeValue
Default Value	
Visibility	public
Multiplicity	

7.2.3 Class AttribteValueExternalData



Name	AttribteValueExternalData
Qualified Name	TestIF::Attributes::AttribteValueExternalData
Visibility	public
Abstract	false
Base Classifier	•AttributeValue
Realized Interface	

7.2.3.1 Fields / Attributes

7.2.3.1.1 definition

Type	AttributeDefinitionExternalData
Default Value	
Visibility	public
Multiplicity	

7.2.3.1.2 externalReference

Machine readable reference to the external content expected to be a URL or relative path to an item contained in the distributed TestIF artifact.

Type	String
Default Value	
Visibility	public
Multiplicity	

7.2.3.1.3 mimeType

Type	String
Default Value	
Visibility	Public
Multiplicity	

7.2.4 Class AttributeDefinition

The abstract super-class for the different types of "attribute definitions". The "attribute definition" is ~~in priniciple~~ the ~~definition~~ specification of an attribute's type ~~column~~ within a TestIF document ~~n~~ ~~RE/RM tool~~ (but without concrete values).

Comment [MW24]: Issue 18343

AttributeDefinition

Name	AttributeDefinition
Qualified Name	TestIF::Attributes::AttributeDefinition
Visibility	
Abstract	True
Base Classifier	Identifiable
Realized Interface	

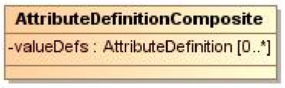
Comment [MW25]: Issue 18344

7.2.5 Class AttributeDefinitionBoolean

AttributeDefinitionBoolean

Name	AttributeDefinitionBoolean
Qualified Name	TestIF::Attributes::AttributeDefinitionBoolean
Visibility	
Abstract	False
Base Classifier	•AttributeDefinition
Realized Interface	

7.2.6 Class AttributeDefinitionComposite


AttributeDefinitionComposite
-valueDefs : AttributeDefinition [0..*]

Name	AttributeDefinitionComposite
Qualified Name	TestIF::Attributes::AttributeDefinitionComposite
Visibility	Public
Abstract	False
Base Classifier	•AttributeDefinition
Realized Interface	

7.2.6.1 Fields / Attributes

7.2.6.1.1 valueDefs

Type	AttributeDefinition
Default Value	
Visibility	private
Multiplicity	0..*

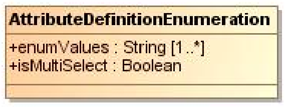
7.2.7 Class AttributeDefinitionDate

AttributeDefinitionDate

Name	AttributeDefinitionDate
Qualified Name	TestIF::Attributes::AttributeDefinitionDate
Visibility	
Abstract	false
Base Classifier	•AttributeDefinition
Realized Interface	

7.2.8 Class AttributeDefinitionEnumeration

This is a definition of a requirement attribute that is based on an "Enumeration" data type. In principle, this element constitutes an attribute column that can contain enumeration values of a certain enumeration data type.



Name	AttributeDefinitionEnumeration
Qualified Name	TestIF::Attributes::AttributeDefinitionEnumeration
Visibility	
Abstract	false
Base Classifier	•AttributeDefinition
Realized Interface	

7.2.8.1 Fields / Attributes

7.2.8.1.1 enumValues

This is the list of enumeration value options.

Type	String
Default Value	
Visibility	public
Multiplicity	1..*

7.2.8.1.2 isMultiSelect

This is a flag indicating whether the selection is single or multiple choice enumerations.

Type	Boolean
Default Value	
Visibility	public
Multiplicity	

7.2.9 Class AttributeDefinitionExternalData

Name	AttributeDefinitionExternalData
Qualified Name	TestIF::Attributes::AttributeDefinitionExternalData
Visibility	Public
Abstract	False
Base Classifier	•AttributeDefinition
Realized Interface	

7.2.10 Class AttributeDefinitionInlineData

Name	AttributeDefinitionInlineData
Qualified Name	TestIF::Attributes::AttributeDefinitionInlineData
Visibility	public
Abstract	false
Base Classifier	•AttributeDefinition
Realized Interface	

7.2.11 Class AttributeDefinitionInteger

Name	AttributeDefinitionInteger
Qualified Name	TestIF::Attributes::AttributeDefinitionInteger
Visibility	
Abstract	False
Base Classifier	•AttributeDefinition
Realized Interface	

7.2.12 Class AttributeDefinitionReal

Name	AttributeDefinitionReal
Qualified Name	TestIF::Attributes::AttributeDefinitionReal
Visibility	
Abstract	false
Base Classifier	•AttributeDefinition
Realized Interface	

7.2.13 Class AttributeDefinitionString

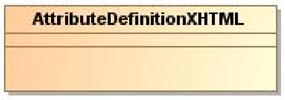
Name	AttributeDefinitionString
Qualified Name	TestIF::Attributes::AttributeDefinitionString
Visibility	
Abstract	False
Base Classifier	•AttributeDefinition
Realized Interface	

7.2.14 Class AttributeDefinitionUUID

Name	AttributeDefinitionUUID
Qualified Name	TestIF::Attributes::AttributeDefinitionUUID
Visibility	Public
Abstract	False
Base Classifier	•AttributeDefinition
Realized Interface	

7.2.15 Class AttributeDefinitionXHTML

This is a definition of a requirement attribute that is based on a formatted data type.



Name	AttributeDefinitionXHTML
Qualified Name	TestIF::Attributes::AttributeDefinitionXHTML
Visibility	
Abstract	false
Base Classifier	•AttributeDefinition
Realized Interface	

7.2.16 Class AttributeValue

This is the abstract super-class for concrete values of the different data type.

Name	AttributeValue
Qualified Name	TestIF::Attributes::AttributeValue
Visibility	
Abstract	True
Base Classifier	•Identifiable
Realized Interface	

7.2.17 Class AttributeValueBoolean

AttributeValueBoolean
+theValue : Boolean

Name	AttributeValueBoolean
Qualified Name	TestIF::Attributes::AttributeValueBoolean
Visibility	
Abstract	false
Base Classifier	•AttributeValue
Realized Interface	

7.2.17.1 Fields / Attributes

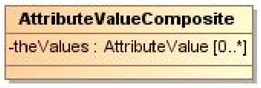
7.2.17.1.1 definition

Type	AttributeDefinitionBoolean
Default Value	
Visibility	public
Multiplicity	

7.2.17.1.2 theValue

Type	Boolean
Default Value	
Visibility	public
Multiplicity	

7.2.18 Class AttributeValueComposite



Name	AttributeValueComposite
Qualified Name	TestIF::Attributes::AttributeValueComposite
Visibility	public
Abstract	false
Base Classifier	•AttributeValue
Realized Interface	

7.2.18.1 Fields / Attributes

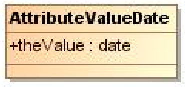
7.2.18.1.1 definition

Type	AttributeDefinitionComposite
Default Value	
Visibility	public
Multiplicity	

7.2.18.1.2 theValues

Type	AttributeValue
Default Value	
Visibility	private
Multiplicity	0..*

7.2.19 Class AttributeValueDate



Name	AttributeValueDate
Qualified Name	TestIF::Attributes::AttributeValueDate
Visibility	
Abstract	False
Base Classifier	•AttributeValue
Realized Interface	

7.2.19.1 Fields / Attributes

7.2.19.1.1 definition

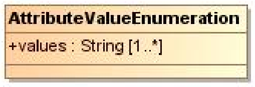
Type	AttributeDefinitionDate
Default Value	
Visibility	Public
Multiplicity	

7.2.19.1.2 theValue

Type	Date
Default Value	
Visibility	Public
Multiplicity	

7.2.20 Class AttributeValueEnumeration

This contains the concrete values of an "Enumeration" data type. Note that in case of "multi value enumerations", a set of different enumeration values can be specified. The value is thus indicated by multiple references ("values") to enumeration values that are contained in the associated enumeration data type.



Name	AttributeValueEnumeration
Qualified Name	TestIF::Attributes::AttributeValueEnumeration
Visibility	
Abstract	false
Base Classifier	•AttributeValue
Realized Interface	

7.2.20.1 Fields / Attributes

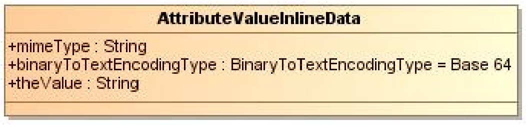
7.2.20.1.1 definition

Type	AttributeDefinitionEnumeration
Default Value	
Visibility	public
Multiplicity	

7.2.20.1.2 values

Type	String
Default Value	
Visibility	public
Multiplicity	1..*

7.2.21 Class AttributeValueInlineData


AttributeValueInlineData
+mimeType : String
+binaryToTextEncodingType : BinaryToTextEncodingType = Base 64
+theValue : String

Name	AttributeValueInlineData
Qualified Name	TestIF::Attributes::AttributeValueInlineData
Visibility	Public
Abstract	False
Base Classifier	•AttributeValue
Realized Interface	

7.2.21.1 Fields / Attributes

7.2.21.1.1 binaryToTextEncodingType

Type	BinaryToTextEncodingType
Default Value	Base 64
Visibility	public
Multiplicity	

7.2.21.1.2 definition

Type	AttributeDefinitionInlineData
Default Value	
Visibility	public
Multiplicity	

7.2.21.1.3 mimeType

Type	String
Default Value	
Visibility	public
Multiplicity	

7.2.21.1.4 theValue

Type	String
Default Value	
Visibility	public
Multiplicity	

7.2.22 Class AttributeValueInteger

AttributeValueInteger
+theValue : Integer

Name	AttributeValueInteger
Qualified Name	TestIF::Attributes::AttributeValueInteger
Visibility	
Abstract	False
Base Classifier	•AttributeValue
Realized Interface	

7.2.22.1 Fields / Attributes

7.2.22.1.1 definition

Type	AttributeDefinitionInteger
Default Value	
Visibility	Public
Multiplicity	

7.2.22.1.2 theValue

Type	Integer
Default Value	
Visibility	Public
Multiplicity	

7.2.23 Class AttributeValueReal

AttributeValueReal
+theValue : double

Name	AttributeValueReal
Qualified Name	TestIF::Attributes::AttributeValueReal
Visibility	
Abstract	false
Base Classifier	•AttributeValue
Realized Interface	

7.2.23.1 Fields / Attributes

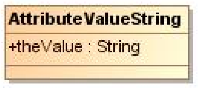
7.2.23.1.1 definition

Type	AttributeDefinitionReal
Default Value	
Visibility	public
Multiplicity	

7.2.23.1.2 theValue

Type	double
Default Value	
Visibility	public
Multiplicity	

7.2.24 Class AttributeValueString



Name	AttributeValueString
Qualified Name	TestIF::Attributes::AttributeValueString
Visibility	
Abstract	false
Base Classifier	• AttributeValue
Realized Interface	

7.2.24.1 Fields / Attributes

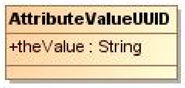
7.2.24.1.1 definition

Type	AttributeDefinitionString
Default Value	
Visibility	public
Multiplicity	

7.2.24.1.2 theValue

Type	String
Default Value	
Visibility	public
Multiplicity	

7.2.25 Class AttributeValueUUID



Name	AttributeValueUUID
Qualified Name	TestIF::Attributes::AttributeValueUUID
Visibility	public
Abstract	false
Base Classifier	•AttributeValue
Realized Interface	

7.2.25.1 Fields / Attributes

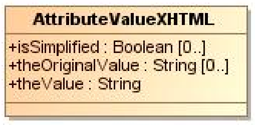
7.2.25.1.1 definition

Type	AttributeDefinitionUUID
Default Value	
Visibility	public
Multiplicity	

7.2.25.1.2 theValue

Type	String
Default Value	
Visibility	public
Multiplicity	

7.2.26 Class AttributeValueXHTML


AttributeValueXHTML
+isSimplified : Boolean [0..]
+theOriginalValue : String [0..]
+theValue : String

Name	AttributeValueXHTML
Qualified Name	TestIF::Attributes::AttributeValueXHTML
Visibility	
Abstract	false
Base Classifier	•AttributeValue
Realized Interface	

7.2.26.1 Fields / Attributes

7.2.26.1.1 definition

Type	AttributeDefinitionXHTML
Default Value	
Visibility	public
Multiplicity	

7.2.26.1.2 isSimplified

This is a flag indicating that the value is simplified from the original XHTML. This is a simplified means that the string in the value has minimal or no XHTML formatting data.

Type	Boolean
Default Value	
Visibility	public
Multiplicity	0..

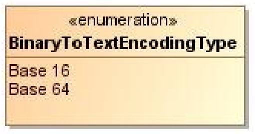
7.2.26.1.3 theOriginalValue

Type	String
Default Value	
Visibility	public
Multiplicity	0..

