

IEEE Standard for Automatic Test Markup Language (ATML) Test Descriptions

IEEE Standards Coordination Committee 20

Sponsored by the
IEEE Standards Coordinating Committee 20 (SCC20) on
Test and Diagnosis for Electronic Systems

IEEE
3 Park Avenue
New York, NY 10016-5997
USA

IEEE Std 1671.1™-2017
(Revision of
IEEE Std 1671.1-2009)

IEEE Standard for Automatic Test Markup Language (ATML) Test Description

Sponsor

**IEEE Standards Coordinating Committee 20 (SCC20) on
Test and Diagnosis for Electronic Systems**

Approved 6 December 2017

IEEE-SA Standards Board

Abstract: An exchange format, utilizing Extensible Markup Language (XML), for both the static description of unit under test (UUT), and the specific description of UUT instance information is defined in this standard.

Keywords: ATE, ATML, ATS, ATML instance document, automatic test equipment, Automatic Test Markup Language, automatic test system, IEEE 1671.1™, test program documentation, test requirements, unit under test, UUT, XML schema

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2018 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 19 March 2018. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

PDF: ISBN 978-1-5044-4631-0 STD22963
Print: ISBN 978-1-5044-4632-7 STDPD22963

IEEE prohibits discrimination, harassment, and bullying.

For more information, visit <http://www.ieee.org/web/aboutus/whatis/policies/p9-26.html>.

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page, appear in all standards and may be found under the heading “Important Notice” or “Important Notices and Disclaimers Concerning IEEE Standards Documents.”

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents (standards, recommended practices, and guides), both full-use and trial-use, are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (“IEEE-SA”) Standards Board. IEEE (“the Institute”) develops its standards through a consensus development process, approved by the American National Standards Institute (“ANSI”), which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims all warranties (express, implied and statutory) not included in this or any other document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of material. In addition, IEEE disclaims any and all conditions relating to: results; and workmanlike effort. IEEE standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE should be considered the approved IEEE standard.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to comments or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in revisions to an IEEE standard is welcome to join the relevant IEEE working group.

Comments on standards should be submitted to the following address:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854 USA

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under U.S. and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate fee, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. A current IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE-SA Website at <http://ieeexplore.ieee.org/xpl/standards.jsp> or contact IEEE at the address listed previously. For more information about the IEEE-SA or IEEE's standards development process, visit the IEEE-SA Website at <http://standards.ieee.org>.

Errata

Errata, if any, for all IEEE standards can be accessed on the IEEE-SA Website at the following URL: <http://standards.ieee.org/findstds/errata/index.html>. Users are encouraged to check this URL for errata periodically.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA Website at <http://standards.ieee.org/about/sasb/patcom/patents.html>. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants

At the time this IEEE standard was completed, the 1671.1 Working Group had the following membership:

Anand Jain, *Chair*

Malcom Brown
Chris Gorringer
Teresa Lopes

Ion Neag
Leslie Orlidge
Hugh Pritchett
Mike Seavey

John Stabler
Joseph Stanco
Ronald Taylor

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

W. Larry Adams, Jr
Malcom Brown
Juan Carreon
Chris Gorringer
Randall Groves
Werner Hoelzl
Bernard Homes
Noriyuki Ikeuchi

Anand Jain
Teresa Lopes
Mukund Modi
Ion Neag
Leslie Orlidge
Mike Seavey
Robert Spinner
Joseph Stanco

Michael Stora
Walter Struppler
Marcy Stutzman
Ronald Taylor
Louis Ungar
John Vergis
Oren Yuen
Daidi Zhong

When the IEEE-SA Standards Board approved this standard on 6 December 2017, it had the following membership:

Jean-Philippe Faure, *Chair* **Gary Hoffman, *Vice Chair*** **John D. Kulick, *Past Chair*** **Konstantinos Karachalios, *Secretary***

Chuck Adams
Masayuki Ariyoshi
Ted Burse
Stephen Dukes
Doug Edwards
J. Travis Griffith
Michael Janezic

Thomas Koshy
Joseph L. Koepfinger*
Kevin Lu
Daleep Mohla
Damir Novosel
Ronald C. Petersen
Annette D. Reilly

Robby Robson
Dorothy Stanley
Adrian Stephens
Mehmet Ulema
Phil Wennblom
Howard Wolfman
Yu Yuan

*Member Emeritus

Introduction

This introduction is not part of IEEE Std 1671.1-2017, IEEE Standard for Automatic Test Markup Language (ATML) Test Description.

This child, or “dot,” standard, also known as an ATML component standard, provides for the definition of the test description schemas, and contains references to XML instance document examples; both of which accompany this standard.

The XML schemas defined by this standard provide for the representation of test description information.

Where appropriate, the XML schemas utilize and reference components of the Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML Standard (IEEE Std 1671) schema set. ATML’s XML schemas define the basic information required within any test application and provide a vehicle for formally defining the test environment by defining a class hierarchy corresponding to these basic information entities and provide several methods within each to enable basic operations to be performed on these entities. ATML component standards within the ATML framework define the particular requirements within the test environment.

Contents

1. Overview	10
1.1 Scope	10
1.2 Application	10
1.3 Conventions used within this document	11
2. Normative references.....	13
3. Definitions, acronyms, and abbreviations	13
3.1 Definitions	13
3.2 Acronyms and abbreviations	14
4. Test Description schema.....	16
4.1 Applicability	16
4.2 Root element.....	16
4.3 Security classification.....	17
4.4 Describing the UUT.....	17
5. Describing test program structure.....	20
5.1 Tests.....	20
5.2 Test groups	29
5.3 Using identifiers and identifier references.....	31
6. Describing control flow.....	31
6.1 Entry points	32
6.2 Test sequencing	32
6.3 Initialization and termination.....	35
6.4 Conditions.....	36
6.5 Test group calls.....	38
7. Describing data.....	41
7.1 Overview of data items.....	41
7.2 Overview of data flows.....	43
7.3 Describing data items	47
7.4 Describing data flows	50
8. Describing test behavior.....	58
8.1 Free-form description	59
8.2 Test group call	59
8.3 Operations.....	60
9. Describing fault isolation	73
9.1 UUT faults and failures	73
9.2 Fault trees	75
9.3 Fault dictionaries	75
10. ATML Test Description XML schema names and locations	76
11. ATML XML schema extensibility	78
12. Conformance	79

Annex A (normative) XML schemas	80
A.1 TestDescription XML schema	80
Annex B (informative) TSF library examples	213
B.1 ParallelDigitalTSFLib.xsd	213
B.2 SerialDigitalTSFLib.xsd	218
Annex C (informative) Describing digital testing	222
C.1 Describing digital tests.....	222
C.2 Describing parallel digital signals.....	222
C.3 Describing digital test operations.....	226
C.4 Describing digital tests using DTIF	231
Annex D (informative) Describing serial digital bus exchanges	233
D.1 Describing UUT serial digital buses, protocols, and messages.....	233
D.2 Describing serial bus signals.....	233
D.3 Describing serial bus operations.....	237
D.4 Describing message scheduling and frames.....	247
Annex E (informative) IEEE download web-site material associated with this document	255
Annex F (informative) Users information and examples	257
F.1 Example circuit card assembly UUT	257
F.2 Example 1	257
F.3 Example 2	260
F.4 Example 3	261
F.5 Example 4.....	262
F.6 Example 5.....	263
F.7 Example 6.....	263
F.8 Example for parallel digital tests.....	264
F.9 Examples for RS-232 serial bus	265
F.10 Examples for ARINC 429 serial bus	266
F.11 Examples for MIL-STD-155B serial bus	267
Annex G (informative) Glossary	270
Annex H (informative) Bibliography	271

IEEE Standard for Automatic Test Markup Language (ATML) Test Description

1. Overview

Automatic Test Markup Language (ATML) is a collection of IEEE standards and associated Extensible Markup Language (XML) schemas that allows automatic test system (ATS) and test information to be exchanged in a common format adhering to the XML specifications.¹

The ATML framework and the ATML family of standards have been developed and are maintained under the guidance of the IEEE Standards Coordinating Committee 20 (SCC20) and serve as a comprehensive environment for integrating design data, test strategies, test requirements, test procedures, test results management, and test system implementations, while allowing test program, test asset interoperability, and unit under test (UUT) data to be interchanged between heterogeneous systems.

This standard (as well as the XML schemas and XML instance document examples² that accompany this standard) is intended to be used in identifying and documenting a unit under test (UUT).

1.1 Scope

This standard defines an exchange format, utilizing Extensible Markup Language (XML), for specifying test performance, test conditions, diagnostic requirements, and support equipment to locate, align, and verify the proper operation of a unit under test (UUT). This is in support of the life cycle of test program sets (TPSs) that will be used in an automatic test environment.

1.2 Application

1.2.1 Of this document

This standard provides for test descriptions to be utilized for a variety of purposes, including test program generation, test requirement document (TRD) development and maintenance (MIL-STD-1345B [B10])

¹This information is given for the convenience of users of this standard and does not constitute an endorsement by the IEEE of this consortium standard. Equivalent standards or products may be used if they can be shown to lead to the same results.

²The XML schemas and XML instance document examples that accompany this standard are available at the locations defined in [Annex E](#).

MIL-STD-1519 [B11]) and test description analysis.³ These test descriptions can reference IEEE Std 1641-2010 that describes signals and their behavior.⁴

Current test programs developed to run on ATSs are implemented with tight coupling between components. The components are typically developed specific to a particular architecture. Once the test program is fielded, the requirements and strategies used initially to develop the test program typically become obsolete as the test program evolves in its life cycle. As the ATS is replaced or achieves some level of obsolescence, it is typical to rehost the implementation of the TPS; rehosting is a more expensive, time consuming task than that of implementing the UUT test requirement on the newer ATS or as part of instrument(s) replaced within the existing ATS. This standard will allow a common test description to be transportable across a variety of ATSs used in industries such as automotive, semiconductor, aerospace, military, etc.

The information contained in XML documents conforming to this standard will be useful to the following:

- a) Test program set (TPS) developers
- b) TPS maintainers
- c) Automatic test equipment (ATE) system developers
- d) ATE system maintainers
- e) Developers of ATML-based tools and systems

1.2.2 Of this document's annexes

This document includes five annexes.

[Annex B](#) through [Annex H](#) are informative, and thus are provided strictly as information, for users, implementers, and maintainers of this document.

1.3 Conventions used within this document

1.3.1 General

In accordance with the *IEEE Standards Style Manual*, any schema examples will be shown in `Courier` font. In cases where instance document examples are necessary to depict the use of a schema type or element, such examples will also be shown in `Courier` font. When the characters “...” appear in an example, it indicates that the example component is incomplete.

All simple, complex types attribute groups and elements will be listed; explanatory information will be provided, along with examples if additional clarification is needed. The explanatory information shall include information on the intended use of the elements and/or attributes where the name of the entity does not clearly indicate its intended use. For elements derived from another source type (e.g., an abstract type), only attributes that extend the source type shall be listed; details regarding the base type shall be listed along with the base type.

In tables that describe XML elements, the column “Use” indicates the occurrence constraints for each element.

³The numbers in brackets correspond to those of the bibliography in [Annex H](#).

⁴Information on references can be found in [Clause 2](#).