7.2.26.1.4 theValue

Type	String
Default Value	
Visibility	public
Multiplicity	

7.2.27 Enumeration BinaryToTextEncodingType



Name	BinaryToTextEncodingType
Qualified Name	TestIF::Attributes::BinaryToTextEncodingType
Visibility	public
Base Classifier	

7.3 Package TestIF::Test Classes

The Test Classes package of TestIF provides the meat of the TestIF standard. Figure 7-5 shows the identified classes in the TestIF Specification.

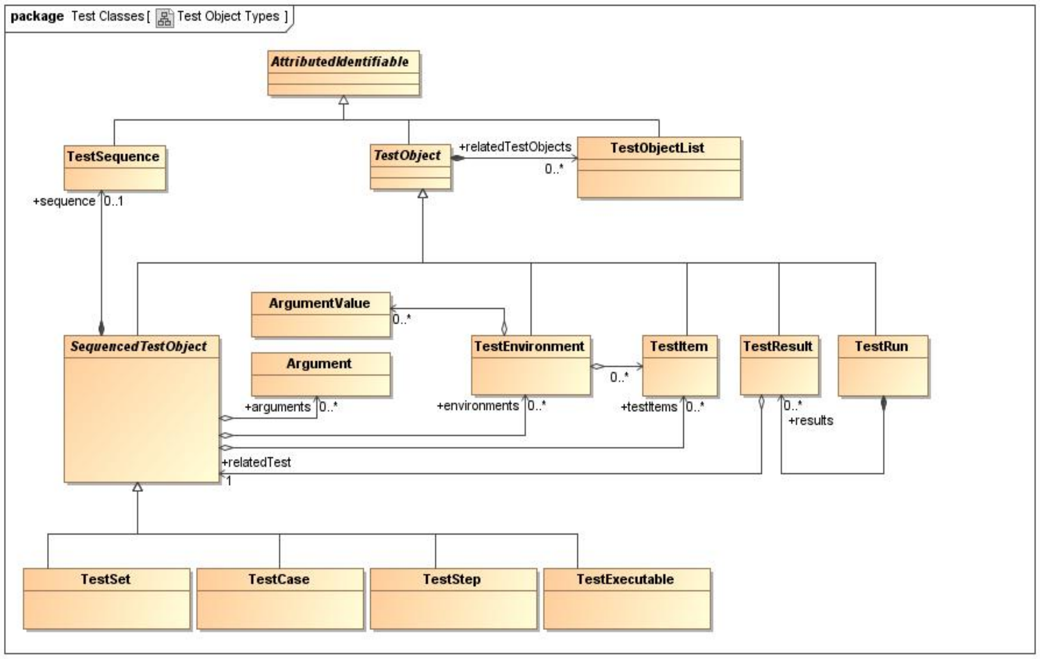


Figure 7-5: Test Object Types

A key concept in TestIF is the ability to relate various data items. There are three main relationship types in TestIF: 1) attributes, 2) sequences, and 3) related objects. Attributes have already been covered earlier in the spec.

Sequences:
Items that are put together to describe a chain of actions to define a test are arranged in Sequences. There are four kinds of SequencedTestObjects in TestIF: TestSet, TestCase, TestStep, and TestExecutable. Figure 7-6 shows the conceptual relationship ~~between~~ among these SequencedTestObject types. While the standard does not structurally enforce these “ownership” rules, ~~it is expected that~~ implementers ~~will~~ are required to follow these semantic levels of hierarchy in their implementations.

Comment [MW26]: Issue 18360

Comment [MW27]: Issue 18345

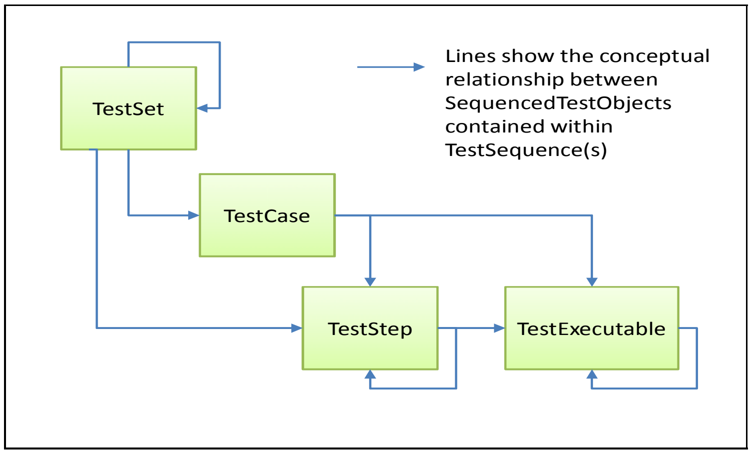

Lines show the conceptual relationship between SequencedTestObjects contained within TestSequence(s)
TestSet
TestCase
TestStep
TestExecutable

Figure 7-6: Conceptual Relationship ~~between~~ among these SequencedTestObject Types

Comment [MW28]: Issue: 18360

SequencedTestObjects contain a TestSequence which in turn contains a series of directed edge graph nodes captured in SequenceStep objects. Each SequenceStep contains a pointer to the related SequencedTestObject that is represented by the test. A SequencedTestObject may define its expected Arguments. Each SequenceStep then "fills in" the expected Arguments with specific ArgumentValues. The ExpectedResult list provides the ability to define the desired outcome from each step.

Figure 7-7 shows the relationships between these various types.

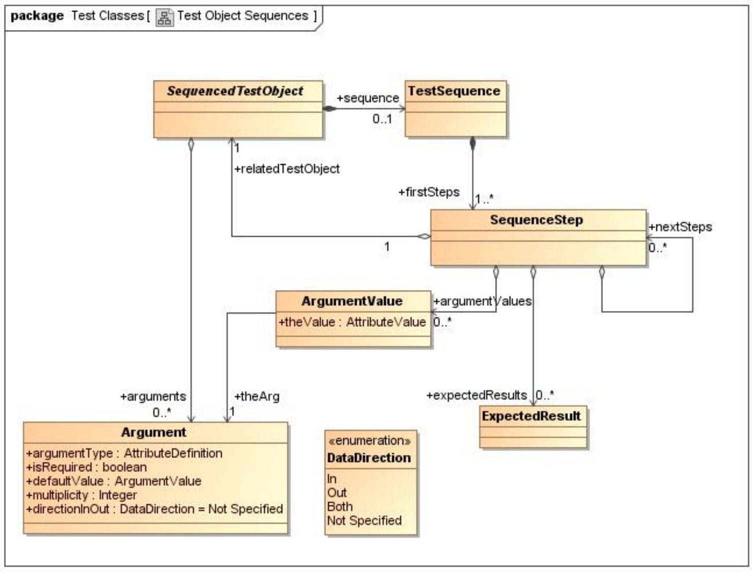


Figure 7-7: Test Object Sequences

Related Objects:
Outside the concept of sequences, TestObjects can be related to any other TestObject. There is any number of reasons why or when this is appropriate for your test definition. TestIF provides several predefined typical relationships (shown in Figure 7-5). TestIF also provides the ability to create TestObjectLists that contain any number of related TestObjects. Note that TestObjectLists are AttributedIdentifiable; therefore, you can name them and provide a description of their purpose. Figure 7-8 shows the basic construct for ~~genrally~~generally related TestObjects.

Comment [MW29]: Issue 18360

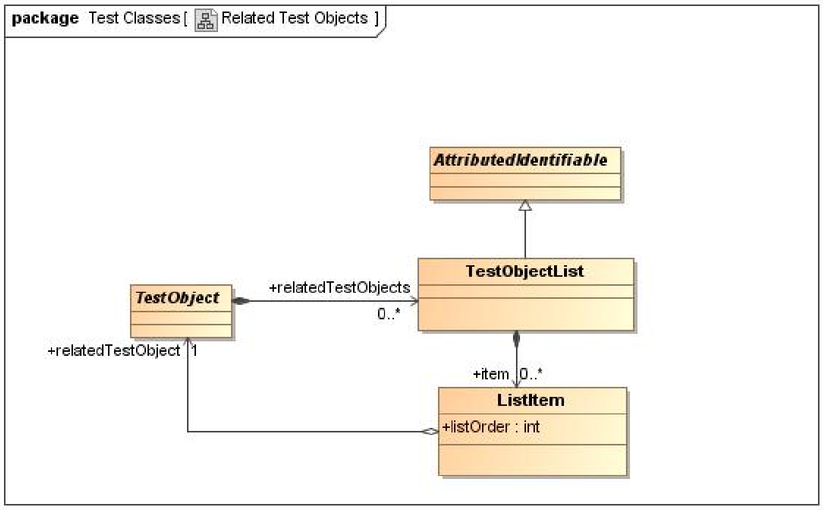


Figure 7-8: Related Test Objects

Once tests have been "executed" their results can be stored in TestRuns. A TestRun contains some basic data describing when the test occurred and which test object was the "starting" node. TestRuns then contain an ordered list of TestResults. Each TestResult contains a pointer back to the SequenceStep or the SequencedTestObject that "created" it.

Figure 7-9 shows the relationships relevant to TestResults.

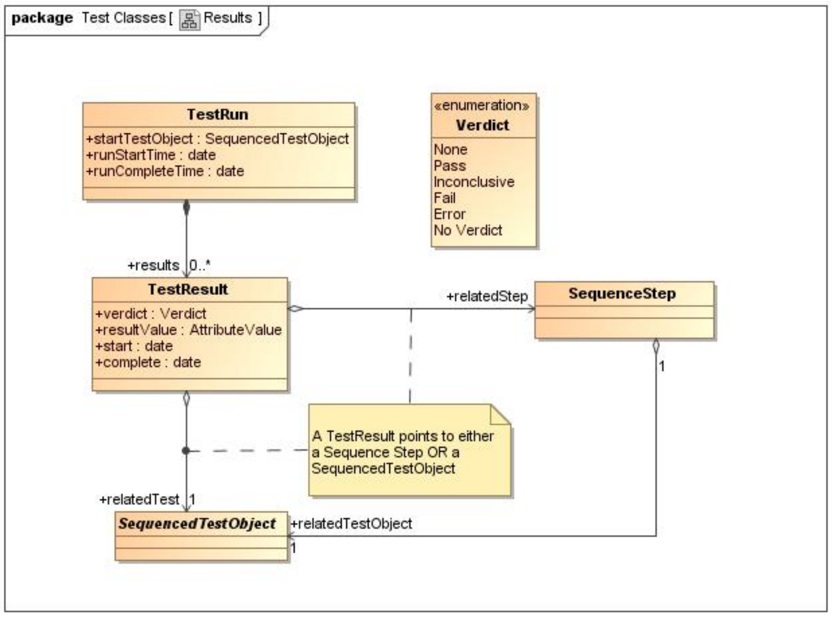


Figure 7-9: Results

7.3.1 Class ExpectedResult

This is the value or outcome that is anticipated to be achieved for a sequence step.

Name	ExpectedResult
Qualified Name	TestIF::Test Classes::ExpectedResult
Visibility	Public
Abstract	False
Base Classifier	•AttributeValueComposite
Realized Interface	

7.3.2 Class ListItem

ListItem
+listOrder : int

Name	ListItem
Qualified Name	TestIF::Test Classes::ListItem
Visibility	Public
Abstract	False
Base Classifier	•AttributedIdentifiable
Realized Interface	

7.3.2.1 Fields / Attributes

7.3.2.1.1 listOrder

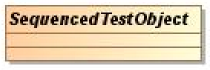
Type	int
Default Value	
Visibility	public
Multiplicity	

7.3.2.1.2 relatedTestObject

Type	TestObject
Default Value	
Visibility	public
Multiplicity	1

7.3.3 Class SequencedTestObject

SequencedTestObject is an abstract TestObject that contains sequential procedural steps. The sequential steps are contained in zero or more related TestSequence objects.



Name	SequencedTestObject
Qualified Name	TestIF::Test Classes::SequencedTestObject
Visibility	Public
Abstract	True
Base Classifier	•TestObject
Realized Interface	

7.3.3.1 Fields / Attributes

7.3.3.1.1 arguments

Type	Argument
Default Value	
Visibility	Public
Multiplicity	0..*

7.3.3.1.2 environments

Type	TestEnvironment
Default Value	
Visibility	Public
Multiplicity	0..*

7.3.3.1.3 sequence

Type	TestSequence
Default Value	
Visibility	Public
Multiplicity	0..1

7.3.3.1.4 testItems

Type	TestItem
Default Value	
Visibility	Public
Multiplicity	0..*

7.3.4 Class SequenceStep

SequenceStep is the directed graph node that is contained within a TestSequence. The sequence step contains a reference to the SequencedTestObject that it is representing in the directed graph. Each SequenceStep may have ArgumentValues that relate to the specific input variables into the related SequencedTestObject. Being an AttributedIdentifiable, each sequence step may also reference other Attributes to provide the additional context as needed. The nextSteps contain reference(s) to the SequenceStep object(s) (directed graph node) that follow this step.

Name	SequenceStep
Qualified Name	TestIF::Test Classes::SequenceStep
Visibility	Public
Abstract	False
Base Classifier	•AttributedIdentifiable
Realized Interface	

7.3.4.1 Fields / Attributes

7.3.4.1.1 argumentValues

Type	ArgumentValue
Default Value	
Visibility	Public
Multiplicity	0..*

7.3.4.1.2 expectedResults

Type	ExpectedResult
Default Value	
Visibility	Public
Multiplicity	0..*

7.3.4.1.3 nextSteps

Type	SequenceStep
Default Value	
Visibility	Public
Multiplicity	0..*

7.3.4.1.4 relatedTestObject

Type	SequencedTestObject
Default Value	
Visibility	public
Multiplicity	1

7.3.5 Class TestCase

This class defines the set of steps necessary to verify something. A Test Case may contain any number of Test Steps, defined in a Test Sequence, to define the series of occurrences to satisfy the test case.

TestCase
+testPurpose : String

Name	TestCase
Qualified Name	TestIF::Test Classes::TestCase
Visibility	public
Abstract	false
Base Classifier	•SequencedTestObject
Realized Interface	

7.3.5.1 Fields / Attributes

7.3.5.1.1 testPurpose

Test Purpose provides a textual description of the reason for a test. This is optional.

Type	String
Default Value	
Visibility	public
Multiplicity	

7.3.6 Class TestEnvironment

Test Environment is a Test Object that describes the system configuration necessary to execute a test. It allows for the definition of system constraints and variables that can be reused by other TestObjects. Test Environments may reference other TestObjects such as Test Items as necessary. As TestEnvironments are “referenced” in the course of a test sequence, their “effect” is assumed to remain in place throughout the rest of the sequence (unless overridden by a subsequent TestEnvironment reference in the sequence).

Name	TestEnvironment
Qualified Name	TestIF::Test Classes::TestEnvironment
Visibility	public
Abstract	False
Base Classifier	•TestObject
Realized Interface	

7.3.6.1 Fields / Attributes

7.3.6.1.1 realtedTestItem

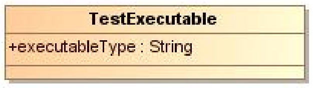
Type	TestItem
Default Value	
Visibility	Private
Multiplicity	0..*

7.3.6.1.2 theValue

Type	ArgumentValue
Default Value	
Visibility	Private
Multiplicity	0..*

7.3.7 Class TestExecutable

Test Executable is similar to Test Step, for use by steps that contain low-level procedural (code/script) step definitions (intended for automation code).



Name	TestExecutable
Qualified Name	TestIF::Test Classes::TestExecutable
Visibility	Public
Abstract	False
Base Classifier	•SequencedTestObject
Realized Interface	

7.3.7.1 Fields / Attributes

7.3.7.1.1 executableType

This is a human readable indicator of the language or format of the associated executable data, such as "java code", VBScript, etc. This is optional.

Type	String
Default Value	
Visibility	Public
Multiplicity	

7.3.8 Class TestItem

Test Item represents an entity in the scope of the test. You can use Test Item to describe the System Under Test (SUT) and any necessary related objects in the system. Given the compositional capabilities of TestObjects in TestIF, TestItems can be composed of multiple “smaller” TestItems items as necessary to fully describe the SUT.

Name	TestItem
Qualified Name	TestIF::Test Classes::TestItem
Visibility	Public
Abstract	False
Base Classifier	•TestObject
Realized Interface	

7.3.9 Class TestObject

TestObject is the base class for the defined types of test classes in TestIF.

Name	TestObject
Qualified Name	TestIF::Test Classes::TestObject
Visibility	Public
Abstract	True
Base Classifier	•AttributedIdentifiable
Realized Interface	

7.3.9.1 Fields / Attributes

7.3.9.1.1 relatedTestObjects

Type	TestObjectList
Default Value	
Visibility	Public
Multiplicity	0..*

7.3.10 Class TestObjects

Name	TestObjects
Qualified Name	TestIF::Test Classes::TestObjects
Visibility	Public
Abstract	False
Base Classifier	
Realized Interface	

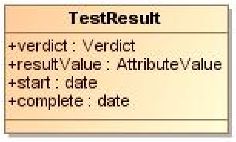
7.3.10.1 Fields / Attributes

7.3.10.1.1 testObjects

Type	TestObject
Default Value	
Visibility	Public
Multiplicity	0..*

7.3.11 Class TestResult

Test Results provide an ordered list of the outcomes from executing a test. In addition to any result data, Test Results contain references back to the original Test Objects for which each result was created. These results can be used to support arbitration and verdict declarations; however those issues are outside the scope of TestIF. Note: A value can be attributed to TestResult at any level at any time by any tool - whether by a separate Arbiter, or at run-time by automation script, etc.



Name	TestResult
Qualified Name	TestIF::Test Classes::TestResult
Visibility	Public
Abstract	False
Base Classifier	•TestObject
Realized Interface	

7.3.11.1 Fields / Attributes

7.3.11.1.1 complete

Optional

Type	Date
Default Value	
Visibility	Public
Multiplicity	

7.3.11.1.2 relatedStep

Type	SequenceStep
Default Value	
Visibility	Public
Multiplicity	

7.3.11.1.3 relatedTest

Type	SequencedTestObject
Default Value	
Visibility	Public
Multiplicity	1

7.3.11.1.4 resultValue

Type	AttributeValue
Default Value	
Visibility	Public
Multiplicity	

7.3.11.1.5 start

Optional

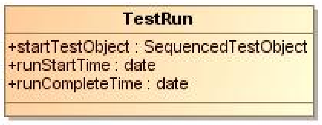
Type	Date
Default Value	
Visibility	Public
Multiplicity	

7.3.11.1.6 verdict

Type	Verdict
Default Value	
Visibility	Public
Multiplicity	

7.3.12 Class TestRun

A Test Run is a container for all of the contextual information about when a test has been executed and the results of the run.



Name	TestRun
Qualified Name	TestIF::Test Classes::TestRun
Visibility	Public
Abstract	False
Base Classifier	•TestObject
Realized Interface	

7.3.12.1 Fields / Attributes

7.3.12.1.1 results

Type	TestResult
Default Value	
Visibility	Public
Multiplicity	0..*

7.3.12.1.2 runCompleteTime

Optional

Type	Date
Default Value	
Visibility	Public
Multiplicity	

7.3.12.1.3 runStartTime

Date / Time stamp of the start time of the run.

Type	Date
Default Value	
Visibility	Public
Multiplicity	

7.3.12.1.4 startTestObject

This is an optional indicator of the starting point of the test run.

Type	SequencedTestObject
Default Value	
Visibility	Public
Multiplicity	

7.3.13 Class TestRuns

This is a container for all Test Rub items to be exchanged in TestIF.

Name	TestRuns
Qualified Name	TestIF::Test Classes::TestRuns
Visibility	Public
Abstract	False
Base Classifier	
Realized Interface	

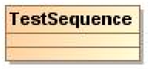
7.3.13.1 Fields / Attributes

7.3.13.1.1 testRuns

Type	TestRun
Default Value	
Visibility	Public
Multiplicity	0..*

7.3.14 Class TestSequence

This class defines the order and progression of execution for a series of Sequenced Test Objects. Each Test Sequence contains a collection of one or more directed edge-graph nodes. Each node points to a sequenced test object and the next step(s) in the sequence. A test sequence can have multiple first steps and/or next steps, which supports parallel test "execution". Test Sequences are required for defining and directing the execution necessary to "run" tests.



Name	TestSequence
Qualified Name	TestIF::Test Classes::TestSequence
Visibility	Public
Abstract	False
Base Classifier	•AttributedIdentifiable
Realized Interface	

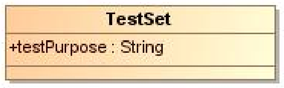
7.3.14.1 Fields / Attributes

7.3.14.1.1 firstSteps

Type	SequenceStep
Default Value	
Visibility	public
Multiplicity	1..*

7.3.15 Class TestSet

TestSet is a SequencedTestObject that can serve as a container for multiple Test Sets, Test Cases, and Test Sets and related test objects such as Test Environment.



Name	TestSet
Qualified Name	TestIF::Test Classes::TestSet
Visibility	public
Abstract	false
Base Classifier	•SequencedTestObject
Realized Interface	

7.3.15.1 Fields / Attributes

7.3.15.1.1 testPurpose

Test Purpose provides a textual description of the reason for a test. This is optional.

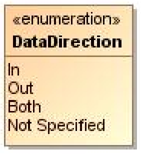
Type	String
Default Value	
Visibility	public
Multiplicity	

7.3.16 Class TestStep

This is an action or validation performed as part of a test sequence. Test Steps are typically contained in Test Cases and/or Test Sets, for execution together, in a specified order (within a Test Sequence). Test Steps can also contain other Test Steps to decompose the steps necessary to capture test execution.

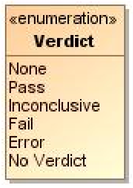
Name	TestStep
Qualified Name	TestIF::Test Classes::TestStep
Visibility	public
Abstract	false
Base Classifier	•SequencedTestObject
Realized Interface	

7.3.17 Enumeration DataDirection



Name	DataDirection
Qualified Name	TestIF::Test Classes::DataDirection
Visibility	public
Base Classifier	

7.3.18 Enumeration Verdict



Name	Verdict
Qualified Name	TestIF::Test Classes::Verdict
Visibility	Public
Base Classifier	

7.4 Predefined Attribute Definitions

7.4.1 Predefined Attributes

Table 7- 1 lists AttributeDefinitions in the org.omg.testIF namespace. The attributes listed below are part of the TestIF standard and therefore TestIF compliant parsers/exporters must support these attributes. The companion document "org_omg_testif_attrbutes.xml" contains the normative XML definitions for the following attributes.

Comment [MW30]: Issue 18326

Table 7-1: AttributeDefinitions in the org.omg.testIF namespace

Identifier (UUID)	Name	Description
org.omg.testIF.testPurpose	Test Purpose	- A human-readable string description of the purpose of a Test Case or Test Set. - No formatting restrictions. - May be attached to Test Set or Test Case.
org.omg.testIF.attributeDefinitionRef	Attribute Definition Reference	- A machine-readable attribute whose value is the identifier of an AttributeDefinition.
org.omg.testIF.optionalFlagDef	Optional	- A machine-readable tagging attribute that indicates that the item it is attached to optional. - No value is required, and any supplied value will be ignored.
org.omg.testIF.dataFlowDirection	Data Flow Direction	- A machine-readable string value that contains one of the valid Data Flow Directions below.

Identifier (UUID)	Name	Description
		- NOT_SPECIFIED - IN - OUT - BOTH
org.omg.testIF.min	Minimum	- A machine-readable numeric attribute that indicates the minimum of the item it is attached to. - A value of -INF indicates that the minimum is unbounded.
org.omg.testIF.max	Maximum	- A machine-readable numeric attribute that indicates the maximum of the item it is attached to. - A value of INF indicates that the maximum is unbounded.
org.omg.testIF.multiplicity	Multiplicity	- A machine-readable composite attribute that indicates the valid number of the items it is attached to. - The first child Attribute is the minimum. - The second child Attribute is the maximum. <AttributeDefinitionRef> org.omg.testIF.min </AttributeDefinitionRef> <AttributeDefinitionRef> org.omg.testIF.max </AttributeDefinitionRef>
org.omg.testIF.argumentDefinition	argumentDefiniti on	- The longName xml attribute of the AttributeValueComposite implementing this definition is required and is the name of the argument. - The first child Attribute is required and is a reference to the AttributeDefinition that defines the type (eg. org.omg.testIF.stringData, etc.). - All other child attributes are optional and can be presented in any order. - An argument defined by usage of this definition is known to be a required argument if an org.omg.testIF.optionalFlag is not present. - The default Data Flow Direction is IN, if an org.omg.testIF.dataFlowDirection is not present. - The default multiplicity is 1, if an org.omg.testIF.multiplicity is not present - There is no default argument value if an org.omg.testIF.argumentValue is not present. - Additional Child Attributes can be added to further define/constrain the valid values of the argument as needed (eg. org.omg.testIF.min/max, etc.). - May be attached to any SequencedTestObject. <AttributeDefinitionRef> org.omg.testIF.attributeDefinitionRef </AttributeDefinitionRef>
org.omg.testIF.argumentValue	argumentValue	- A tagging composite attribute that identifies the child Attribute set after the first child as being an argument value for a test or some operation. - The first child Attribute is required and is the reference to the argumentDefinition for this argumentValue. - At least one more child Attribute must be contained. - More than one more child Attribute is allowed if multiple Attributes are needed to represent a single argument value correctly. - This attribute was designed to be used to carry argument values for SequenceStep and TestEnvironment, but it may be used anywhere an Attribute needs to be tagged as being an argument value.