- a) “required” indicates that the element shall appear exactly once
- b) “optional” indicates that the element may appear once or not at all
- c) “1..∞” indicates that the element shall appear at least once and may appear multiple times
- d) “0..∞” indicates that the element may appear multiple times, once, or not at all

The namespace prefixes “td:” identify that the type or attribute group is contained in an XML schema associated with this document. See [Annex E](#) for a description of where these XML schemas can be located.

When referring to an attribute of an XML element, the convention of *[element]@[attribute]* is used. In cases where an attribute name is referred to with no associated element, the attribute name is enclosed in single quotes. When referencing signal and test definitions conformant with IEEE Std 1641-2010, signal attributes are enclosed in double quotes. Element, attribute, and type names are always be *italicized* when appearing in text.

This standard uses the vocabulary and definitions of relevant IEEE standards. In case of conflict of definitions, except for those portions quoted from standards, the following precedence shall be observed:

- 1) [Clause 3](#)
- 2) [Annex G](#)
- 3) *IEEE Standards Dictionary Online*⁵

1.3.2 Precedence

In the event of conflict between this document and a normatively referenced standard (see [Clause 2](#)), the normatively referenced standard, as it applies to the information being produced, shall take precedence.

The *TestDescription* schema (specified in files “TestDescription.xsd”, “DatumTypes.xsd”, “IeeeStd1641.xsd”, and “Operations.xsd”) element, child element, and annotation information shall take precedence over the descriptive information contained in [Clause 4](#).

The *TestDescription* schema and the material contained in [Clause 4](#) shall take precedence over the example information represented in [Annex F](#).

The *ParallelDigital* schema (specified in file “ParallelDigitalTSFLib.xsd”) element, child element, and annotation information shall take precedence over the example information represented in [Annex F](#).

The *SerialDigital* schema (specified in file “SerialDigitalTSFLib.xsd”) element, child element, and annotation information shall take precedence over the example information represented in [Annex F](#).

For clarity, portions of IEEE Std 1671.3-2017 have been repeated within this standard. The version of IEEE Std 1671.3-2017 referenced in [Clause 2](#) shall take precedence over any IEEE Std 1671.3-2017 material that is repeated within this standard.

For clarity, portions of IEEE Std 1641-2010 have been repeated within this standard. The version of IEEE Std 1641-2010 referenced in [Clause 2](#) shall take precedence over any IEEE Std 1641-2010 material that is repeated within this standard.

⁵IEEE Standards Dictionary Online subscription is available at: <http://dictionary.ieee.org>.

1.3.3 Word usage

In accordance with the *IEEE Standards Style Manual*, the word *shall* is used to indicate mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*). The use of the word *must* is used only to describe unavoidable situations. The use of the word *will* is only used in statements of fact.

The word *should* is used to indicate that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others (*should* equals *is recommended that*).

The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted to*).

The word *can* is used for statements of possibility and capability (*can* equals *is able to*).

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 1671TM-2010, IEEE Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML.⁶

IEEE Std 1671.3TM-2017, IEEE Standard for Standard for Automatic Test Markup Language (ATML) Unit Under Test (UUT) Description.

IEEE Std 1671.5TM-2015, IEEE Standard for Automatic Test Markup Language (ATML) Test Adapter Description.

IEEE Std 1641TM-2010, IEEE Standard for Signal and Test Definition.

IEEE Std 260.1TM, IEEE Standard Letter Symbols for Units of Measurement.

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary Online*⁷ should be consulted for terms not defined in this clause. In the event a term is explicitly redefined, or defined in more detail in an ATML component standard, the component standards definition shall be normative for that ATML component standard.

abstract type: A declared type that can only be used by reference. Thus, in an instance document, any element of an abstract type must also be explicitly declared to be of an `xsi:[type]`.

⁶IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08855, USA (<http://standards.ieee.org>).

⁷*IEEE Standards Dictionary Online subscription* is available at: <http://dictionary.ieee.org>.

Automatic Test Markup Language (ATML) instance document: An **Extensible Markup Language (XML) instance document** that conforms to an **XML schema** defined in an ATML component standard.

Automatic Test Markup Language (ATML) namespace: A collection of names; identified by a uniform resource identifier (URI) reference.

digital rights management: Access control technologies that may be utilized to impose limitations on the usage of digital content material that are not foreseen by the content provider. Within the context of Automatic Test Markup Language (ATML), the digital content materials are the associated XML schemas, and the content provider is the IEEE.

entity: Something that has a distinct separate instance.

eXtensible markup language (XML) attribute: Name-value pair associated with an XML element.

Extensible Markup Language (XML) document: A data object that conforms to the XML requirements for being well formed. In addition, the data object is valid if it additionally conforms to semantic rules of the XML schema.

eXtensible markup language (XML) element: A bounded component of the logical structure of an Extensible Markup Language (XML) document that has a type and that may have XML attributes and content.

Extensible Markup Language (XML) instance document: An **Extensible Markup Language (XML) document** that conforms to a particular **XML schema**.

Extensible Markup Language (XML) schema: The definition of a class of XML documents, which are typically expressed in terms of constraints on the structure and the content of documents of that class, above and beyond the basic syntax constraints imposed by XML itself.

hardware unit under test (UUT): Physical equipment designed for the purpose of performing some function.

software unit under test (UUT): Software programs, procedures, data, and rules designed for the purpose of performing some function.

well formed: Conforming to all of eXtensible Markup Language's (XML's) syntax rules.

3.2 Acronyms and abbreviations

AC	alternating current
ARINC	Aeronautical Radio, Incorporated
ATE	automatic test equipment
ATML	Automatic Test Markup Language
ATPG	Automatic Test Program Generation
ATS	automatic test system
BIT	built-in test

BSC	Basic Signal Component
DC	direct current
DTIF	Digital Test Interchange Format
EQ	equal
FMECA	Failure Mode, Effects and Criticality Analysis
GE	greater or equal
GT	greater than
LE	lower or equal
LT	lower than
NE	not equal
RFI	Receiver Fixture Interface
SCC20	Standards Coordinating Committee 20
STD	Signal and Test Definition
STTO	Safe To Turn On
TPS	test program set
TRD	Test Requirements Document
TSF	Test Signal Framework
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF-8	8-bit Unicode Transformation Format
UUID	universally unique identifier
UUT	unit under test
W3C	World Wide Web Consortium
XML	Extensible Markup Language

4. Test Description schema

4.1 Applicability

The *TestDescription* schema, specified in [Annex A](#) of this document, provides a standard format for the representation of test description information.

4.2 Root element

The root element of the instance document, *TestDescription*, represents the description of test performance, test conditions, diagnostic requirements, and support equipment to locate, align, and verify the proper operation of a UUT. The following information shall be provided in the required attributes of the root element:

- a) Namespaces of all the XML schemas that define the format of the instance document. This information is necessary for schema validation. The following namespaces shall be included:
 - 1) The namespace of the *TestDescription* schema, urn:IEEE-1671.1:2017:TestDescription
 - 2) The namespaces of the *Common* and *HardwareCommon* schemas referenced by the *TestDescription* schema. The *Common* and *HardwareCommon* schemas are defined in Annex B of IEEE Std 1671-2010. Make sure you reference the versions used by the *TestDescription* schema you are referencing from the current instance document.
 - 3) The namespace of the *UUTDescription* schema referenced by the *TestDescription* schema. The *UUTDescription* schema is defined in Annex A of IEEE Std 1671.3-2017. Make sure you reference the version used by the *TestDescription* schema you are referencing from the current instance document. Refer to instance document “IEEE 1671.1 – Example1.xml” for an example.
 - 4) For instance documents that contain fixture (test adapter) descriptions conformant with IEEE Std 1671.5-2015: the namespace of the *TestAdapterDescription* schema. The *TestAdapterDescription* schema is defined in Clause 4 of IEEE Std 1671.5-2015. Refer to instance document “IEEE 1671.1 – Example1.xml” for an example.
 - 5) For instance documents using XML signal or test definitions conformant with IEEE Std 1641-2010: the namespaces of the applicable XML schemas, as defined by IEEE Std 1641-2010. This shall include the *STDBSC* namespace and the schema namespaces of all referenced TSF Libraries. Refer to instance document “IEEE 1671.1 – Example2.xml” for an example.
 - 6) For instance documents using custom extensions: the namespaces of the XML schemas that define those extensions.
- b) ‘*xsi:schemaLocation*’: The location of the XML schema files. This information is necessary for schema validation.⁸
- c) ‘*uuid*’: Universally unique identifier (UUID) for the XML instance document. The UUID may be used for referencing the current XML instance document from other XML instance documents or from software applications. To provide unambiguous identification, a different UUID shall be assigned to each XML instance document.

Refer to B.1.4.2 of IEEE Std 1671-2010 for additional information on root element attributes of ATML documents.

⁸To validate the example XML instance document provided with this standard, the XML schema files must be located in the same directory as the XML instance document.

4.3 Security classification

If the instance document is classified, the root element attribute '*classified*' shall be set to "true". The root element attribute '*securityClassification*' shall specify the classification level, for example, "Secret".

Classification information can be provided in a similar manner for individual tests (through attributes of complex type *Test*) and for individual test groups (through attributes of complex type *TestGroup*).

Refer to B.1.4.1 of IEEE Std 1671-2010 for additional information on classification attributes.

4.4 Describing the UUT

The element *td:TestDescription/td:UUT* shall contain descriptive UUT information required to support the test procedure specified in the current instance document.

4.4.1 IEEE Std 1671.3 UUT description

The element *td:TestDescription/td:UUT/td:Description* shall contain descriptive information for the UUT. This includes the following data categories:

- UUT model identification, including a required model name, a required part or model number, optional version information, and optional manufacturer-specific information
- The list of applicable UUT documentation
- Specification of UUT physical characteristics such as dimensions and weight
- Description of the UUT physical interface, including connectors, pins, ports, and digital buses
- Descriptions of UUT components
- Specification of power, operational, environmental, and calibration requirements for the UUT
- Description of possible UUT faults and failures

The following two alternatives exist for packaging the above information:

- a) Element *td:TestDescription/td:UUT/td:Description/td:DescriptionDocumentReference* shall be used to reference a separate instance document that conforms to IEEE Std 1671.3-2017. The attribute '*uuid*' of element *DescriptionDocumentReference* shall contain the universally unique identifier (UUID) specified in the root element of the referenced document. Refer to instance document "IEEE 1671.1 – Example5.xml" for an example.
- b) Alternatively, element *td:TestDescription/td:UUT/td:Description/td:Definition* shall be used to embed in the current instance document XML content conformant with IEEE Std 1671.3-2017. Refer to instance document "IEEE 1671.1 – Example1.xml" for an example.

Refer to IEEE Std 1671.3-2017 for detailed information on describing the UUT. Refer to "IEEE 1671.1 – Example1.xml" for an example.

4.4.1.1 Describing UUT documentation

The element *td:TestDescription/td:UUT/td:Description/td:Definition/uut:Hardware/hc:Documentation* shall contain references to all UUT-specific documents relevant for the users of the *TestDescription* instance document.

Note that the type of *hc:Document* elements, *c:Document*, has the required attribute 'uuid'. When referencing documents that are not ATML instance documents and not otherwise identified by a UUID, the value of this attribute shall be the null GUID, i.e., {00000000-0000-0000-0000-000000000000}.

Refer to 4.3.2.1 of IEEE Std 1671.3-2017 for detailed information on describing UUT documentation.

4.4.1.2 Describing the UUT interface

The element *td:TestDescription/td:UUT/td:Description/td:Definition/uut:Hardware/hc:Interface* shall describe the electrical interface of the UUT, including connectors, connector pins, and test points.

Refer to 4.3.2.3 of IEEE Std 1671.3-2017 for detailed information on describing UUT connectors and connector pins. Refer to 4.3.2.2 of IEEE Std 1671.3-2017 for detailed information on describing UUT test points.

4.4.1.3 Describing UUT components

The element *td:TestDescription/td:UUT/td:Description/td:Definition/uut:Hardware/hc:Components* shall describe the components of the UUT. The level of detail of this description depends on the use cases to be supported by the instance document, for example:

- To support the identification of faulty components, it is sufficient to uniquely identify each components through a reference designator
- To support the description of tests involving manual probing at component pins, component descriptions must include their interface (i.e., pins)

Refer to 4.3.2.4 of IEEE Std 1671.3-2017 for detailed information on describing UUT components.

4.4.1.4 Referencing UUT and component interface elements

UUT and component interface elements such as ports, connector pins, test points, and component pins are referenced from various *TestDescription* elements using the *hc:Network* type. The *hc:Network/hc:Node/hc:Path* child elements use XPath strings to reference the XML elements corresponding to the referenced ports, connector pins, test points, or component pins. The XPath reference strings shall be constructed as follows:

- a) Start from the root element.
- b) Use prefix namespaces “c:” and “hc:” where appropriate. When an element included in the XPath string can have multiple children, identify the referenced child through an attribute with unique values, such as 'name' or 'ID'. The uniqueness of attributes is enforced in the *TestDescription* schema through *xs:unique* and *xs:key* constraints.

- c) In some cases you may need to use more than one attribute to uniquely identify an element, for example: `c:ConnectorPin[@connectorID="J1" and @pinID="1"]`

NOTE—Many ATML Test Description items reference connector pins, test points and component pins indirectly, through ports, as follows:

- Each `hc:Interface/c:Ports/c:Port` element references one or more pins through the '`pinID`' attribute of their `c:ConnectorPins/c:ConnectorPin` descendants.
- TestDescription elements such as `td:OperationConnect/td:Interface` reference the port using the `hc:Network` type.

For additional information regarding the use of `c:Port`, `c:Connector` and `c:Pin` elements refer to Annex F, "Pins, ports, connectors, and wire lists in ATML", of IEEE Std 1671-2010.

4.4.2 Additional UUT test data

The optional element `td:UUT/TestData` can be used to provide additional UUT data, not supported by IEEE Std 1671.3-2017, that may be of use in developing test requirements and test programs. This includes the following data categories:

- a) Requirements for the AC and DC power supplies used to power the UUT during testing.
- b) Instructions necessary for UUT operation and for personnel and equipment protection during testing, including:
 - 1) Specification of any special precautions and instructions regarding personnel and equipment protection in the presence of hazards such as high voltage, RF radiation, or lasers.
 - 2) Specification of any special handling required by the UUT, for example "Electrostatic Sensitive Device".
 - 3) Specification of any general procedures associated with operating the UUT, for example to adjust and align the UUT.
- c) List of all special electric components and special tools required to test the UUT, for example card extenders.

4.4.2.1 UUT power requirements

The following two elements are provided in the schema for describing power requirements for testing the UUT:

- a) Element `td:TestDescription/td:DetailedTestInformation/td:SignalRequirements` shall be used when signal requirements descriptions conformant with IEEE Std 1641-2010 are mandated.
- b) Element `td:TestDescription/td:UUT/td:TestData/td:PowerRequirements` can be used when IEEE Std 1641-2010 conformance is not necessary. Refer to B.2.2.131 of IEEE Std 1671-2010 for detailed information on using the `hc:PowerSpecifications` type to specify AC and DC power requirements.

NOTE—In addition to the two elements listed above, a third schema element, `td:TestDescription/td:UUT/td:Description/td:Definition/uut:Hardware/hc:PowerRequirements`, supports the specification of power requirements. That element will be used to describe power requirements for normal operation of the UUT. Those could be different from the power requirements applicable to UUT testing.

4.4.2.2 Fixtures, special electric components, and special tools

The fixtures, special electric components and special tools required for testing the UUT shall be described as follows:

- Element *td:TestDescription/td:InterfaceRequirements/td:Fixtures* shall describe the test fixtures and signal conditioning modules. This can include, for example, UUT mounting, holding and support fixtures, cooling carts, pneumatic function generators, hydraulic fixtures, etc.
- Element *td:TestDescription/td:UUT/td:TestData/td:SpecialElectricalComponents* shall identify any special electric components required to test the UUT, such as external resistive loads.
- Element *td:TestDescription/td:UUT/td:TestData/td:SpecialTools* shall identify any special tools required to test the UUT, such as card adapters.

The following two alternatives are provided in the schema for packaging test fixture information:

- a) Element *td:Fixtures/td:Fixture/td:DescriptionDocumentReference* shall be used to reference a separate instance document that conforms to IEEE Std 1671.5-2015 ATML Test Adapter Description. The attribute 'uuid' of element *DescriptionDocumentReference* shall contain the universally unique identifier (UUID) specified in the root element of the referenced document.
- b) Element *td:Fixtures/td:Fixture/td:Definition* shall be used to embed in the current instance document XML content describing a fixture. One of two complex types can be assigned to this element, as follows:
 - 1) The element shall have the default type *c:ItemDescription* if the information supported by this type is sufficient. The type *c:ItemDescription* allows the identification of the fixture through part number and version.
 - 2) Alternatively, the element shall have the type *ta:TestAdapterDescription*, defined in Clause 4 of IEEE Std 1671.5-2015. The type *ta:TestAdapterDescription* supports the complete description of the fixture, including its interfaces, components, internal wiring, etc.

Refer to IEEE Std 1671.5-2015 for detailed information on describing UUT test fixtures.

NOTE—When active fixtures are used, it is often convenient to reference from *TestDescription* the ports of the fixture, or the ports of the fixture's components. You can reference fixture ports from all locations that support referencing of UUT ports by using the *hc:Network* type. Refer to 4.4.1.4 for detailed information on referencing UUT ports.

5. Describing test program structure

The *TestDescription* schema allows the description of a typical test program structure, consisting of tests and test groups.

5.1 Tests

A typical test verifies one characteristic of the UUT by applying a set of stimuli to the UUT, measuring the UUT's responses, and comparing the responses with limits or expected values that define the acceptable UUT operation. When measurement results are within limits or are equal to the expected values, the test

returns a “Passed” outcome. When limits are exceeded, or the expected values are not matched, the test returns a “Failed” outcome and, if applicable, a qualifier indicating which limit was exceeded (e.g., “Low”, “High”). When a test was not completed, it returns an “Aborted” outcome. The qualifier of that outcome may provide additional information regarding the cause of the abort, for example “Manual Intervention”, “Timeout”, “Instrument Error”, etc.

A special class of tests, named “session actions” in the ATML family of standards, perform “non-test” operations, i.e., operations that do not verify UUT characteristics. For example, a session action could apply power to the UUT, control UUT ambient temperature, download firmware in the UUT computer, instruct the ATE operator to perform a manual action, and so on. When successfully completed, session actions return a “Done” outcome. When not completed, they return an “Aborted” outcome. The qualifier of that outcome may provide additional information regarding the cause of the abort.

The ATML *TestDescription* schema represents a test using elements of type *td:Test* located under *td:TestDescription/td:DetailedTestInformation/td:Tests*. The attributes of the *td:Test* type shall be used as follows:

- a) *'ID'*: The attribute shall contain a unique identifier. Refer to 5.3 for additional information on identifiers.
- b) *'name'*: The attribute shall contain a descriptive name. The name is typically used for display and logging purposes.
- c) *'type'*: The value “SessionAction” shall be assigned to this attribute to identify the test as a session action. The other values defined in the *td:TestType* enumeration correspond to other types applicable to tests that are not session actions.
- d) *'documentRequirementID'*: When the *TestDescription* instance document is derived from test requirements data stored in a requirements management system, this attribute can be used to trace each test to the requirement it verifies. The content of the attribute is specific to the requirements management system and could be a numeric identifier, a string identifier, a GUID, and so on. If needed, multiple requirements IDs can be combined using the character ‘;’ as a separator.
- e) *'customType'*: The attribute can be used to identify a category to which the test belongs. This could be for descriptive purposes such as test program documentation and test reporting. Categories can also be used to guide the implementation of the test program; for example, tests from the same category could be implemented in the same function or procedure. This standard does not define specific categories. The definition and use of categories is left to the application.

Note that in many legacy applications tests do not have names. Instead, they are identified by unique test numbers. In this situation the *'name'* attribute can be a string representation of the test number. The *'ID'* can be the same test number, or some other unique identifier.

The elements *td:Test/td:Outcomes/td:Outcome* shall describe all the possible outcomes of the test. The attributes of the *td:Outcome* type shall be used as follows:

- *'ID'*: The attribute shall contain a unique identifier. Refer to 5.3 for additional information on identifiers.
- *'value'*: Shall contain the value of the outcome: “Passed”, “Failed”, “Done”, or “Aborted”.
- *'qualifier'*: When necessary, this attribute can be used to specify additional information that relates to the outcome value. For example, in the case of tests that measure analog values and perform analog comparisons with limits, it can be used to describe distinct outcomes for “Failed High” and “Failed Low”. This information is often used for fault detection and isolation.

This standard does not define specific values for qualifiers. For interoperability, the following value-qualifier combinations should be specified for tests that perform a comparison:

- For comparison with an expected value:
 - 1) value = “Passed”
 - 2) value = “Failed”
- For comparison with an upper limit:
 - 1) value = “Passed”
 - 2) value = “Failed”, qualifier = “High”
- For comparison with a lower limit:
 - 1) value = “Passed”
 - 2) value = “Failed”, qualifier = “Low”
- For comparison with an upper limit and a lower limit:
 - 1) value = “Passed”
 - 2) value = “Failed”, qualifier = “High”
 - 3) value = “Failed”, qualifier = “Low”

The example instance document “IEEE 1671.1 – Example1.xml” contains representative examples of tests and test outcomes. The test with ‘ID’ = “test11” from example instance document “IEEE 1671.1 – Example2.xml” is a typical session action.

5.1.1 Test results

In addition to the outcome, tests often return test results. These can be measured values, calculated values, values received by the test through parameters or variables, and so on.

- Test results should be described, as a minimum, for all the values that are used in comparisons that determine the test outcome. The comparison limits should be included in these test results.
- Additionally, test results can be described for values being used for other purposes such as display, analysis, reporting, and so on.

The elements *td:Test/td:TestResults/td:TestResult* shall describe the test results of the test. The attributes of the *td:TestResult* type shall be used as follows:

- ‘ID’: The attribute shall contain a unique identifier. Refer to 5.3 for additional information on identifiers.
- ‘name’: The attribute shall contain a descriptive name. The name is typically used for display and logging purposes.
- ‘documentRequirementID’: When the *TestDescription* instance document is derived from test requirements data stored in a requirements management system, this attribute can be used to trace each *TestResult* to the requirement it verifies.