Identifier (UUID)	Name	Description
		<AttributeDefinitionRef> org.omg.testIF.argumentDefinition </AttributeDefinitionRef>
org.omg.testIF.name	Name	- A human and machine-readable string value that contains only a name. - Leading and/or trailing white spaces are removed.
org.omg.testIF.mimeType	MIME Type	- A machine-readable string value that contains only a valid MIME Type (IETF: RFC 2045, RFC 2046, RFC 2047, RFC 4288, RFC 4289 and RFC 2049). - Leading and/or trailing white spaces are removed.
org.omg.testIF.externalReference	External Reference	- A machine-readable string value that contains only a reference to an external content (such as an architectural component in UPDM, model element, defect report, or other component). - Leading and/or trailing white spaces are removed. - Not defined past the link to the external content. - Expected to be a publicly available URL or a relative path to an item contained with the distribution of the document that contains the reference.
org.omg.testIF.stringData	String Data	- A machine-readable String value that contains data. - This AttributeDefinition is meant to be used within an AttributeDefinitionComposite which will give it context, provide a description of the contents, and describe the format.
org.omg.testIF.integerData	Integer Data	- A machine-readable Integer value that contains data. - This AttributeDefinition is meant to be used within an AttributeDefinitionComposite which will give it context, provide a description of the contents, and describe the format.
org.omg.testIF.realData	Real Data	- A machine-readable Real value that contains data. - This AttributeDefinition is meant to be used within an AttributeDefinitionComposite which will give it context, provide a description of the contents, and describe the format.
org.omg.testIF.booleanData	Boolean Data	- A machine-readable Boolean value that contains data. - This AttributeDefinition is meant to be used within an AttributeDefinitionComposite which will give it context, provide a description of the contents, and describe the format.
org.omg.testIF. binaryToTextEncodingType	Binary to Text Encoding Type	- A machine-readable string value that contains one of the valid binary to text encodings below. - Base16 (hexadecimal) - Base64
org.omg.testIF. requirementReference	Requirement Reference	- A machine-readable string value that contains only a reference to a requirement (ReqIF or any other requirement source). - Leading and/or trailing white spaces are removed. - Supports the need for traceability from tests to requirements, and for results arbitration at the requirement level. - May be attached to a Test Set, Test Case, or Test Step.
org.omg.testIF.preCondition	Pre-Condition	- A human-readable string description of a condition that must be true in order to execute the test. - No formatting restrictions. - May be attached to a Test Set, Test Case, or Test Step.
org.omg.testIF.PreConditionList	Preconditions	A list of preconditions. <AttributeDefinitionRef> org.omg.testIF.PreCondition </AttributeDefinitionRef>
org.omg.testIF.postCondition	preCondition	A description of a condition that must be true when the test completes.

Comment [MW31]: Issue 18349

Identifier (UUID)	Name	Description
org.omg.testIF.PostConditionList	Postconditions	A list of postconditions. <AttributeDefinitionRef> org.omg.testIF.PostCondition </AttributeDefinitionRef>
org.omg.testIF.countsForScore	Counts for Score	- A machine-readable flag indicating that the item that it is attached to should be used in test result arbitration. - May be attached to a Test Set, Test Case, or Test Step.
org.omg.testIF.criticality	Criticality	- A machine-readable string value that is an Indication of importance, for use in result arbitration. - May be attached to a Test Set, Test Case, or Test Step. - The value must be one of the valid criticality value names below: - Normal: no special criticality imparted - High Importance: indicates high importance to results arbitration - Critical: Typically indicates the entire 'container' fails when the item fails.
org.omg.testIF.lastUpdatedBy	Last Updated by	- The name of the person who last updated the test case. - Machine readable and human readable. - Leading and/or trailing white spaces are removed. - Care should be taken to use the same spelling of an individual's name throughout the usage of this field.
org.omg.testIF.executableType	Executable Type	- The name of the execution system that is expected to process the item that this Attribute is attached to. - Machine readable and human readable. - Leading and/or trailing white spaces are removed. - Care should be taken to use the same spelling of an Executable Type throughout the usage of this field.
org.omg.testIF.expectedResultValue	expectedResult	- A tagging composite attribute that identifies the child Attribute set as being an expected result value for a test or some operation. - At least one child Attribute must be contained in associated Attribute Value. - More than one child Attribute is allowed if multiple Attributes are needed to represent a single expected result value correctly. - This attribute was designed to be used to carry expected result values for SequencedTestObject and/or SequenceStep, but it may be used anywhere an Attribute needs to be tagged as being an expected result value.
org.omg.testIF.resultValue	Result Value	- A tagging composite attribute that identifies the child Attribute set as being a result value from a test or some operation. - At least one child Attribute must be contained in associated Attribute Value. - More than one child Attribute is allowed if multiple Attributes are needed to represent a single result value correctly. - This attribute was designed to be used to carry result values for the TestResult, but it may be used anywhere an Attribute needs to be tagged as being a result value.

Identifier (UUID)	Name	Description
org.omg.testIF.sequenceConstraints.PASS	Sequence Constraint PASS	- A machine-readable tagging Attribute that indicates that the test sequence should only continue to the Sequence Step specified in the value if the Verdict of the current step is PASS. - This attribute exists to be used when a test's flow between sequence steps needs to be explicitly constrained/controlled in the definition. - The value of this attribute must be the identifier of the sequence step to be constrained. - This Attribute is only intended to be used on SequenceSteps.
org.omg.testIF.sequenceConstraints.INCONCLUSIVE	Sequence Constraint INCONCLUSIVE	- A machine-readable tagging Attribute that indicates that the test sequence should only continue to the Sequence Step specified in the value if the Verdict of the current step is INCONCLUSIVE. - This attribute exists to be used when a test's flow between sequence steps needs to be explicitly constrained/controlled in the definition. - The value of this attribute must be the identifier of the sequence step to be constrained. - This Attribute is only intended to be used on SequenceSteps.
org.omg.testIF.sequenceConstraints.FAIL	Sequence Constraint FAIL	- A machine-readable tagging Attribute that indicates that the test sequence should only continue to the Sequence Step specified in the value if the Verdict of the current step is FAIL. - This attribute exists to be used when a test's flow between sequence steps needs to be explicitly constrained/controlled in the definition. - The value of this attribute must be the identifier of the sequence step to be constrained. - This Attribute is only intended to be used on SequenceSteps.
org.omg.testIF.sequenceConstraints.ERROR	Sequence Constraint ERROR	- A machine-readable tagging Attribute that indicates that the test sequence should only continue to the Sequence Step specified in the value if the Verdict of the current step is ERROR. - This attribute exists to be used when a test's flow between sequence steps needs to be explicitly constrained/controlled in the definition. - The value of this attribute must be the identifier of the sequence step to be constrained. - This Attribute is only intended to be used on SequenceSteps.
org.omg.testIF.applicationUnderTestName	Application Under Test Name	- A human and machine-readable string value that contains only the name of application being tested. - Leading and/or trailing white spaces are removed.
org.omg.testIF.applicationUnderTestVersion	Application Under Test Version	- A human and machine-readable string value that contains only the version of application being tested. - Leading and/or trailing white spaces are removed.
org.omg.testIF.createdBy	Created By	- A human and machine-readable string value that contains only the name of the Author of the item this Attribute is attached to. - Leading and/or trailing white spaces are removed. - Care should be taken to use the same spelling of an individual's name throughout the usage of this field. - Multiple Created By Attributes can be added if there are multiple authors.
org.omg.testIF.dateCreated	dateCreated	- The machine readable creation date of the item this Attribute is attached to.

Identifier (UUID)	Name	Description
org.omg.testIF.shortDescription	shortDescription	- A human and machine-readable string value that contains a one line summary/description of the item this Attribute is attached to. - Leading and/or trailing white spaces are removed.
org.omg.testIF.note	Note	- A human readable string value that contains a note about the item this Attribute is attached to. - For example, if attached to a TestSet/Case/Step, any helpful hints or other text that pertains to the test. - Use the notesList Attribute for multiple related notes.
org.omg.testIF.notesList	Notes List	- A human readable list of notes. <AttributeDefinitionRef> org.omg.testIF.note </AttributeDefinitionRef>
org.omg.testIF.errorImpact	Error Impact	- A human and machine-readable enumeration that indicates what action should be taken when an Error is encountered during execution. - The enumeration value PAUSE indicates that the test executor should pause test execution, display the error, and prompt the user for direction. - The enumeration value STOP_SCOPED_SEQUENCE_AND_RETURN_ERROR indicates that the test executor should stop only the sequence that had the error and set the result for the container of the sequence to ERROR. - The enumeration value STOP indicates that the test executor should stop the entire test execution and display the error. - This attribute was designed to be used to carry the Error Impact on the SequenceStep object. <EnumerationValue>CONTINUE</EnumerationValue> <EnumerationValue>PAUSE</EnumerationValue> <EnumerationValue>STOP_SCOPED_SEQUENCE_AND_RETURN_ERROR</EnumerationValue> <EnumerationValue>STOP</EnumerationValue>

7.4.2 Predefined Attribute Values

The following Table 7-2 lists AttributeValues in the org.omg.testIF namespace. The attributes values listed below are part of the TestIF standard and therefore TestIF compliant parsers/exporters must support these attribute values. The values are provided for some of the Attributes Definitions that either require no value or the value set is already well defined. TestIF uses a set of predefined Attribute Definitions and Attribute Value types as part of the normative specification. This same basic pattern can be used in the process of extending the standard.

Comment [MW32]: Issue 18350

Table 7-2: AttributeValues in the org.omg.testIF namespace

UUID	Attribute Definition	Value
org.omg.testIF.stringData.DefinitionRef	org.omg.testIF.attributeDefinitionRef	org.omg.testIF.stringData
org.omg.testIF.integerData.DefinitionRef	org.omg.testIF.attributeDefinitionRef	org.omg.testIF.integerData
org.omg.testIF.realData.DefinitionRef	org.omg.testIF.attributeDefinitionRef	org.omg.testIF.realData
org.omg.testIF.booleanData.DefinitionRef	org.omg.testIF.attributeDefinitionRef	org.omg.testIF.booleanData

org.omg.testIF.optionalFlag	org.omg.testIF.optionalFlagDef	

7.5 Extending the Standard

The following section outlines the normative process for extending the standard to accommodate specific tool features and other needs.

Comment [MW33]: Issue 18346

The TestIF standard covers the basic structures and components of Tests, but it is not intended to directly cover all the potential Test structures or the specifics of various Test tools. Instead the standard provides a framework for expanding on the covered concepts to allow 3rd parties to define extensions in a format that can be shared and understood by TestIF import/export/parse implementers.

The method for TestIF extension is through new AttributeDefinitions. Attributes are the general mechanism for adding semantic meaning or well defined information to almost any TestIF object. Creating a new AttributeDefinition allows the 3rd party to attach their specific semantic meaning to a TestIF object or to add new information to that object in a format that can be parsed by other tools.

When creating new AttributeDefinitions to handle concepts not covered directly by the standard, thought should be given to whether or not the concept that is being defined is a general concept vs. a very tool specific concept. If it is a general concept, then the first step is to search the other TestIF implementors published AttributeDefinitions to make sure the concept has not already been defined. When the concept has already been defined elsewhere it should be reused opposed to being redefined. When the concept has not already been defined it should be defined in such a way that others could reuse it without inheriting vendor/tool specific concepts. For example the “org.omg.testIF.resultValue” attribute definition contains the following text as part of its description: “This attribute was designed to be used to carry result values for the TestResult, but it may be used anywhere an Attribute needs to be tagged as being a result value". Along with the other description lines the semantic meaning for the intended use case is covered, but it’s not restricted to be used only in the originally intended location.

Comment [MW34]: Issue 18346

After creating a set of new AttributeDefinitions they should be published in the TestIF format and made publicly available (ie. on your website) for other TestIF import/export/parse implementers to download. This will allow other groups the ability to build support for your extensions into their tools.

If an AttributeDefinition has been published, then modifications that change its parsing and/or semantic meaning should be avoided at all costs since a change of this type would invalidate existing TestIF files still using the old format. Instead a new AttributeDefinition should be created with a new UUID (e.g. org.omg.testIF.name.V2). This approach will maintain backwards compatibility and allow TestIF import/export/parse implementers to support both concepts in time.