The element *td:TestResult/td:ValueDescription* shall be used to describe the values of the *TestResult*. This description indicates the data type, the unit and optionally the nominal value, without specifying an actual value. The actual value is produced at run time, for example through measurement. The child elements of *td:ValueDescription* shall be used to describe different types of data, as follows:

- A single value shall be described using a child element whose type is derived from the *td:DatumDescriptionType* type.
 - 1) The attribute '*xsi:type*' shall indicate the derived type.
 - 2) The *TestDescription* schema defines, in schema file "DatumTypes.xsd", derived types for commonly used numeric data types and for strings.
 - 3) Custom types may be defined by XML instance document producers by deriving from the *td:DatumDescriptionType* type.
- An array of values shall be described using a child element whose type is derived from the *td:IndexedArrayDescriptionType* type.
 - 1) The attribute '*xsi:type*' shall indicate the derived type.
 - 2) The attribute '*dimensions*' can be used to specify the number of dimensions and array bounds (when known in advance).
 - 3) The *TestDescription* schema defines, in schema file "DatumTypes.xsd", derived types for arrays of commonly used numeric data types and for arrays of strings.
 - 4) Custom types may be defined by XML instance document producers.
- A collection of values (i.e., a set of named values) shall be described using a child element of type *td:CollectionDescription*.
 - 1) For each collection element, the *td:ItemDescription* child element shall be used to provide a name and a value description, which may include data type, unit and nominal value.

The common attributes of the above child elements shall be used as follows:

- The unit of the value shall be specified using the '*standardUnit*' attribute (for units compliant with IEEE Std 260.1) or the '*nonStandardUnit*' attribute (for units not defined in IEEE Std 260.1).
 - 1) When necessary, additional information related to the unit can be specified using the '*unitQualifier*' attribute. For example, if an amplitude is measured in mV_{PP} then '*standardUnit*' = "mV" and '*qualifier*' = "pk_pk". This standard does not define specific values for unit qualifiers. For interoperability, the qualifiers defined in B.4.2.2 of IEEE Std 1641-2010 should be used whenever applicable.
 - 2) For arrays, a common unit applicable to all the elements of the array can be specified using the '*standardUnit*', '*nonStandardUnit*' and '*unitQualifier*' attributes. Alternatively, the units of individual array elements can be specified using the *NominalElementValue* child elements of the *IndexedArrayDescriptionType*-derived types.

The following code example illustrates the use of the *td:ValueDescription* type to describe a scalar floating-point *TestResult*:

```
<td:TestResult ID="1" name="Resistance">  
  <td:ValueDescription>
```

```
<td:DatumDescription xsi:type="td:doubleDescription"
  standardUnit="kOhm">
  <td:NominalValue value="13.2"/>
</td:DatumDescription>
</td:ValueDescription>
</td:TestResult>
```

The following code example illustrates the use of the *td:ValueDescription* type to describe a single-dimensional array *TestResult*, with a common unit:

```
<TestResult ID="1" name="Gain">
  <ValueDescription>
    <IndexedArrayDescription xsi:type="doubleArrayDescription"
      standardUnit="dB" dimensions="[5]"/>
  </ValueDescription>
</TestResult>
```

The following code example illustrates the use of the *td:ValueDescription* type to describe a two-dimensional array *TestResult*, with nominal values and element-specific units:

```
<td:TestResult ID="1" name="GainVsFreq">
  <td:ValueDescription>
    <td:IndexedArrayDescription
      xsi:type="td:doubleArrayDescription" standardUnit="dB"
      dimensions="[2,5]"/>
    <td:DefaultElementValue value="0.0"/>
    <td:NominalElementValue position="[1,1]" value="9.0"
      standardUnit="MHz"/>
    <td:NominalElementValue position="[1,2]" value="9.1"
      standardUnit="MHz"/>
    <td:NominalElementValue position="[1,3]" value="9.2"
      standardUnit="MHz"/>
    <td:NominalElementValue position="[1,4]" value="9.3"
      standardUnit="MHz"/>
    <td:NominalElementValue position="[1,5]" value="9.4"
      standardUnit="MHz"/>
    <td:NominalElementValue position="[2,1]" value="0.0"
      standardUnit="dB"/>
    <td:NominalElementValue position="[2,2]" value="0.0"
      standardUnit="dB"/>
    <td:NominalElementValue position="[2,3]" value="0.0"
      standardUnit="dB"/>
    <td:NominalElementValue position="[2,4]" value="0.0"
      standardUnit="dB"/>
    <td:NominalElementValue position="[2,5]" value="0.0"
      standardUnit="dB"/>
  </td:ValueDescription>
</td:TestResult>
```

The following code example illustrates the use of the *td:ValueDescription* type to describe a collection *TestResult* with two floating-point members:

```
<td:TestResult ID="1" name="Coordinates">
  <td:ValueDescription>
    <td:CollectionDescription>
      <td:ItemDescription name="x">
```

```
<td:DatumDescription xsi:type="td:doubleDescription"
  standardUnit="mm"/>
</td:ItemDescription>
<td:ItemDescription name="y">
  <td:DatumDescription xsi:type="td:doubleDescription"
    standardUnit="mm"/>
  </td:ItemDescription>
</td:CollectionDescription>
</td:ValueDescription>
</td:TestResult>
```

The example instance document “IEEE 1671.1 – Example1.xml” contains other representative examples of test results.

5.1.2 Limits

When applicable, the elements *td:TestResult/td:TestLimits/td:Limit* shall describe the expected values or limits associated with the *TestResult*, as follows:

- a) If the test result is compared with one expected value, one limit or one pair of limits, there shall be one *Limit* element and the ‘*operator*’ attribute shall be absent.
- b) If the test result is compared with multiple expected values, or multiple limits, or multiple limit pairs, there shall be one *c:Limit* element for each such value, limit or limit pair. In this situation, the results of individual comparisons are combined using a logical formula, for example: “<test result> is equal to <expected value one> OR <test result> is equal to <expected value two>”. For each *c:Limit*, the ‘*operator*’ attribute shall specify the logical operator that describes its contribution to the logical formula.

The child elements of *c:Limit* shall be used to describe different types of limits and expected values, as follows:

- To describe a single limit, the *SingleLimit* element shall be used. The ‘*comparator*’ attribute of the *SingleLimit* element shall specify the comparison operator to be used when comparing the test result with the limit, as follows:
 - 1) “GE” or “GT” shall be used to describe a lower limit.
 - 2) “LE” or “LT” shall be used to describe an upper limit.
- To describe a limit pair, the *LimitPair* element shall be used. The ‘*comparator*’ attribute of each *Limit* element under *LimitPair* shall describe the comparison operator for that limit. Note that one of the operators must identify an upper limit and the other one a lower limit. In addition, the ‘*operator*’ attribute of *LimitPair* shall describe the logical operator used to combine the results of individual comparisons, as follows:
 - 1) “AND” shall indicate that acceptable values are between limits. In this case, the lower limit shall have ‘*comparator*’ = “GE” or “GT” and the upper limit shall have ‘*comparator*’ = “LE” or “LT”.
 - 2) “OR” shall indicate that acceptable values are outside limits. In this case, the lower limit shall have ‘*comparator*’ = “LE” or “LT” and the upper limit shall have ‘*comparator*’ = “GE” or “GT”.
- To describe an expected value, the *Expected* element shall be used. The ‘*comparator*’ attribute of the *Expected* element shall describe the comparison operator to be used when comparing the test result with the expected value, as follows:

- 1) “EQ” shall indicate that the test result is expected to be equal to the value.
- 2) “NE” shall indicate that the test result is expected to be different from the value.

The example instance document “IEEE 1671.1 – Example1.xml” contains the following representative examples for limits:

- Single limits:
 - 1) Lower Limit: In the *TestResult* with ‘ID’ = “t3tr1” of *Test* with ‘ID’ = “test3”, 40 mVP-P is the lower limit for “V_O AC Voltage”. Note that the “Failed” outcome of the test has the qualifier “Low”.
 - 2) Upper Limit: In the *TestResult* with ‘ID’ = “t2tr1” of *Test* with ‘ID’ = “test2”, 10 mV is the lower limit for “V_O DC Voltage”. Note that the “Failed” outcome of the test has the qualifier “High”.
- Limit pair: In the *TestResult* with ‘ID’ = “t7tr1” of *Test* with ‘ID’ = “test7”, 0.99 V is the lower limit and 1.2 V is the upper limit for “VBO DC Voltage”. Note that the test has two “Failed” outcomes, one with the qualifier “Low” and one with the qualifier “High”.

5.1.3 Parameters

In many cases, tests are reusable within a test description or across multiple test descriptions. To increase the potential for reuse, tests are often parameterized. Parameters determine the behavior of the test. They can be UUT stimulus data, comparison limits, and so on.

The elements *td:Test/td:Parameters/td:Parameter* shall describe the parameters of the test. The attributes of the *td:Parameter* element shall be used as follows:

- a) ‘ID’: The attribute shall contain a unique identifier. Refer to 5.3 for additional information on identifiers.
- b) ‘name’: The attribute shall contain a descriptive name. The name is typically used for display and logging purposes.
- c) ‘documentRequirementID’: When the *TestDescription* instance document is derived from test requirements data stored in a requirements management system, this attribute can be used to trace each parameter to the requirement it verifies.

The element *td:Parameter/td:Value* shall be used to describe the value of the parameter. This description shall specify the data type, the unit, and a value. The child elements of *td:Value* shall be used to describe different types of data, as follows:

- A single value shall be described using a child element whose type is derived from the *c:DatumType* type.
 - 1) The attribute ‘xsi:type’ shall indicate the derived type.
 - 2) The ATML Common schema defines, in schema file “Common.xsd”, derived types for commonly used numeric data types and for strings.
 - 3) Custom types may be defined by XML instance document producers by deriving from the *c:DatumType* type.
- An array of values shall be described using a child element whose type is derived from the *c:IndexedArrayType* type.

- 1) The attribute '*xsi:type*' shall indicate the derived type.
 - 2) The attribute '*dimensions*' shall be used to specify the number of dimensions and array bounds.
 - 3) The ATML Common schema defines, in schema file "Common.xsd", derived types for arrays of commonly used numeric data types and for arrays of strings.
 - 4) Custom types may be defined by XML instance document producers.
- A collection of values (i.e., a set of named values) shall be described using a child element of type *c:Collection*.
- 1) For each collection element, the Item child element shall be used to provide a name and a value.

The common attributes of the above child elements shall be used as follows:

- The unit of the value shall be specified using the '*standardUnit*' attribute (for units compliant with IEEE Std 260.1) or the '*nonStandardUnit*' attribute (for units not defined in IEEE Std 260.1).
- 1) When necessary, additional information related to the unit can be specified using the '*unitQualifier*' attribute. For example, if an amplitude is measured in mV_{PP} then '*standardUnit*' = "mV" and '*qualifier*' = "pk_pk". This standard does not define specific values for unit qualifiers. For interoperability, the qualifiers defined in B.4.2.2 of IEEE Std 1641-2010 should be used whenever applicable.
 - 2) For arrays, a common unit applicable to all the elements of the array can be specified using the '*standardUnit*', '*nonStandardUnit*' and '*unitQualifier*' attributes. Alternatively, the units of individual array elements can be specified using the *NominalElementValue* child elements of the *IndexedArrayDescriptionType*-derived types.

The following code example illustrates the use of the *c:ValueType* type to describe a scalar floating-point parameter:

```
<td:Parameter ID="1" name="VCC">
  <td:Value>
    <c:Datum xsi:type="c:double" standardUnit="V" value="12.0"/>
  </td:Value>
</td:Parameter>
```

The following code example illustrates the use of the *c:ValueType* type to describe a string parameter:

```
<td:Parameter ID="1" name="Command">
  <td:Value>
    <c:Datum xsi:type="c:string">
      <c:Value>IDN?</c:Value>
    </c:Datum>
  </td:Value>
</td:Parameter>
```

The following code example illustrates the use of the *c:ValueType* type to describe parameter that is a single-dimensional array with a common unit:

```
<td:Parameter ID="1" name="Gain">
```

```
<td:Value>
  <c:IndexedArray xsi:type="c:doubleArray" standardUnit="dB"
    dimensions="[5]">
    <c:Element position="[0]" value="1.0"/>
    <c:Element position="[1]" value="1.2"/>
    <c:Element position="[2]" value="1.2"/>
    <c:Element position="[3]" value="1.1"/>
    <c:Element position="[4]" value="0.9"/>
  </c:IndexedArray>
</td:Value>
</td:Parameter>
```

The following code example illustrates the use of the *c:ValueType* type to describe parameter that is a two-dimensional array with element-specific units:

```
<td:Parameter ID="1" name="GainVsFreq">
  <td:Value>
    <c:IndexedArray xsi:type="c:doubleArray" dimensions="[2,5]">
      <c:Element position="[0,1]" value="900.0"
        standardUnit="MHz"/>
      <c:Element position="[1,1]" value="1.0" standardUnit="dB"/>
      <c:Element position="[0,2]" value="910.0"
        standardUnit="MHz"/>
      <c:Element position="[1,2]" value="1.2" standardUnit="dB"/>
      <c:Element position="[0,3]" value="920.0"
        standardUnit="MHz"/>
      <c:Element position="[1,3]" value="1.2" standardUnit="dB"/>
      <c:Element position="[0,4]" value="930.0"
        standardUnit="MHz"/>
      <c:Element position="[1,4]" value="1.1" standardUnit="dB"/>
      <c:Element position="[0,5]" value="940.0"
        standardUnit="MHz"/>
      <c:Element position="[1,5]" value="0.9" standardUnit="dB"/>
    </c:IndexedArray>
  </td:Value>
</td:Parameter>
```

The following code example illustrates the use of the *c:ValueType* type to describe parameter of type collection:

```
<td:Parameter ID="1" name="Coordinates">
  <td:Value>
    <c:Collection>
      <c:Item name="x">
        <c:Datum xsi:type="c:double" standardUnit="mm"
          value="2.55"/>
      </c:Item>
      <c:Item name="y">
        <c:Datum xsi:type="c:double" standardUnit="mm"
          value="3.46"/>
      </c:Item>
    </c:Collection>
  </td:Value>
</td:Parameter>
```

The example instance document “IEEE 1671.1 – Example1.xml” contains other representative examples of parameters.

5.2 Test groups

In the ATML family of standards a “test group” represents a group of related tests performed on the UUT. The criteria used to group tests are not defined by this standard, being left to the application. A test group could, for example, test one UUT subsystem such as a channel, test the entire UUT under specific environmental conditions (for example, at a particular temperature), perform fault isolation on a UUT subsystem, and so on. Test groups can also represent reusable sets of tests.

A typical test group verifies multiple UUT characteristics and infers one outcome. Test groups performing fault detection return a “Passed” outcome or a “Failed” outcome. Test groups performing fault isolation return a “Passed” outcome or one of several “Failed” outcomes, possibly differentiated by their qualifiers. Each such outcome identifies one or more UUT faults that were isolated.

The *TestDescription* schema represents a test using elements of type *td:TestGroup* located under *td:TestDescription/td:DetailedTestInformation/td:TestGroups*. The attributes of the *td:TestGroup* type shall be used as follows:

- a) *‘ID’*: The attribute shall contain a unique identifier. Refer to 5.3 for additional information on identifiers.
- b) *‘name’*: The attribute shall contain a descriptive name. The name is typically used for display and logging purposes.
- c) *‘type’*: The attribute shall identify the purpose of the *TestGroup*. The value “SessionAction” shall not be used, being reserved for *Tests*.
- d) *‘documentRequirementID’*: When the *TestDescription* instance document is derived from test requirements data stored in a requirements management system, this attribute can be used to trace each *TestGroup* to the requirement or requirements it verifies. The content of the attribute is specific to the requirements management system and could be a numeric identifier, a string identifier, a GUID, and so on. If needed, multiple requirements IDs can be combined using the character ‘;’ as a separator.

The elements *td:TestGroup/td:Outcomes/td:Outcome* shall describe all the possible outcomes of the test group. The attributes of the *td:Outcome* type shall be used as follows:

- *‘ID’*: The attribute shall contain a unique identifier. Refer to 5.3 for additional information on identifiers.
- *‘value’*: Shall contain the value of the outcome: “Passed”, “Failed”, “Done”, or “Aborted”.
- *‘qualifier’*: When necessary, this attribute can be used to specify additional information that relates to the outcome value. For example, in the case of test groups that perform fault isolation, it can be used to identify the failed component or components, possibly including the isolated failure modes. The content of the qualifier is not mandated by this standard, being left to the application.

A test group consists of one or more steps, where each step represents the execution of a test. These steps are defined under child elements *td:Sequence*, *td:Serial*, *td:Parallel*, *td:UnspecifiedOrder* or *td:DiagnosticModel*. The child element being used determines the order of execution for steps, in other words, the “control flow” of the test group. Refer to 6.2 for information on describing control flow in test groups.

The example instance document “IEEE 1671.1 – Example1.xml” illustrates a simple test description that contains a single test group performing fault detection and isolation. The outcome qualifiers identify the failed components and failure modes. For example, “Q1.BE.SR” means that transistor Q1 has a short (SR) between Base (B) and Emitter (E). Refer to 9.2 for information on the *td:DetectionIsolation* child elements of *td:Outcome*.

The example instance document “IEEE 1671.1 – Example2.xml” contains a representative example of a test description where test groups contain tests with a similar purpose.

The example instance document “IEEE 1671.1 – Example3.xml” contains a representative example of a test description where a test group is reused. The *TestGroup* with ‘ID’ = “seq2” is called from test with ‘ID’ = “test8” and from test with ‘ID’ = “test9”. These tests are referenced by steps in the *TestGroup* with ‘ID’ = “seq1”.

5.2.1 Test results

In addition to the outcome, test groups can return test results. Unlike the test results of tests, the test results of test groups are not used to determine the test group outcome. They are used for other purposes such as display, analysis, reporting, and so on.

When a test group represents a reusable sequence of tests, its test results can be used to transfer values back to the test that “called” the test group. Refer to 6.5 for detailed information on test groups calls.

The elements *td:TestGroup/td:TestResultDescriptions/td:TestResultDescription* shall describe the test results of the test group. The attributes of the *td:TestResultDescription* element shall be used as follows:

- a) ‘ID’: The attribute shall contain a unique identifier. Refer to 5.3 for additional information on identifiers.
- b) ‘name’: The attribute shall contain a descriptive name. The name is typically used for display and logging purposes.
- c) ‘documentRequirementID’: When the *TestDescription* instance document is derived from test requirements data stored in a requirements management system, this attribute can be used to trace each *TestResult* to the requirement it verifies.

5.2.2 Parameters

To increase the potential for reuse, tests groups can be parameterized. Parameters determine the behavior of the test group. They are typically transferred to test parameters or to variables.

The elements *td:TestGroup/td:ParameterDescriptions/td:ParameterDescription* shall describe the parameters of the test group. The attributes of the *td:ParameterDescription* element shall be used as follows:

- a) ‘ID’: The attribute shall contain a unique identifier. Refer to 5.3 for additional information on identifiers.
- b) ‘name’: The attribute shall contain a descriptive name. The name is typically used for display and logging purposes.

- c) ‘documentRequirementID’: When the *TestDescription* instance document is derived from test requirements data stored in a requirements management system, this attribute can be used to trace each parameter description to the requirement it verifies.

Note that the type *td:ParameterDescription* has a child element of type *td:ValueDescription*, describing the data type, the unit and optionally the nominal value, without specifying an actual value. The actual value is assigned at run time, for example from a parameter or variable of the “calling” test.

5.3 Using identifiers and identifier references

Many elements defined by the *TestDescription* schema contain “identifier” attributes, which enable the referencing of these elements from the “identifier reference” attributes of other elements. For example, the ‘ID’ attributes of *td:Test* elements defined under *td:TestDescription/ td:DetailedTestInformation/td:Tests* are referenced by the ‘testID’ attributes of *td:Step* elements.

To enable proper identification, the “identifier” attributes must have unique values within a defined scope. For instance, the ‘ID’ attributes of *td:Test* elements must be unique for the entire XML instance document. In contrast, the ‘ID’ attributes of *td:Test/td:Parameters/td:Parameter* elements must only be unique within each *td:Test* element. Uniqueness rules for *TestDescription* instance documents are enforced by *xs:unique* and *xs:key* constraints defined in the XML schema, making possible their verification through validation. For example, the uniqueness of Test IDs is enforced by the *xs:key testKey* associated with element *td:TestDescription/td:DetailedTestInformation*.

For referencing, the values of “identifier reference” attributes must match the values of “identifier” attributes defined for the referenced elements. Referential integrity rules for *TestDescription* instance documents are enforced by *xs:keyref* constraints defined in the XML schema, making possible their verification through validation. For example the correct referencing of Test IDs from *td:Step* elements is enforced by the *xs:keyref “testStepRef”* defined within element *td:TestDescription/ td:DetailedTestInformation*.

Note that “identifier” attributes are intended exclusively for referencing XML elements from other XML elements. They should not be used for other purposes, such as display or reporting. When available, the “name” attributes should be used for such purposes.

The creation of *TestDescription* instance documents requires a method for generating unique IDs. For readability, the example instance documents provided for this standard use an ID format that encodes the type of the element and in some cases the ID of the parent element. For example, Tests have IDs “test1”, “test2”, etc. The *Parameters* of *Test* with ‘ID’ = “test2” have IDs “t2p1”, “t2p2”, etc. As XML instance documents are generally intended for electronic processing rather than human interpretation, it is not necessary to implement in applications the above ID format. Instead, IDs can be integer numbers generated in sequence, Globally Unique Identifiers (GUIDs) generated with an appropriate algorithm, etc.

6. Describing control flow

The *TestDescription* schema describes the “control flow” of testing, which includes the following:

- a) The first test group to be performed
- b) The order in which subsequent test groups and tests are performed
- c) “Branching” to different tests and test groups depending on the outcomes of previous tests

6.1 Entry points

In many applications, ATE operators are able to select an “entry point” at the beginning of execution. In some cases, entry points are selected automatically by the run time software.

In the *TestDescription* schema, the entry points are references to test groups. An instance document can specify a single entry point, or multiple entry points. When there are multiple entry points, one of them is designated as “primary”. The primary entry point is used at run time in the absence of an entry point selection (for example, by the ATE operator).