The name/value pairing combined with the ability to create arbitrarily complex nesting via the AttributeDefinitionComposite should allow 3rd parties to create extensions to the TestIF standard for their specific concepts without requiring the TestIF structure to change (ideally).
The requirements for the use of each of the AttributeDefinition's fields from the base class “Identifiable” are specified below:

Comment [MW35]: Issue 18347

identifier

- Identifiers of new Attribute Defnitions must be unique (UUIDs). A suitably specific namespace and type name could work, but when there is any doubt using a standard generated UUID as a prefix is suggested.
- Identifiers beginning with "org.omg.testIF" are reserved for use by this standard.
- Identifiers beginning with “localonly” are understood to be file local identifiers only and can be rewritten by import/export/parse tools to maintain uniqueness as needed.

description

- The Description field must contain the following information if it is relevant to the Attribute being defined:
- Full description of the imparted semantic meaning
- Machine readable and/or human readable

- Data type information
- Formatting information
- Restrictions on which object it may be placed on
- If the AttributeDefinition is of type AttributeDefinitionComposite then the position (if required) and type of all the child elements must be explained here.

dateLastUpdated

- The creation date/time of this AttributeDefinition or the last update to any of its fields.

longName

- A human readable name for this AttributeDefinition.

Examples of Some TestIF defined AttributeDefinitions:

```
<AttributeDefinitionReal
    identifier="org.omg.testIF.max"
    longName="Maximum"
    description="
        - A machine-readable numeric attribute that indicates the
          maximum of the item it is attached to.
        - A value of INF indicates that the maximum is unbounded."
    dateLastUpdated="2012-09-18T16:00:00"/>


<AttributeDefinitionComposite
    identifier="org.omg.testIF.multiplicity"
    longName="Multiplicity"
    description="
        - A machine-readable composite attribute that indicates the
          valid number of the items it is attached to.
        - The first child Attribute is the minimum.
        - The second child Attribute is the maximum."
    dateLastUpdated="2012-09-18T16:00:00">
  <AttributeDefinitionRef> org.omg.testIF.min </AttributeDefinitionRef>
  <AttributeDefinitionRef> org.omg.testIF.max </AttributeDefinitionRef>
</AttributeDefinitionComposite>
```

Examples of 3rd Party AttributeDefinitions:

```
<AttributeDefinitionComposite
    identifier=" com.idt.atrt.ParameterEdge"
    longName=" ATRT Test Manager Parameter Edge "
    description="
            - The definition of an ATRT Test Manager ParameterEdge. The
              ParameterEdge defines a specific named data item that is made
              available by the source SequenceStep after it completes it's
              execution. This value is then made available to the destination
              SequenceStep as a specific named data item.
            - This Attribute is only valid on SequenceSteps in the TestIF. The
              SequenceStep that this attribute is attached to is the source of
              the parameter edge.
            - This Attribute is Machine readable.
            - The first child is the Identifier of the SequenceStep that is the
              destination of the parameter edge.
            - The second child is the name of the source output variable.
            - The third child is the name of the destination input variable.”
    dateLastUpdated="2012-09-18T16:00:00" >
  <AttributeDefinitionReference> org.omg.testIF.stringData  </AttributeDefinitionReference>
  <AttributeDefinitionReference> org.omg.testIF.name </AttributeDefinitionReference>
  <AttributeDefinitionReference> org.omg.testIF.name </AttributeDefinitionReference>
</AttributeDefinitionComposite>


<AttributeDefinitionString
    identifier="com.idt.atrt.Job"
    longName="ATRT Test Manager Job"
    description="
       - A machine readable tagging attibute that when applied to TestSets identifies
         the TestSet as being an ATRT Test Manager Job. This attribute has no value.
       - This attribute is only valid on TestSet objects."
    dateLastUpdated="2012-09-18T16:00:00" />
```

8 XML Platform Specific Model

8.1 Purpose

This was created to allow for interchange of Test information in the widely supported XML format.

8.2 Method of Mapping

The PIM objects/structures are represented directly in the XML. Items that are single valued and directly representable by an XML datatype (xs:string, xs:integer, etc.) are mapped as XML attributes of their containing object. Items that are multivalued and/or complex (ie. not directly representable by an XML data type) are mapped as XML elements of their containing object and if needed their internals are defined in an xs:complexContent XML object.

The inheritance hierarchy defined in the PIM is mapped to the xs:extension method in the XML PSM which maintains the same inheritance as shown in the PIM. See the example below of the IdentifiableType and the AttributedIdentifiableType sub type.

```
 <xs:complexType name="IdentifiableType">
  <xs:sequence>
   <xs:element name="AlternateID" type="xs:string" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="identifier" type="xs:string" use="required" />
  <xs:attribute name="description" type="xs:string" use="optional" />
  <xs:attribute name="dateLastUpdated" type="xs:dateTime" use="required" />
  <xs:attribute name="longName" type="xs:string" use="optional" />
 </xs:complexType>


 <xs:complexType name="AttributedIdentifiableType">
  <xs:complexContent>
   <xs:extension base="IdentifiableType">
```

```
    <xs:sequence>
     <xs:element name="AttributeValueRef" type="xs:string" minOccurs="0"
maxOccurs="unbounded" />
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
```

Those objects that add no new fields to their super types are not represented as separate objects in the XML PSM and show up only as PIM named XML elements in their XML container. See the examples below of the definition of TestStep, TestEnvironment, and TestItem within the TestObjects container.

```
   <xs:element name="TestObjects" minOccurs="0" maxOccurs="1">
    <xs:complexType>
     <xs:all>
      <xs:element name="TestSet" type="TestSetType" minOccurs="0" maxOccurs="unbounded"
/>
      <xs:element name="TestCase" type="TestCaseType" minOccurs="0"
maxOccurs="unbounded" />
      <xs:element name="TestStep" type="SequencedTestObjectType" minOccurs="0"
maxOccurs="unbounded" />
      <xs:element name="TestExecutable" type="TestExecutableType" minOccurs="0"
maxOccurs="unbounded" />
      <xs:element name="TestEnvironment" type="SequencedTestObjectType" minOccurs="0"
maxOccurs="unbounded" />
      <xs:element name="TestItem" type="TestObjectType" minOccurs="0"
maxOccurs="unbounded" />
     </xs:all>
    </xs:complexType>
   </xs:element>
```

The TestIFToolExtensionType ~~exists~~ serves in the XML PSM ~~to be~~ as a container for non-normative and unforeseen items that need to be included in the TestIF Interchance document ~~XML~~ but that could not be handled with the normative ~~standard~~ Attribute based extension mechanism explained earlier. The TestIFToolExtensionType should only be used in those scenarios where attempts to use Attributes were unstatifactory somehow. This should very rare, and in the cases where it must occur an Attribute should be created that links TestIF objects to and explains the semantics/format/usage of the information contained within the TestIFToolExtensionType.

Comment [MW36]: Issue 18351

The entire TestIF xsd is available in a separate file named testIF.xsd.

8.3 Using the XML PSM

This section provides a non-normative set of examples for using the XML PSM.

Comment [MW37]: Issue 18352

8.3.1 Simple Attributes

This example adds a simple string attribute named *ApprovalStatus* with a corresponding value of *Approved* to a test case.

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

```
<TestIF>
  <TestIFHeader testIFVersion="1.0" comment="Example"
    dateCreated="2012-08-02T12:00:00"
    identifier=" 79FA8C28E67D0DAE89099999BA66B9A6" title="Example"
  />
  <TestIFContent>
    <AttributeDefinitions>
      <AttributeDefinitionString identifier="com.mydomain.myproduct.ApprovalStatus"
        longName="Approval Status" description="Test case approval state."
        dateLastUpdated="2012-08-02T12:00:00"
      />
      :
      :
    </AttributeDefinitions>
    <AttributeValues>
      <AttributeValueString definition="com.mydomain.myproduct.ApprovalStatus"
         identifier="53BC2B4235012BC3C67D39C800A4F000" value="Approved"
         longName="" dateLastUpdated="2012-08-02T12:00:00"
      />
    </AttributeValues>
       :
       :
    <TestCase identifier="0F3CBB010A7D4624B3474B6FFDC95E29"
      description="Tests something interesting in the application."
      dateLastUpdated="2012-08-02T12:00:00" longName="Example_Testcase_34" >
      <AttributeValueRef> 53BC2B4235012BC3C67D39C800A4F00 </AttributeValueRef>
       :
       :
    </TestCase>
  </TestIFContent>
</TestIF>
```

8.3.2 Attribute Lists

This example adds an attribute list called Preconditions to a test case.

```
<?xml version="1.0" encoding="utf-8"?>
<TestIF>
  <TestIFHeader testIFVersion="1.0" comment="Example"
    dateCreated="2012-08-02T12:00:00"
    identifier=" 79FA8C28E67D0DAE89099999BA66B9A6" title="Example"
  />
<TestIFContent>
  <AttributeDefinitions>
    <AttributeDefinitionList
      identifier="org.omg.testIF.PreConditionList"
      longName="Preconditions"
      description="A list of preconditions."
      dateLastUpdated="2012-09-18T16:00:00"
      <AttributeDefinitionRef> org.omg.testIF.PreCondition </AttributeDefinitionRef>
    </AttributeDefinitionList>

    <AttributeDefinitionString
      identifier="org.omg.testIF.preCondition"
      longName="Pre-Condition"
      description="
              - A human-readable string description of a condition that must be true in order
                to execute the test.
              - No formatting restrictions.
              - May be attached to a Test Set, Test Case, or Test Step."
      dateLastUpdated="2012-09-18T16:00:00"
      />
    </AttributeDefinitions>


    <AttributeValues>
      <AttributeValueList definition="org.omg.testIF.PreConditionList"
        identifier="0F3CBB010A7D4624B3474B6FFDC95E80" longName="Preconditions"
        dateLastUpdated="2012-08-02T12:00:00">
        <AttributeValueRef> 5012BC3C64F0007D39C8053BC2B4230A </AttributeValueRef>
        <AttributeValueRef> 3C67D39C800A4F00053BC2B4235012BC </AttributeValueRef>
      </AttributeValueList>
       :
       :

      <AttributeValueString definition="org.omg.testIF.preCondition"
        identifier="5012BC3C64F0007D39C8053BC2B4230A"
        value="User must be logged in with administrator privileges."
        longName="" dateLastUpdated="2012-08-02T12:00:00"
      />
      <AttributeValueString definition="org.omg.testIF.preCondition"
        identifier="3C67D39C800A4F00053BC2B4235012BC"
        value="User must have submitted at least one product order."
        longName="" dateLastUpdated="2012-08-02T12:00:00"
      />
    </AttributeValues>

       :
       :
    <TestCase identifier="FDC95E290F3CBB010A7D4624B3474B6F"
      description="Tests something interesting in the application."
      dateLastUpdated="2012-08-02T12:00:00" longName="Example_Testcase_21" >
      <AttributeValueRef>0F3CBB010A7D4624B3474B6FFDC95E80 </AttributeValueRef>
       :
      :
    </TestCase>
  </TestIFContent>
</TestIF>
```

8.3.3 Composite Attributes

This example adds a composite attribute named SignoffInfo to a test case. Signoff info is comprised of 3 attributes, SignoffName, SignoffTitle, and SignoffDate.

```
<?xml version="1.0" encoding="utf-8"?>
<TestIF>
  <TestIFHeader testIFVersion="1.0" comment="Example"
    dateCreated="2012-08-02T12:00:00"
    identifier="28E690997D079FA8CDAE6B9A68999BA6" title="Example"
  />
```

```
<TestIFContent>
  <AttributeDefinitions>

    <AttributeDefinitionComposite identifier="com.mydomain.myproduct.SignoffInfo"
      longName="Signoff" description="Signoff information."
      dateLastUpdated="2012-08-02T12:00:00">
      <AttributeDefinitionRef> com.mydomain.myproduct.SignoffName  </AttributeDefinitionRef>
      <AttributeDefinitionRef> com.mydomain.myproduct.SignoffTitle </AttributeDefinitionRef>
      <AttributeDefinitionRef> com.mydomain.myproduct.SignoffDate  </AttributeDefinitionRef>
      </AttributeDefinitionList>



      <AttributeDefinitionString identifier="com.mydomain.myproduct.SignoffName"
        longName="Signoff Name"
        description="Name of the person signing off."
        dateLastUpdated="2012-08-02T12:00:00"
      />
      <AttributeDefinitionString identifier="com.mydomain.myproduct.SignoffTitle"
        longName="Title"
        description="Title of the person signing off."
        dateLastUpdated="2012-08-02T12:00:00"
      />
      <AttributeDefinitionDate identifier="com.mydomain.myproduct.SignoffName"
        longName="Signoff Date"
        description="Date signed off."
        dateLastUpdated="2012-08-02T12:00:00"
      />
      :
      :

    </AttributeDefinitions>

  <AttributeValues>
    <AttributeValueComposite definition=" com.mydomain.myproduct.SignoffInfo"
      identifier="A7D4624B3474B0F3CBB0106FFDC95E80" longName="Signoff Information"
      dateLastUpdated="2012-08-02T12:00:00">
      <AttributeValueRef> 2B4230A5012BC3C64F0007D39C8053BC </AttributeValueRef>
      <AttributeValueRef> D39C803C670A4F000535012BCBC2B423 </AttributeValueRef>
      <AttributeValueRef> 0A4F0D39C803535012BCBC6700C2BCFF </AttributeValueRef>
      </AttributeValueList>




    <AttributeValueString definition="com.mydomain.myproduct.SignoffName"
      identifier="2B4230A5012BC3C64F0007D39C8053BC"
      value="George Jetson"
      longName="" dateLastUpdated="2012-08-02T12:00:00" />
    <AttributeValueString definition="com.mydomain.myproduct.SignoffTitle"
      identifier="D39C803C670A4F000535012BCBC2B423"
      value="VP, Software Development"
      longName="" dateLastUpdated="2012-08-02T12:00:00" />
    <AttributeValueString definition="com.mydomain.myproduct.SignoffTitle"
      identifier="0A4F0D39C803535012BCBC6700C2BCFF "
      value="2012-07-22"
      longName="" dateLastUpdated="2012-08-02T12:00:00" />
      </AttributeValues>
      <TestCase identifier="474B6F5E290F3CBB010AFDC97D4624B3"
        description="Tests something interesting in the application."
        dateLastUpdated="2012-08-02T12:00:00" longName="Example_Testcase_795" >
        <AttributeValueRef> A7D4624B3474B0F3CBB0106FFDC95E80 </AttributeValueRef>
    </TestCase>
  </TestIFContent>
</TestIF>
```