For example, one entry point could reference a test group performing fault detection, while a second entry point could reference a test group performing fault isolation. In another example, each test group performs fault isolation for a specific subsystem of the UUT.

Note that it is the responsibility of the instance document designer to select as entry points only test groups that can be executed standalone (in other words, do not rely on other tests or test groups being executed previously).

The *TestDescription* schema uses */td:TestDescription/td:DetailedTestInformation/td:EntryPoints/td:EntryPoint* elements to describe the entry points. The attribute ‘*testGroupID*’ of each *td:EntryPoint* element references by ID the *TestGroup* identified as entry point. The attribute ‘*primaryEntryPointID*’ of the collector element *td:EntryPoints* references by ID the *TestGroup* identified as primary entry point.

6.2 Test sequencing

In the *TestDescription* schema, tests are not described directly within test groups. Instead, tests descriptions are located outside the test groups; test groups are described as a set of steps, where each step represents the performance of a test. This indirection allows a particular test to be referenced from more than one test group, or multiple times from the same test group, thus facilitating the development of reusable test programs.

The order in which steps are performed can be specified as one of the following:

- a) The steps are performed sequentially.
 - 1) The order of execution is predefined.
 - 2) The order of execution is determined at run time, according to the outcomes returned by the tests, the values returned in test results, or some other criteria.
- b) The steps can be performed in parallel (at the same time), to the extent ATE resources are available.

In the *TestDescription* schema the step order is specified by including under *td:TestGroup* one of the following child elements: *td:Sequence*, *td:Serial*, *td:Parallel*, *td:UnspecifiedOrder*, or *td:DiagnosticModel*.

Within each of the above child elements, the tests to be performed are identified by *td:Step* descendants. The attributes of the *td:Step* element shall be used as follows:

- ‘*ID*’: The attribute shall contain a unique identifier for the step. Refer to 5.3 for additional information on identifiers.
- ‘*testID*’: references by ID the test to be performed in that step.

- *'name'*: This optional attribute can contain a descriptive name. The name is typically used for display and logging purposes.

The child element *td:step/td:Description* allows the addition of detailed, multi-line descriptive information for the step, for example to serve as test program documentation.

6.2.1 Serial

When this element is present the steps are performed sequentially, in the order in which they are listed under the *td:Steps* collector element.

The outcome returned by the test group shall be determined as follows:

- a) If the *td:FaultDictionary* child element is present, then the outcome returned by the test group shall be determined by the rules specified in the fault dictionary. Refer to 9.3 for additional information on fault dictionaries.
- b) If the *td:FaultDictionary* child element is absent, then the outcome returned by the test group shall be determined according to the following rules, evaluated in order:
 - 1) If at least one test returned an “Aborted” outcome, then the outcome of the test group shall be “Aborted”.
 - 2) Otherwise, if at least one test returned a “Failed” outcome, then the outcome of the test group shall be “Failed”.
 - 3) Otherwise, the outcome of the test group shall be “Passed”.

6.2.2 Sequence

When this element is present, the steps are performed sequentially. The order of execution depends on the outcomes returned by the tests according to a predefined “fault tree”.

Under each *td:Step* element, there shall be one *td:Results/td:Result* element for each possible *td:Outcome* descendant of the *td:Test* element referenced by the step. The corresponding *td:Outcome* element shall be identified by ID using the attribute *'testOutcomeID'* of child element *td:TestOutcomeReference*. The action to be taken when the referenced outcome is returned shall be as follows:

- a) If the *td:NextStep* child element is present, the step identified by attribute *td:NextStep@stepID* shall be performed.
- b) If the *td:SequenceOutcome* child element is present, the execution of the test group shall finish and the test group outcome identified by attribute *td:SequenceOutcome@sequenceOutcomeID* shall be returned.
- c) If the *td:Adjust* child element is present, test group execution shall hold until one or more UUT components are adjusted. Once adjustment is completed, test group execution shall continue as follows:
 - 1) If adjustment was successful:
 - i) If element *td:Adjust/td:AfterAdjust/td:Repeat* is present, then the previously performed step shall be performed again.
 - ii) If element *td:Adjust/td:AfterAdjust/td:NextStep* is present, then the step identified by attribute *td:NextStep@stepID* shall be performed.

- iii) If element *td:Adjust/td:AfterAdjust/td:Finish* is present, then the execution of the test group shall finish and the test group outcome identified by attribute *td:Finish@sequenceOutcomeID* shall be returned.
- 2) If adjustment was unsuccessful, for example due to insufficient range:
 - i) If element *td:Adjust/td:OnAdjustFailure/td:NextStep* is present, then the step identified by attribute *td:NextStep@stepID* shall be performed.
 - ii) If element *td:Adjust/td:OnAdjustFailure/td:Finish* is present, then the execution of the test group shall finish and the test group outcome identified by attribute *td:Finish@sequenceOutcomeID* shall be returned.

Note that the *td:NextStep* elements referenced in the above algorithm have optional *td:DetectionIsolation* child elements. Refer to 9.2 for detailed information on fault detection and isolation and the use of these elements.

The example instance document “IEEE 1671.1 – Example1.xml” illustrates the use of a fault tree, described by type *td:Sequence*, to perform fault isolation.

6.2.3 Unspecified order

When this element is present the steps are performed sequentially, in an arbitrary order.

The ordering of steps is not specified by the instance document and can be application-specific. For example, applications could determine at run time the order in which steps are performed according to the availability of ATE resources (instruments, signal paths), to the time it takes to perform each step, to the likelihood of each step achieving fault detection on isolation, and so on.

The termination condition for the test group is not specified by the instance document and can be application-specific. For example, step execution could continue until a sufficient degree of fault isolation is achieved (for example, the ambiguity group size is one). In another application, step execution could end after the first test failure.

The outcome returned by the test group shall be determined by the algorithm described in 6.2.1 for the *td:Serial* element.

6.2.4 Diagnostic model

When this element is present, the steps are performed sequentially, in the order determined at run time by a diagnostic reasoner, in accordance to a diagnostic model. The outcome returned by the test group shall be determined by the diagnostic reasoner, along with any applicable fault detection and isolation results.

The reasoning algorithm and the diagnostic model type are not specified in this standard and can be application-specific. For example, the diagnostic model could be one of the models defined by IEEE Std 1232 [B6]. The document that contains the diagnostic model data shall be referenced using the element *td:TestGroup/td:DiagnosticModel/td:ModelReference*.

6.2.5 Parallel

When this element is present, the steps can be performed in parallel, or simultaneously. Note that the steps are not required to be performed in parallel. In a particular implementation, some steps can be performed simultaneously, while other could be sequential. The parallelization of steps is not specified by this

standard and can be application-specific. For example, an application could determine at run time the steps to be ran in parallel according to the availability of ATE resources (instruments, signal paths).

The termination condition for the test group is not specified by the instance document and can be application-specific. For example, step execution could continue until a sufficient degree of fault isolation is achieved (for example, the ambiguity group size is one). In another application, step execution could end after the first test failure.

The outcome returned by the test group shall be determined by the algorithm described in 6.2.1 for the *td:Serial* element.

6.3 Initialization and termination

In many applications specific actions need to be performed before UUT tests can be executed. These actions could include, for example, instructing the operator to connect to the ATE fixtures, the UUT, and ancillary equipment; verifying that the UUT is properly connected to the ATE; verifying that any applicable ancillary equipment is connected to the ATE and operational; identifying the UUT model; verifying that the UUT does not have faults that could damage the ATE; turning on UUT cooling; turning on UUT power; uploading specialized test firmware to the UUT; and so on.

Similarly, specific actions may need to be performed after UUT tests are executed. These actions could include, for example, uploading operational firmware to the UUT, powering down the UUT, turning off UUT cooling, instructing the operator to disconnect from the ATE the UUT, fixtures and ancillary equipment, displaying instructions and warnings for subsequent UUT disassembly (for example, ESD warnings).

6.3.1 Initialization

In the *TestDescription* schema the actions to be performed before a test group is executed are represented by the *td:TestGroup/td:Initialization* element.

- a) If the actions can be described through tests (including session actions), then the child element *td:Initialization/td:Test* shall be used. Note that the *td:Test* element identified by the 'testID' attribute can be a "regular" test (typically of type "session action"), or could be a call to a test group where multiple tests are executed. Refer to 8.2 for information of test group calls.
- b) Alternatively, the actions to be performed for initialization can be described through free-form text using the *td:Initialization/td:DetailedDescription* child element.

If initialization is performed by a test, then the *td:Initialization/td:Results* element shall be present and there shall be one *td:Result* child element for each possible outcome returned by the referenced test. For each *td:Result*:

- The presence of child element *td:Result/td:Continue* indicates that the execution of the test group shall continue with the first step. This situation corresponds to a successful initialization.
- The presence of child element *td:Result/td:TestGroupOutcome* indicates that the no steps shall be executed and the test group outcome identified by attribute *td:TestGroupOutcome@testGroupOutcomeID* shall be returned for the test group. As this situation corresponds to a failed initialization, a "Failed" or "Aborted" outcome would typically be returned.

The example instance document "PrePostCond.xml" contains a simple example for specifying test group initialization through a test that calls a test group.

6.3.2 Termination

In the *TestDescription* schema the actions to be performed after a test group is executed are represented by the *td:TestGroup/td:Termination* element.

- a) If the actions can be described through tests (including session actions), then the child element *td:Termination/td:Test* shall be used. Note that the *td:Test* element identified by the 'testID' attribute can be a “regular” test (typically of type “session action”), or could be a call to a test group where multiple tests are executed. Refer to 8.2 for information of test group calls.
- b) Alternatively, the actions to be performed for termination can be described through free-form text using the *td:Termination/td:DetailedDescription* child element.

If termination is performed by a test, then the *td:Termination/td:Results* element shall be present and there shall be one *td:Result* child element for each possible outcome returned by the referenced test. For each *td:Result* element:

- The presence of child element *td:Result/td:Continue* indicates that the outcome determined through test group execution will be returned unchanged.
- The presence of child element *td:Result/td:TestGroupOutcome* indicates that the outcome identified by attribute *td:TestGroupOutcome@testGroupOutcomeID* shall be returned. In other words, the outcome specified by *td:TestGroupOutcome* overrides the outcome determined through test group execution. This is typically done to force a “Failed” or “Aborted” outcome when test group termination fails.

The example instance document “PrePostCond.xml” contains a simple example for specifying test group termination through a test that calls a test group.

6.4 Conditions

6.4.1 Preconditions

In many applications, specific preconditions must be met before UUT tests can be performed. These preconditions could be, for example: the UUT is connected to the ATE, power is applied to the UUT, the appropriate firmware is loaded in the UUT, UUT temperature is within specified limits, and so on.

In the *TestDescription* schema, the preconditions are represented as follows:

- a) For a test, by the *td:Test/td:Conditions/td:PreConditions* element and its descendants.
- b) For a test group, by the *td:TestGroup/td:Conditions/td:PreConditions* element and its descendants.

Each *td:PreCondition* child element shall specify one precondition. Multiple preconditions can be combined through a logical formula using *td:PreConditions/td:Composition* elements and *td:PreCondition@operatorNot* attributes, for example “Condition1 AND (NOT Condition2)”.