8.3.4 Test Steps, Simple Sequence

This example illustrates the XML for a simple test case with 3 steps (Login, DoSomething, Logout).

```
<?xml version="1.0" encoding="utf-8"?>
<TestIF>
  <TestIFHeader testIFVersion="1.0" comment="Example"
    dateCreated="2012-08-02T12:00:00"
    identifier="28E690997D079FA8CDAE6B9A68999BA6" title="Example"
  />
  <TestIFContent>
    <TestCase identifier="BC857033178D40D7BC2374ABABCC6034"
      description="Tests the app’s ability to do something intelligent."
      dateLastUpdated="2012-08-02T12:00:00" longName="Example_Testcase_562" >
      <Sequence
        identifier="3DAC417F515941C5ACC9F942D9B75C8D"
        dateLastUpdated="2012-08-02T12:00:00"
        longName="Test steps for testcase 562" >
        <Step
          identifier="4D8EABFA5C9D419CAA3129BC9856A368"
          dateLastUpdated="2012-08-02T12:00:00"
          longName="Login step" >
          <TestObjectRef>3E324D83930F4B64B990DCA63042789E </TestObjectRef>
          <NextStepRef> 400CA6B8E1BF43D79D1D6865E53F08B7 </NextStepRef>
        </Step>
        <Step
          identifier="400CA6B8E1BF43D79D1D6865E53F08B7"
          dateLastUpdated="2012-08-02T12:00:00"
          longName="Processing step" >
          <TestObjectRef> FF5337B60C674151936B81B4AE64B423 </TestObjectRef>
          <NextStepRef> DF929A14628A4DFDA2E755D58D5A84EE </NextStepRef>
        </Step>
        <Step
          identifier="DF929A14628A4DFDA2E755D58D5A84EE"
          dateLastUpdated="2012-08-02T12:00:00"
          longName="Logout step" >
          <TestObjectRef> 160FB16200EF4FE58B95CE5C8B21FD7B </TestObjectRef>
          <NextStepRef> </NextStepRef>
        </Step>
        <FirstStepRef> 4D8EABFA5C9D419CAA3129BC9856A368 </FirstStepRef>
      </Sequence>
    </TestCase>
       :
       :
    <TestStep identifier="3E324D83930F4B64B990DCA63042789E"
```

```
      description="Login to the application" dateLastUpdated="2012-08-02T12:00:00"
      longName="Login" >
    </TestStep>
    <TestStep identifier="FF5337B60C674151936B81B4AE64B423"
      description="Perform some kind of operation using the app"
      dateLastUpdated="2012-08-02T12:00:00" longName="DoSomething" >
    </TestStep>
    <TestStep identifier="160FB16200EF4FE58B95CE5C8B21FD7B"
      description="Log out of the application"
      dateLastUpdated="2012-08-02T12:00:00" longName="Logout" >
    </TestStep>
      :
      :
  </TestIFContent>
</TestIF>
```

8.3.5 Test Steps with Arguments

This example illustrates the XML for a test case using a Login test step with 2 arguments, Username and Password. Arguments to a test step can be considered as analogs to parameters in a subroutine or function call. In TestIF, arguments are implemented using attributes. This allows maximum flexibility, but it comes at the expense of a more complex attribute structure.

1. The *org.omg.testIF.argumentDefinition*, *org.omg.testIF.attributeDefinitionRef*, and *org.omg.testIF.ArgumentValue* definitions are predefined attributes that you use to construct the argument definitions. Thus, the *values* of these attributes become, in effect, the *definitions* of the individual arguments.
2. The values of the attributes above are themselves composite attributes that, at a minimum, reference the data type of the argument. The composite may also contain additional attributes that specify the direction of data flow, multiciplicity, min/max restrictions, etc.
3. These values (which are the argument definitions) are referenced from within the containing object (normally the Test Step definition).
4. The actual value of the argument is a composite attribute defined by *org.omg.testIF.ArgumentValue* that contains two references, one to the composite attribute described in item 3 above and the other to a string attribute value that specifies the actual runtime argument value.

```
<?xml version="1.0" encoding="utf-8"?>
<TestIF>
  <TestIFHeader testIFVersion="1.0" comment="Example"
    dateCreated="2012-08-02T12:00:00"
    identifier="28E690997D0799A68999BA6FA8CDAE6B" title="Example"
  />
  <TestIFContent>
    <AttributeDefinitions>
      <AttributeDefinitionComposite
        identifier="org.omg.testIF.argumentDefinition"
        longName="argumentDefinition"
        description="
              - The longName xml attribute of the AttributeValueComposite implementing this
                definition is required and is the name of the argument.
              - The first child Attribute is required and is a reference to the
                AttributeDefinition that defines the type (e.g. org.omg.testIF.stringData).
              - All other child attributes are optional and can be presented in any order.
              - An argument defined by usage of this definition is known to be a required
                argument if an org.omg.testIF.optionalFlag is not present.
        - The default Data Flow Direction is IN, if an org.omg.testIF.dataFlowDirection
           is not present.
        - The default multiplicity is 1, if an org.omg.testIF.multiplicity is not present
        - There is no default argument value if an org.omg.testIF.argumentValue is not
           present.
        - Additional Child Attributes can be added to further define/constrain the valid
           values of the argument as needed (eg. org.omg.testIF.min/max, etc.).
        - May be attached to any SequencedTestObject."
    dateLastUpdated="2012-09-18T16:00:00"
    <AttributeDefinitionRef>org.omg.testIF.attributeDefinitionRef</AttributeDefinitionRef>
  </AttributeDefinitionComposite>
  <AttributeDefinitionUUIDType
    identifier="org.omg.testIF.attributeDefinitionRef"
    longName="Attribute Definition Reference"
    description="
        - A machine-readable attribute whose value is the identifier of an
          AttributeDefinition."
    dateLastUpdated="2012-09-18T16:00:00"
  />
  <AttributeDefinitionComposite
    identifier="org.omg.testIF.argumentValue"
    longName="argumentValue"
    description="
        - A tagging composite attribute that identifies the child Attribute set after
          the first child as being an argument value for a test or some operation.
        - The first child Attribute is required and is the reference to the
           argumentDefinition for this argumentValue.
        - At least one more child Attribute must be contained.
        - More than one more child Attribute is allowed if multiple Attributes are
          needed to represent a single argument value correctly.
        - This attribute was designed to be used to carry argument values for
           SequenceStep and TestEnvironment, but it may be used anywhere an Attribute
          needs to be tagged as being an argument value. "
    dateLastUpdated="2012-09-18T16:00:00"
    <AttributeDefinitionRef>org.omg.testIF.argumentDefinition</AttributeDefinitionRef>
  </AttributeDefinitionComposite>
</AttributeDefinitions>

<AttributeValues>
  <AttributeValueComposite definition="org.omg.TestIF.argumentDefinition"
    identifier="6626C77BCCAD4E709A07548783E354F3"
    longName="Username"
    dateLastUpdated="2012-08-02T12:00:00" >
    <AttributeValueRef>org.omg.stringData.DefinitionRef</AttributeValueRef>
  </AttributeValueComposite>
  <AttributeValueComposite definition="org.omg.TestIF.argumentValue"
    identifier="50293D3C4EE74735B5F5FE47526C3DF4"
    dateLastUpdated="2012-08-02T12:00:00" >
    <AttributeValueRef>6626C77BCCAD4E709A07548783E354F3</AttributeValueRef>
    <AttributeValueRef>179E0F3083514AC6B02E6F398F345BC8</AttributeValueRef>
  </AttributeValueComposite>
  <AttributeValueString definition="org.omg.TestIF.stringData"
    identifier="179E0F3083514AC6B02E6F398F345BC8"
    dateLastUpdated="2012-08-02T12:00:00"
    value="JetsonG" >
  </AttributeValueString>
```

```xml
      <AttributeValueComposite definition="org.omg.TestIF.argumentDefinition"
        identifier="24F243810741471CB16D85C4B464E8E6"
        longName="Password"
        dateLastUpdated="2012-08-02T12:00:00" >
        <AttributeValueRef>org.omg.stringData.DefinitionRef</AttributeValueRef>
      </AttributeValueComposite>
      <AttributeValueComposite definition="org.omg.TestIF.argumentValue"
        identifier="EE74735B5F5FE50293D3C4475F426C3D"
        dateLastUpdated="2012-08-02T12:00:00" >
        <AttributeValueRef>24F243810741471CB16D85C4B464E8E6</AttributeValueRef>
        <AttributeValueRef>804B8A8E3ACB49539FBD48CC27D078B0</AttributeValueRef>
      </AttributeValueComposite>
      <AttributeValueString definition="org.omg.TestIF.stringData"
        identifier="804B8A8E3ACB49539FBD48CC27D078B0"
        dateLastUpdated="2012-08-02T12:00:00"
        value="password_for_JetsonG" >
      </AttributeValueString>

    </AttributeValues>

    <TestCase identifier="BC857033178D40D7BCBCC60342374ABA"
      description="Tests the app's ability to do something intelligent."
      dateLastUpdated="2012-08-02T12:00:00" longName="Example_Testcase_563" >
      <Sequence  identifier="C9F942D9B75C8D3DAC417F515941C5AC"
        dateLastUpdated="2012-08-02T12:00:00"
        longName="Test steps for testcase 563" >
        <Step  identifier="4D8EAB9856A368FA5C9D419CAA3129BC"
          dateLastUpdated="2012-08-02T12:00:00"
          longName="Login step" >
          <TestObjectRef>3E324D83930F4B64B9789E90DCA63042</TestObjectRef>
          <NextStepRef>400CA6B8E1BF43D79D1D6865E53F08B7</NextStepRef>
          <AttributeValueRef>50293D3C4EE74735B5F5FE47526C3DF4</AttributeValueRef>
          <AttributeValueRef>EE74735B5F5FE50293D3C4475F426C3D</AttributeValueRef>
      </Step>
        :
        :
        <FirstStepRef>4D8EAB9856A368FA5C9D419CAA3129BC</FirstStepRef>
      </Sequence>
    </TestCase>
    :
    :
    <TestStep identifier="3E324D83930F4B64B9789E90DCA63042"
      description="Login to the application" dateLastUpdated="2012-08-02T12:00:00"
      longName="Login" >
      <AttributeValueRef>6626C77BCCAD4E709A07548783E354F3</AttributeValueRef>
      <AttributeValueRef>24F243810741471CB16D85C4B464E8E6</AttributeValueRef>
    </TestStep>
    :
    :
  </TestIFContent>
</TestIF>
```

8.3.6 Test Steps, Parallel Processing

This example illustrates the XML for a test case with 3 steps (Login, DoSomething, and DoSomethingSimultaneously, where the last two steps execute in parallel).

```xml
<?xml version="1.0" encoding="utf-8"?>
<TestIF>
  <TestIFHeader testIFVersion="1.0" comment="Example"
    dateCreated="2012-08-02T12:00:00"
    identifier="28E690997D079FA8CDAE6B9A68999BA6" title="Example"
  />
  <TestIFContent>
    <TestCase identifier="BC857033178D40D7BC2374ABABCC6034"
      description="Tests the app’s ability to do something intelligent."
      dateLastUpdated="2012-08-02T12:00:00" longName="Example_Testcase_562" >
      <Sequence
        identifier="3DAC417F515941C5ACC9F942D9B75C8D"
        dateLastUpdated="2012-08-02T12:00:00"
        longName="Test steps for testcase 562" >
        <Step
          identifier="4D8EABFA5C9D419CAA3129BC9856A368"
          dateLastUpdated="2012-08-02T12:00:00"
          longName="Login step" >
          <TestObjectRef>6FAA22772BC84F25B6D28DE4C1E02A82</TestObjectRef>
          <NextStepRef> 397FE9EC742B47FC9FD0A771F6CA0C0B </NextStepRef>
          <NextStepRef> C7FFFAEAA13246A1B3B7374889617A07 </NextStepRef>
        </Step>
        <Step
          identifier="397FE9EC742B47FC9FD0A771F6CA0C0B"
          dateLastUpdated="2012-08-02T12:00:00"
          longName="Processing step" >
          <TestObjectRef> 0E78BFB356E4429D9C82583D24AB8B46 </TestObjectRef>
          <NextStepRef> </NextStepRef>
        </Step>
        <Step
          identifier="C7FFFAEAA13246A1B3B7374889617A07"
          dateLastUpdated="2012-08-02T12:00:00"
          longName="Logout step" >
          <TestObjectRef> 160FB16200EF4FE58B95CE5C8B21FD7B </TestObjectRef>
          <NextStepRef> </NextStepRef>
        </Step>
        <FirstStepRef> 4D8EABFA5C9D419CAA3129BC9856A368 </FirstStepRef>
      </Sequence>
    </TestCase>
    :
    :
    <TestStep identifier="6FAA22772BC84F25B6D28DE4C1E02A82"
       description="Login to the application" dateLastUpdated="2012-08-02T12:00:00"
       longName="Login" >
    </TestStep>
    <TestStep identifier="0E78BFB356E4429D9C82583D24AB8B46"
       description="Perform some kind of operation using the app"
       dateLastUpdated="2012-08-02T12:00:00" longName="DoSomething" >
    </TestStep>
    <TestStep identifier="160FB16200EF4FE58B95CE5C8B21FD7B"
       description="Perform a different operation in parallel "
       dateLastUpdated="2012-08-02T12:00:00" longName="DoSomethingSimultaneously" >
    </TestStep>
    :
    :
  </TestIFContent>
</TestIF>
```

8.3.7 Nested Test Steps

This example shows how to specify a test step that is comprised of other test steps. The standard allows an unlimited number of levels of test step composition.

```
<?xml version="1.0" encoding="utf-8"?>
<TestIF>
  <TestIFHeader testIFVersion="1.0" comment="Example"
    dateCreated="2012-08-02T12:00:00"
    identifier="6C738597932E460E927BED57D4E7E4CD" title="Example"
  />
  <TestIFContent>
    <TestCase identifier="0212ECC5D269425A96EB401DE20220A2"
      description="Tests the app’s ability to do something intelligent."
      dateLastUpdated="2012-08-02T12:00:00" longName="Example_Testcase_754" >
      <Sequence
        identifier="4AF125FC75654003ADE54295FF914B6D"
        dateLastUpdated="2012-08-02T12:00:00"
        longName="Test steps for testcase 754" >
        :
        :
        <Step
          identifier="AA3129BC9856A3684D8EABFA5C9D419C"
          dateLastUpdated="2012-08-02T12:00:00"
          longName="Open File step" >
          <TestObjectRef>FABC9766EAF84A2B8C66CF604C56AE44</TestObjectRef>
          <NextStepRef> <!-- UUID of the next step goes here --> </NextStepRef>
        </Step>
        :
        :
        <FirstStepRef> <!-- UUID of the first step goes here --> </FirstStepRef>
      </Sequence>
    </TestCase>
        :
        :
    <TestStep identifier="FABC9766EAF84A2B8C66CF604C56AE44"
      description="Use the File/Open menu to open a new file"
      dateLastUpdated="2012-08-02T12:00:00"  longName="OpenFile" >
      <Sequence
        identifier="24A8292BCEE243549E6EFAB609010A22"
        dateLastUpdated="2012-08-02T12:00:00"
        longName="Test steps for testcase 754" >
        <Step
          identifier="9EEB34517FD0494F993E5E2288E67073"
          dateLastUpdated="2012-08-02T12:00:00"
          longName="MenuSelect" >
          <TestObjectRef>061C026377C447BFA845DD476CFB0FBA </TestObjectRef>
          <NextStepRef> 006E988919664BF0AD1B4443AC54DACE </NextStepRef>
        </Step>
        <Step
          identifier="51DB80887AA44A30A09A962D749BA2F5"
          dateLastUpdated="2012-08-02T12:00:00"
          longName="WaitForWindow" >
          <TestObjectRef> 006E988919664BF0AD1B4443AC54DACE </TestObjectRef>
          <NextStepRef> 04869419AA75455ABDC8B1ED65AAE066 </NextStepRef>
        </Step>
        <Step
          identifier="650F2B9823A142F38F36C1F3A7817867"
          dateLastUpdated="2012-08-02T12:00:00"
          longName="TextEntry" >
          <TestObjectRef> 04869419AA75455ABDC8B1ED65AAE066 </TestObjectRef>
          <NextStepRef> 17CF12262E0D42048A9A538719FDD917 </NextStepRef>
        </Step>
        <Step
          identifier="79307423D9814C838B20A477F183A315"
          dateLastUpdated="2012-08-02T12:00:00"
          longName="ButtonClick" >
          <TestObjectRef> 17CF12262E0D42048A9A538719FDD917 </TestObjectRef>
        </Step>
        <FirstStepRef> 9EEB34517FD0494F993E5E2288E67073 </NextStepRef>
      </Sequence>
    </TestStep>

    <TestStep identifier="061C026377C447BFA845DD476CFB0FBA"
       description="(sub-step) Select a menu item"
       dateLastUpdated="2012-08-02T12:00:00" longName="MenuSelect" >
    </TestStep>
    <TestStep identifier="006E988919664BF0AD1B4443AC54DACE"
       description="(sub-step) Wait for a dialog box to appear"
       dateLastUpdated="2012-08-02T12:00:00" longName="WaitForWindow" >
    </TestStep>
    <TestStep identifier="04869419AA75455ABDC8B1ED65AAE066"
       description="(sub-step) Enter text in a control"
       dateLastUpdated="2012-08-02T12:00:00" longName="TextEntry" >
    </TestStep>
    <TestStep identifier="17CF12262E0D42048A9A538719FDD917"
       description="(sub-step) Click a button"
       dateLastUpdated="2012-08-02T12:00:00" longName="ButtonClick" >
    </TestStep>
    :
    :
  </TestIFContent>
</TestIF>
```

8.3.8 Test Sets and Test Cases

This example shows how to use a test set element to group 2 test cases.

```
<?xml version="1.0" encoding="utf-8"?>
<TestIF>
  <TestIFHeader testIFVersion="1.0" comment="Example"
    dateCreated="2012-08-02T12:00:00"
    identifier="BC857033178D40D7BC2374ABABCC6034" title="Example"
  />
  <TestIFContent>
    <TestCase identifier="C2374ABABCC6034BC857033178D40D7B"
      description="Tests the app’s ability to do something intelligent."
      dateLastUpdated="2012-08-02T12:00:00" longName="Example_Testcase_562" >
      :
      :
    </TestCase>
    <TestCase identifier="90DCA63042789E3E324D83930F4B64B9"
      description="Tests the app’s ability to do something intelligent."
      dateLastUpdated="2012-08-02T12:00:00" longName="Example_Testcase_565" >
      :
      :
    </TestCase>

    <TestSet identifier="BFA5C9D44D8EA19CC9856A368AA3129B"
      description="Groups Example_Testcase_562 and Example_Testcase_565 together."
      dateLastUpdated="2012-08-02T12:00:00" longName="TestSet_1" >
      <Sequence
        identifier="C5ACC9F942D9B75C8D3DAC417F515941"
        dateLastUpdated="2012-08-02T12:00:00" longName="" >
        <Step
          identifier="4D8EABFA5C9D419CAA3129BC9856A368"
          dateLastUpdated="2012-08-02T12:00:00"
          longName="Login step" >
          <TestObjectRef> C2374ABABCC6034BC857033178D40D7B </TestObjectRef>
          <NextStepRef> 90DCA63042789E3E324D83930F4B64B9 </NextStepRef>
        </Step>
        <Step
          identifier="400CA6B8E1BF43D79D1D6865E53F08B7"
          dateLastUpdated="2012-08-02T12:00:00"
          longName="Processing step" >
          <TestObjectRef> 90DCA63042789E3E324D83930F4B64B9 </TestObjectRef>
        </Step>
        <FirstStepRef> 4D8EABFA5C9D419CAA3129BC9856A368 </FirstStepRef>
      </Sequence>
    </TestSet>
  </TestIFContent>
</TestIF>
```

8.3.9 External References

This example contains 2 references to an external requirements system, and a reference to a datafile stored at a URL.

```
<?xml version="1.0" encoding="utf-8"?>
<TestIF>
  <TestIFHeader testIFVersion="1.0" comment="Example"
    dateCreated="2012-08-02T12:00:00"
    identifier="57C017488E3C4591A736B8C8F457E531" title="Example"
  />
  <TestIFContent>
    <AttributeDefinitions>
      <AttributeDefinitionString
        identifier="org.omg.testIF.requirementReference"
        longName="Requirement Reference"
        description="
        - A machine-readable string value that contains only a reference to a requirement
          (ReqIF or any other requirement source).
        - Leading and/or trailing white spaces are removed.
        - Supports the need for traceability from tests to requirements, and for results
          arbitration at the requirement level.
        - May be attached to a Test Set, Test Case, or Test Step."
        dateLastUpdated ="2012-08-02T12:00:00" />
      <AttributeDefinitionString
        identifier="org.omg.testIF.externalReference"
        longName="External Reference"
        description="
        - A machine-readable string value that contains only a reference to an external content
          (such as an architectural component in UPDM, model element, defect report, or other
          component).
        - Leading and/or trailing white spaces are removed.
        - Not defined past the link to the external content.
        - Expected to be a publicly available URL.
        - May be attached to a Test Set, Test Case, or Test Step."
        dateLastUpdated ="2012-08-02T12:00:00" />
    </AttributeDefinitions>

    <AttributeValues>
      <AttributeValueString
        definition="org.omg.TestIF.requirementReference"
        identifier="E02F4FC094654E9E83066B92E3D899A7"
        value="R.01"
        description="If the user logs in successfully and their account is Active or
          Pending, display the welcome screen."
        dateLastUpdated ="2012-08-02T12:00:00"
        longName="" />
      <AttributeValueString
        definition="org.omg.TestIF.requirementReference"
        identifier="506A241C27584EB5BA08497D667DFDFB"
        value="R.02"
        description="If the user logs in successfully and their account is Active,
          display Message 2 on the welcome screen."
        dateLastUpdated ="2012-08-02T12:00:00" longName="" />
      <AttributeValueString
        definition="org.omg.TestIF.externalReference"
        identifier="3359F518454C48B890796BDD4E169F04"
        value="https://dummycorp.testif.example/lookupsheet1.xls"
        description=""
        dateLastUpdated ="2012-08-02T12:00:00" longName="" />
    </AttributeValues>

    <TestCase identifier="{3747201B-D141-4843-A512-34C802F201B5}"
      description="Tests something interesting in the application."
      dateLastUpdated="2012-08-02T12:00:00" longName="Example_Testcase_601" >
      <AttributeValueRef> E02F4FC094654E9E83066B92E3D899A7 </AttributeValueRef>
      <AttributeValueRef> 506A241C27584EB5BA08497D667DFDFB </AttributeValueRef>
      <AttributeValueRef> 3359F518454C48B890796BDD4E169F04 </AttributeValueRef>
    </TestCase>
  </TestIFContent>
</TestIF>
    ]
```