Each precondition shall be specified as follows:

- The presence of child element *td:PreCondition/td:VariableValue* indicates that the precondition is determined by the value of a *GlobalVariable*. The precondition shall be true when the variable

referenced by attribute *td:VariableValue@globalVariableID* has the value specified in the *td:VariableValue/tdValue* child element and shall be false otherwise.

- The presence of child element *td:PreCondition/td:Other* indicates that the precondition is specified through free-form text. The element shall contain the description of the precondition, for example “UUT is connected to the ATE”.

Preconditions described through variable values can be handled automatically at run time. Those described through free form text are meant to inform the test program developer regarding the conditions to be verified by the test program.

Note that the handling of preconditions at run time is not specified in this standard and can be application-specific. When a precondition is not met some applications could, for example, terminate test program execution. Other applications could automatically execute an entry point that ensures that the precondition will be met. Applications using a diagnostic reasoner could sequence steps in such a way that preconditions are always met.

6.4.2 Postconditions

Typically, the preconditions identified for tests or test groups can be fulfilled by executing other tests or other test groups. For example, a test of type “session action” that applies power to the UUT fulfills the precondition “power is applied to the UUT” of some other test.

In this standard, the conditions fulfilled through the execution of a test or test group are called “postconditions”. In the *TestDescription* schema the postconditions are represented as follows:

- a) For a test, by the *td:Test/td:Conditions/td:PostConditions* element and its descendants, or by the *td:Test/td:Outcomes/td:Outcome/td:PostConditions/td:PostCondition* element and its descendants.
- b) For a test group, by the *td:TestGroup/td:Conditions/td:PostConditions* element and its descendants, or by the *td:TestGroup/td:Outcomes/td:Outcome/td:PostConditions/td:PostCondition* element and its descendants.

Each *td:PostCondition* child element shall specify one postcondition fulfilled by the test or test groups. The *td:PreConditions/td:Composition* elements and the *td:PreCondition@operatorNot* attribute shall not be used with postconditions.

Each postcondition shall be specified as follows:

- The presence of child element *td:PostCondition/td:VariableValue* indicates that the postcondition is indicated by setting the value of a *GlobalVariable*. When the test referenced by the postcondition returns a “Done” or “Passed” outcome the value specified in the *td:VariableValue/td:Value* element shall be assigned to the *GlobalVariable* referenced by attribute *td:VariableValue@globalVariableID*.
- The presence of child element *td:PreCondition/td:Other* indicates that the postcondition is specified through free-form text. The element shall contain the description of the postcondition, for example “UUT power is on”.

Postconditions associated with specific outcomes of a test or a test group are fulfilled only when that outcome is returned at run time by the test or test group. Postconditions associated with the test or test group are fulfilled when the test or test group is executed, regardless of the outcome returned.

If postconditions are specified for a test or test group and also for some of its outcomes, the following rule applies: if for the currently returned outcome a postcondition is specified, then that postcondition gets fulfilled; otherwise, the postcondition specified for the test or test group gets fulfilled.

If postconditions are *not* specified for a test or test group but are specified for some of its outcomes, the following rule applies: if for the currently returned outcome a postcondition is specified, then that postcondition gets fulfilled; otherwise, no postcondition gets fulfilled (for example, the value of the global variable used to describe conditions remains unchanged).

The example instance document “PrePostCond.xml” illustrates the use of conditions in conjunction with tests, to ensure that UUT power was turned on. The UUT power-on state is represented by the value of the *GlobalVariable* with ‘ID’ = “sv1”. The variable assumes two values, “ON” and “OFF”. The value is initially “OFF”. Its value is set to “ON” by the test with ‘ID’ = “test7” (which applies power) and is set to “OFF” by the test with ‘ID’ = “test8” (which removes power). The test with ‘ID’ = “test2” and the test with ‘ID’ = “test4” require power to be on.

The example instance document “IEEE 1671.1 – Example2.xml” illustrates the use of conditions in conjunction with test groups, to ensure that a “Safe to Turn On” test group was previously performed and passed. Note that in this example the postconditions of the *TestGroup* with ‘ID’ = “seq2” are associated with *specific outcomes* of that *TestGroup*. The UUT safe-to-turn-on state is represented by the value of the *GlobalVariable* with ‘ID’ = “sv1”. The variable of type *c:boolean* has two possible values, “true” and “false”. The value is initially “false”. The value is set to “true” when the *TestGroup* with ‘ID’ = “seq2” returns a “Passed” outcome and to “false” when *TestGroup* returns a “Failed” outcome. If the test is aborted, the value of the *GlobalVariable* remains unchanged.

The example instance document “IEEE 1671.1 – Example4.xml” illustrates the use of conditions to constrain the automatic sequencing of tests by a diagnostic reasoner. Note that a postcondition is associated with the test with @ID = “test” and another postcondition is associated with its “Passed” outcome. The first postcondition gets fulfilled when the test fails or is aborted.

6.4.3 Recommended use of global variables

As described in the previous subclause, preconditions and postconditions are typically related through global variables that assume a discrete set of values. In general, these variables describe the state of the UUT (for example, UUT power is ON) or the state of test execution (for example, test with ‘ID’ = “123” was executed and failed).

The recommended practice is to access these “state” variable exclusively from preconditions and postconditions. Accessing them from within tests (for example, through Operations) makes those tests more dependent on state, limiting their potential for reuse.

6.5 Test group calls

Test groups can be used to describe sequences of tests that are reused multiple times within a test description. For example, a test group can verify the operation of one channel from a multi-channel UUT. The tests within that test group are performed multiple times, once for each UUT channel.

To facilitate this reuse scenario, the *TestDescription* schema includes a special class of tests, whose behavior consists of performing a particular test group. To describe such a test, the child element *td:TestGroupCall* shall appear under *td:Test/td:Behavior*. The attribute ‘testGroupID’ of element *td:TestGroupCall* shall reference by the test group to be performed when the current test is performed.

To describe a situation where sequences of tests are reused, the ATML instance document will typically contain one or more “top-level” test groups, along with some “low-level” test groups that describe the reusable behavior. Some of the tests referenced by steps in top-level test group reference the lower-level test groups, as shown in Figure 1. A low-level test group can be referenced by more than one test.

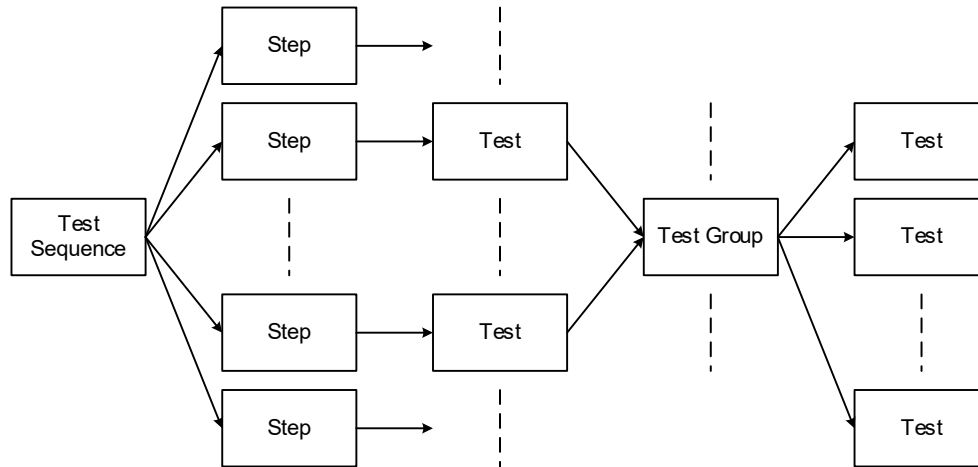


Figure 1—Test groups calls

Note that the *TestDescription* schema does not specify the mechanism for implementing the execution of test groups referenced through *td:TestGroupCall*. This could be achieved, for example, through a function call included in the test program code, through a call made by a run time engine, and so on.

Test groups that describe reusable behavior often have parameters. These parameters allow the customization of the reusable behavior each time that behavior is invoked. For example, a UUT may have multiple channels that process signals of different frequencies. Consequently, the stimulus signals used to test each channel have different frequencies. In this situation, the test group that verifies an individual UUT channel has a parameter that describes the center frequency of the channel. Each time the test group is invoked, a different value can be assigned to this parameter. The value is specified in a parameter of the calling test. It is transferred at run time to a “matching” parameter in the called test group.

Test groups that describe reusable behavior may return test results. These test results are transferred to test results of the calling test, where they can be used to determine the outcome, or for other purposes such as display, logging, and so on.

Any *Test* that calls a *TestGroup* through *td:TestGroupCall* shall have the following characteristics:

- If at least one of the *TestResult* elements of the calling test has limits, the *Outcomes* of the calling test shall be “Passed” and “Failed”, without qualifiers. Otherwise, the *Outcomes* of the calling test shall be identical to the *Outcomes* of the referenced *TestGroup*. This requirement covers both outcome values and outcome qualifiers (including the absence of a qualifier).
 - 1) For example, if the *TestGroup* has an *Outcome* with ‘value’ = “v1” and ‘qualifier’ = “q1”, then the referencing tests are required to have an outcome with ‘value’ = “v1” and ‘qualifier’ = “q1”.
 - 2) As another example, if the *TestGroup* has an *Outcome* with ‘value’ = “v2” and no qualifier, then the referencing tests are required to have an outcome with ‘value’ = “v2” and no qualifier.

- The *Parameters* of the calling test shall match the *ParameterDescriptions* of the referenced *TestGroup*. This requirement covers names, value types, and units (including the absence of a unit).
 - 1) For example, if the *TestGroup* has a *td:ParameterDescription* with 'name' = "param1", data type *td:doubleDescription* assigned to element *td:ParameterDescription/td:ValueDescription/td:DatumDescription*, and the unit "V" assigned to attribute *td:doubleDescription@standardUnit*, then the referencing Tests are required to have a *td:Parameter* with 'name' = "param1", type *c:double* assigned to element *td:Parameter/td:Value/c:Datum*, and a unit convertible to "V" (i.e., a Voltage unit) assigned to attribute *c:double@standardUnit*.
 - 2) As another example, if the *TestGroup* has a *td:ParameterDescription* with 'name' = "param2", data type *td:integerDescription* assigned to element *td:ParameterDescription/td:ValueDescription/td:DatumDescription*, and no unit specified, then the referencing Tests are required to have a *td:Parameter* with 'name' = "param2", type *c:integer* assigned to element *td:Parameter/td:Value/c:Datum*, and no unit specified.
- The *TestResults* of the calling test shall match the *TestResultDescriptions* of the referenced *TestGroup*. This requirement covers names, value types, and units (including the absence of a unit).
 - 1) For example, if the *TestGroup* has a *td:TestResultDescription* with 'name' = "testresult1", data type *td:doubleDescription* assigned to element *td:TestResultDescription/td:ValueDescription/td:DatumDescription*, and the unit "V" assigned to attribute *td:doubleDescription@standardUnit*, then the referencing Tests are required to have a *td:TestResult* with 'name' = "testresult1", data type *c:double* assigned to element *td:TestResult/td:ValueDescription/c:DatumDescription*, and a unit convertible to "V" (i.e., a Voltage unit) assigned to attribute *c:double@standardUnit*.
 - 2) As another example, if the *TestGroup* has a *td:TestResultDescription* with 'name' = "testresult2", data type *td:integerDescription* assigned to element *td:TestResultDescription/td:ValueDescription/td:DatumDescription*, and no unit specified, then the referencing Tests are required to have a *td:TestResult* with 'name' = "testresult2", data type *c:integer* assigned to element *td:TestResult/td:ValueDescription/c:DatumDescription*, and no unit specified.