8.3.10 Environments

This example defines a test environment element containing a variable/value pair that can supply data to a test case.

```
<?xml version="1.0" encoding="utf-8"?>
<TestIF>
  <TestIFHeader testIFVersion="1.0" comment="Example"
    dateCreated="2012-08-02T12:00:00"
    identifier="57C017488E3C4591A736B8C8F457E531" title="Example"
  />
  <TestIFContent>
    <AttributeDefinitions>
      <AttributeDefinitionComposite
        identifier="com.mydomain.myproduct.environmentSetting"
        description="A symbol/value pair used in the setup of the test environment."
        longName="Environment Setting" lastChange="2012-08-02T12:00:00" >
        <AttributeDefinitionRef> org.omg.TestIF.EnvVariable </AttributeDefinitionRef>
        <AttributeDefinitionRef> org.omg.TestIF.EnvValue </AttributeDefinitionRef>
      </AttributeDefinitionComposite>
      <AttributeDefinitionString
        identifier="com.mydomain.myproduct.EnvVariable" longName="Environment variable"
        description="A symbol used in the setup of the test environment."
        lastChange="2012-08-02T12:00:00" />
      <AttributeDefinitionString
        identifier="com.mydomain.myproduct.EnvValue" longName="Environment value"
        description="A value used in the setup of the test environment."
        lastChange="2012-08-02T12:00:00" />
    </AttributeDefinitions>

    <AttributeValues>
      <AttributeValueComposite
        definition="com.mydomain.myproduct.EnvironmentSetting"
        identifier="93971C64EAA24083B75C1DAB6359AA5E"
        lastChange="2012-08-02T12:00:00" >
        <AttributeValueRef> 113AD22711994BE180724C918F303899 </AttributeValueRef>
        <AttributeValueRef> 5AE90B9CE33E4AD7B68508CC6FE28361 </AttributeValueRef>
      </AttributeValueComposite>

      <AttributeValueString
        definition="com.mydomain.myproduct.EnvVariable"
        identifier="113AD22711994BE180724C918F303899"
        value="ENV_FOO" lastChange="2012-08-02T12:00:00"
      />
      <AttributeValueString
        definition=" com.mydomain.myproduct.EnvValue"
        identifier="5AE90B9CE33E4AD7B68508CC6FE28361"
        value="FOO's value" lastChange="2012-08-02T12:00:00"
      />
    </AttributeValues>

    <TestCase identifier="4D4E2A0BB1A5421DA4AE0AADC607CBD3"
      description="Tests something interesting in the application."
      dateLastUpdated="2012-08-02T12:00:00" longName="Example_Testcase_54" >
      <AttributeValueRef> 7649FD57094348E18AFD156C5FE65158 </AttributeValueRef>
      :
      :
    </TestCase>
    <TestEnvironment
      identifier="7649FD57094348E18AFD156C5FE65158"
      description="Test environment for all application tests"
      lastChange="2012-08-02T12:00:00" longName="App configuration" >
      <AttributeValueRef> 93971C64EAA24083B75C1DAB6359AA5E </AttributeValueRef>
    </TestEnvironment>
  </TestIFContent>
</TestIF>
```

9 SQL Platform Specific Model

9.1 Purpose

This was created to allow for storage and interchange of Test information in the widely supported SQL format.

This sub clause provides an overview narrative of the SQL design, relationship diagrams with descriptions and a full set of SQL statements to create a new instance.

In this PSM the Identifier fields are defined as nvarchar(50). This is ~~arbitrary to the standard~~normative to the SQL PSM. Each instance of a TestIF PSM must define Identifier fields sufficiently to contain the UUID format used in the PSM instance.

Comment [MW38]: Issue 18353

9.2 Method of Mapping

Many of the Classes defined by TestIF are expressed by AttributeDefinitions and Attribute Values. This PSM defines the tables necessary to contain and relate the TestIF Class and relations.

The various TestObject Types defined by TestIF are specified as a field in the TestObject table (testObjectType). The list of valid types will be included in the enumerations table with an enumeration type of ObjectType.

Most Test Object Data Fields from the PIM are not explicitly shown in the PSM because they are defined as AttributeDefinitions (simple and composite).

TestIF Arguments and ArgumentValues from the PIM do not need tables in the PSM because Arguments are created using AttributeDefinitionComposite and AttributeValueComposite Classes and corresponding PSM tables.

Some additional tables are created to properly construct lists that link various Identifies. The additional tables are noted below where needed.

~~The entire TestIF SQL schema is available in a separate file named "TestIF SQL PSM.xsl".~~
A SQL creation script is available in a separate file named "TestIF SQL Create Script.sql".

Comment [MW39]: Issue 18354

9.3 Attributes

Attributes are central to all objects in the standard. Any type of Test Object can have a rich collection of attributes. Attributes are used to specify test data, test arguments, environment data and anything else that needs to be associated with a Test Object. Attributes can contain any kind of data or refer to external data sources. Attributes can be grouped into composite attributes that contain other attributes.

Attributes have a definition and any number of values associated with that definition. Test Objects are linked to attribute values through a linking table that allows any number of attribute values to be associated with a Test Object. Attribute values have a unique key that makes any attribute value reusable.

Attributes may be defined at type "composite" which will create a group of unique set of attribute values associated with a single attribute value linked to a Test Object.

The standard defines a composite attribute "Argument". Argument is an attribute that is a set of other attributes.

For example a Test Step may need an argument called style where style defines the characteristics of a block of text. The style values may be Header, Body, and Footer. Each value for the style argument will link to 3 other attributes (Font, Size and Color).

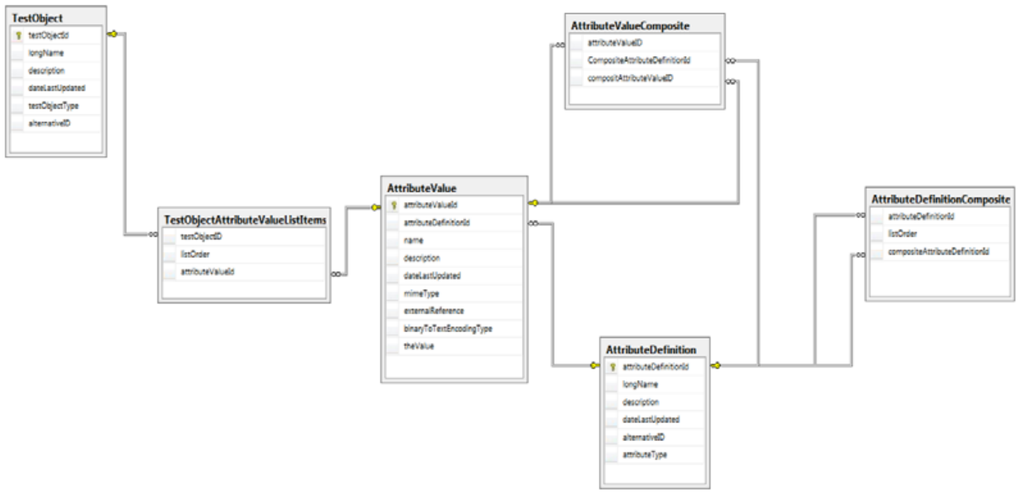


Figure 9-1: Attributes

9.4 Test Sequence

Test Sequence is designed to allow synchronous thread and, therefore, requires multiple links to derive the first/next Test Objects in sequenced steps.

The TestIF Classes support multiple steps to be associated with a Test Step. In the PSM this is accomplished by the addition of a table "TestSequenceList" that is not defined in the PIM. This table allows multiple (list) "SequenceStep" entries to be related to a "TestSequence." This supports (1…n) synchronous steps in a sequenced step.

Arguments are related to Sequenced Steps using a composite attribute. The ArgumentValue is an AttributeValue that relates to an Argument that is a composite type AttributeDefinition. This allows any Sequenced Step to have (0…n) Arguments as related through the ArgumentValueListId table.

A similar relation is created for ExpectedResults through the use of AttributeValues allows any Sequenced Step to have (0…n) ExpectedResults as related through the ExpectedResultIdList table.

Both tables, ArgumentValueListId and ExpectedResultIdList are defined in the PSM but are not defined in the PIM.

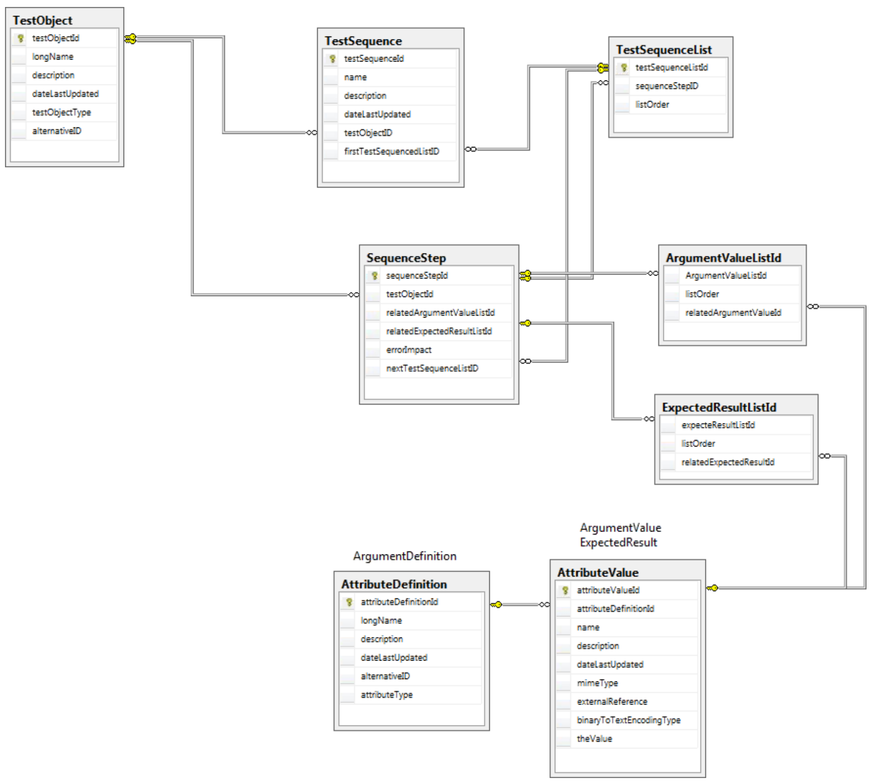


Figure 9-2: Test Sequence

9.5 Related Test Objects

Test Objects (e.g., Test Cases, Test Steps) can be related to other Test Objects to create almost any Test structure required. This design allows self-reference, Test Steps that include other Test Steps, Test Steps that have multiple Executed Steps etc.

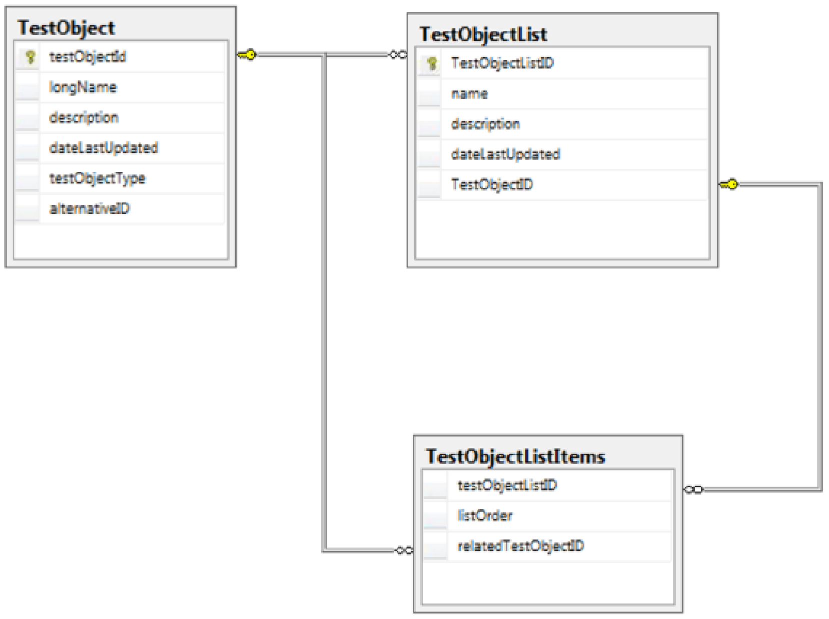


Figure 9-3: Related Test Objects

ANNEX A Example of PSM-Compliant XML

This example shows a longer, somewhat more realistic example of a TestIF interchange XML file. The file contains 2 test cases for 2 different applications, with separate environments. The number of test steps has been minimized for space considerations, but the steps are realistic and have realistic argument lists.

The example file can be found in: *Appendix A – PSM-Compliant Example.xml*

ANNEX B Example of PSM-Compliant XML

This example shows a full XML implementation of the test cases described above in the section called “Example 2 – Requirement Traceability and Results Arbitration for Cause-Effect Model-Generated Tests.”

It is focused on showing how complex requirements traceability and requirement-level results arbitration can be supported by TestIF. Specifically, it shows the following concepts:

- Custom attribute definition to support:
  - Requirement traceability
  - Relationships between test cases
  - Test step input arguments (including complex run-time data pool lookups)
- Test Step reusability across Test Cases
- Simple sequencing (no parallel sequences) of:
  - Test Steps in Test Cases
  - Test Cases in a Test Run
- Test Results of:
  - Sequenced Steps (both for Sequenced Test Steps and Sequenced Test Cases)
  - Non-sequenced Test Cases given Test Results by an external Arbiter

The example can be found in the accompanying file “Appendix B - PSM Compliant Example.xml.”