When a *td:Test* with *td:TestGroupCall* behavior is performed, the following actions shall be performed in order:

- a) If any *Preconditions* are specified for the *Test*, those *Preconditions* shall be verified (for example, by checking the values of *GlobalVariables*).
- b) The values of *Test Parameters* shall be transferred to the corresponding *Parameters* of the called *TestGroup*. The *Test Parameters* and the *TestGroupParameters* shall be matched by name.
- c) The test group call shall be performed as follows:
 - 1) If any *Preconditions* are specified for the *TestGroup*, those *Preconditions* shall be verified.
 - 2) If any *Initialization* actions are specified for the *TestGroup*, those actions shall be performed.
 - 3) The *Steps* of the called *TestGroup* shall be executed according to the sequencing method specified for the *TestGroup* (serial, fault tree, and so on). If the *TestGroup* has multiple *EntryPoint*s, execution shall start at the primary *EntryPoint*.
 - 4) While the *TestGroup* is executing, values shall be assigned to its *TestResults* (if applicable).
 - 5) The outcome of the *TestGroup* shall be determined according to the method specified by the *TestGroup* (for example, using a fault tree or a fault dictionary).
 - 6) If any *Termination* actions are specified for the *TestGroup*, those actions shall be performed.
 - 7) If any *Postconditions* are specified for the *TestGroup*, those *Postconditions* shall be indicated (for example, by setting the values of global variables).

- d) The values of *TestGroup TestResults* shall be transferred to the corresponding *TestResults* of the calling *Test*. The *TestResults* of the *Test* and the *TestResults* of the *TestGroup* shall be matched by name.
- e) If any *Postconditions* are specified for the *Test*, those *Postconditions* shall be recorded.
- f) The test *Outcome* shall be determined as follows:
 - 1) If at least one of the *TestResults* of the calling *Test* has limits, the *Outcome* of the *Test* shall be evaluated by comparing its *TestResults* to the corresponding limits; the *Outcome* shall be “Passed” if all limit comparisons pass, and “Failed” otherwise.
 - 2) Otherwise, the *Outcome* of the *Test* to be equal to the *Outcome* evaluated in a previous *Step* for the *TestGroup*.

The example instance document “IEEE 1671.1 – Example3.xml” contains a representative example of a test description with test group calls. The *TestGroup* with ‘ID’ = “seq2” is called from the *Test* with ‘ID’ = “test8” and from the *Test* with ‘ID’ = “test9”. These *Tests* are referenced by *Steps* in the *TestGroup* with ‘ID’ = “seq1”. The *Outcomes* of the *TestGroup* match the *Outcomes* of the calling *Test*.

7. Describing data

7.1 Overview of data items

The ATML Test Description schema defines several entities that store and transfer data during test program execution, such as parameters, test results, and variables. For brevity, throughout this standard these entities will be named “data items”.

Figure 2 illustrates the data items modeled in the *TestDescription* schema.

- The rectangular shapes represent data items and other items defined in the XML schema, such as Tests and Operations.
- The connecting lines represent hierarchical relationships, for example “Parameter” is a child of “Test”. For simplification, the diagram omits the collector elements present in the XML schema. For example, instead of the “Test” – “Parameters” – “Parameter” relationships represented in the XML schema, the diagram show just “Test” – “Parameter”.
- The thick lines represent items with global scope: *TestGroup* and *GlobalVariable*.

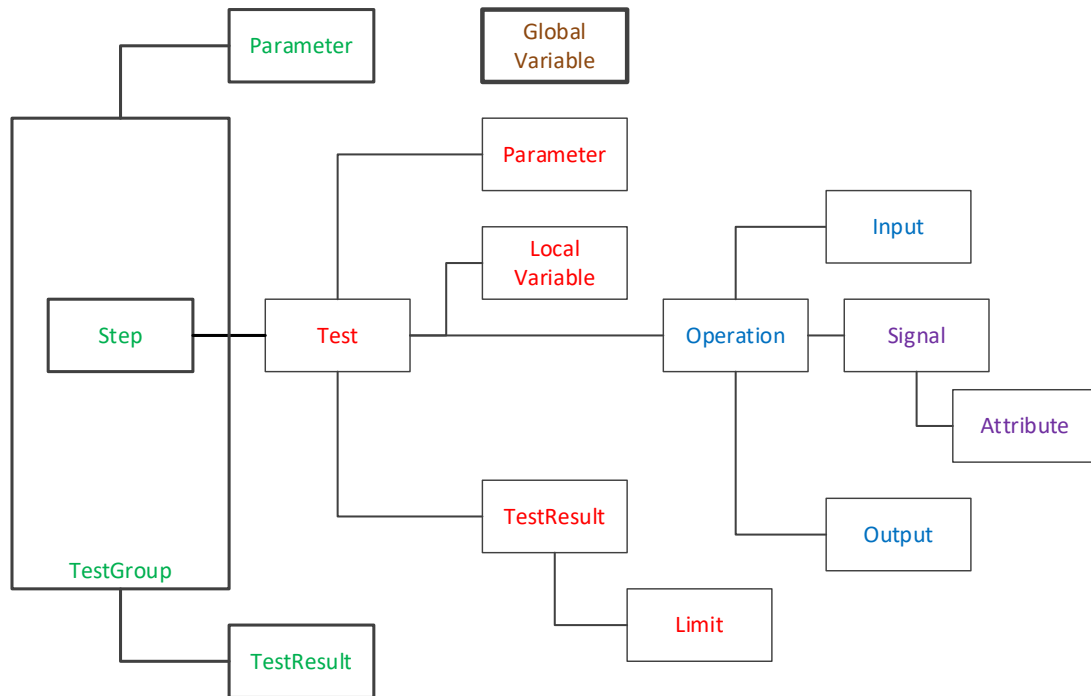


Figure 2—Data items modeled in the ATML Test Description Schema

The most important data items are defined as follows:

Parameters represent the data inputs of Tests and TestGroups. They are used to define reusable Tests and TestGroups. Examples include the frequency range for a power measurement, the number of hops in a frequency sweep, and the UUT ports where a measurement is made.

TestResults represent the data outputs of Tests and TestGroups. Examples include measurement results, calculated values, and messages received from the UUT through a serial data bus. TestResults are used for a dual purpose:

- To identify the test results to be stored during test program execution. For example, the test results can be stored in an XML instance document conformant with IEEE Std 1636.1 [B8].
- To describe the transfer of test result data to another data item. For example, when a test is executed part of a test group the results of the test can be passed back to the test group. Similarly, when a test group is called by a test the results of the test group can be passed back to the calling test.

Variables represent test program items that store data during test program execution, facilitating data transfers between multiple data sources and data destinations. The *TestDescription* schema defines two types of variables:

- **GlobalVariables** store values during the execution of an entire test program. They are created and initialized when the test program starts. GlobalVariables are typically used to store data that must be accessible from multiple Tests. For example, a GlobalVariable stores a power level calculated by one Test and use by several subsequent Tests. GlobalVariables are also used to store references to signals that must remain active during the execution of multiple Tests.
- **Variables** (sometimes referred to as “local variables”) store values during the execution of a Test. They are created and initialized when the Test starts. They are no longer accessible (“out of scope”) after the Test ends. Variables are typically used to store data that must be accessible from multiple

Operations within the Test. For example, a Variable stores a calculated frequency value used by multiple “Setup” operations. Variables are also used to store references to signals that are used locally within the Test.

All variables defined in *TestDescription* instance documents are initialized. The initial value is specified in the *td:Variable/td:Value* element. This can be an immediate value or a reference to another data item from which the initial value is to be retrieved. Refer to 7.3.1 for information on specifying immediate values and to 7.4.1 for information on data item references.

Operations can retrieve input data from various data items in the *TestDescription*. Depending on the Operation type, input data can be used to set signal attributes, perform calculations, display messages to the ATE operator, etc. Operations can return output data to various data items in the test description. Depending on the Operation type, output data can contain measured values of signal attributes, calculation results, input data from the ATE operator, etc.

7.2 Overview of data flows

The *TestDescription* schema allows the description of data transfers between data items. For example, the value of a test parameter can be transferred to an operation, which in turn assigns the value to a signal attribute. Similarly, the measured value of a signal attribute can be passed back to the operation that performs the measurement, then to a test result. These transfers will be named “data flows”.

Table 1 summarizes the data flows that can be represented using the *TestDescription* schema. The numbers that appear in table cells are used to identify the data flow throughout this clause.

- The numbers in round brackets “()” denote data flows described through explicit mapping between the data items.
- The numbers in curly brackets “{ }” denote data flows described through implicit mapping between the data items.
- The numbers in square brackets “[]” denote data flows that can be described indirectly through specialized Operations.

Table 1—Data Flows in the ATML Test Description Schema

To → From ↓	TestGroup TestResult	Test Parameter	Variable (initial value)	Variable	TestResult	TestResult Limit	Operation Input	Operation Output	Signal Attribute	Called TestGroup Parameter	GlobalVariable (initial value)	GlobalVariable
TestGroup Parameter	—	(1)	(2)	[3]	—	(4)	(5)	—			N/A	[6]
Test Parameter	—	—	(7)	[8]	—	(9)	(10)	—		{11}	N/A	[12]
Variable	[13]	N/A	N/A	[14]	[15]	(16)	(17)	—			N/A	[18]
TestResult	(19)	(20)	N/A	N/A	N/A	(21)	(22)	—			N/A	(23)
Operation Input	—	—	—	—	—	—	—	—	(24)	—	—	—
Operation Output	(25)	—	—	(26)	(27)	—	—	—				(28)
Signal Attribute								(29)				
Called TestGroup TestResult		—	—		{30}	—	—	—			—	
GlobalVariable	[31]	(32)	(33)	[34]	[35]	(36)	(37)	—			N/A	[38]

The colors used in the table have the following significance:

- **Red** identifies the Test data items (other than Operations and TestGroup calls) and the data flows between these data items.
- **Green** identifies the TestGroup data items and the data flows between these data items and the data items of Tests that belong to the TestGroup.
- **Blue** identifies the Operation data items and the data flows to, from, and between these data items.
- **Purple** identifies the Signal data items and the data flows to and from these data items.
- **Magenta** identifies the data items for TestGroup calls and the data flows to and from the data items of the called TestGroup.
- **Brown** identifies the global data items and the data flows to and from these data items.

In [Table 1](#), “Called TestGroup” refers to a TestGroup referenced by a Test with TestGroupCall behavior. The data flows marked “N/A” can be described using the available data types, but cannot be implemented in test programs because the data source is not available at the time of the data transfer. For example, the initial value of a Variable can be set to reference a TestResult of the same Test. However, this data transfer cannot be implemented because the Variable is initialized when the test starts, at which time the test result is not determined yet.

[Figure 3](#) shows examples of data flows between data items belonging to TestGroups and Tests, as well as GlobalVariables. The numbers printed on the data flow arrows match the numbers in [Table 1](#). The dash-dotted lines represent data transfers to the initial value of a variable.

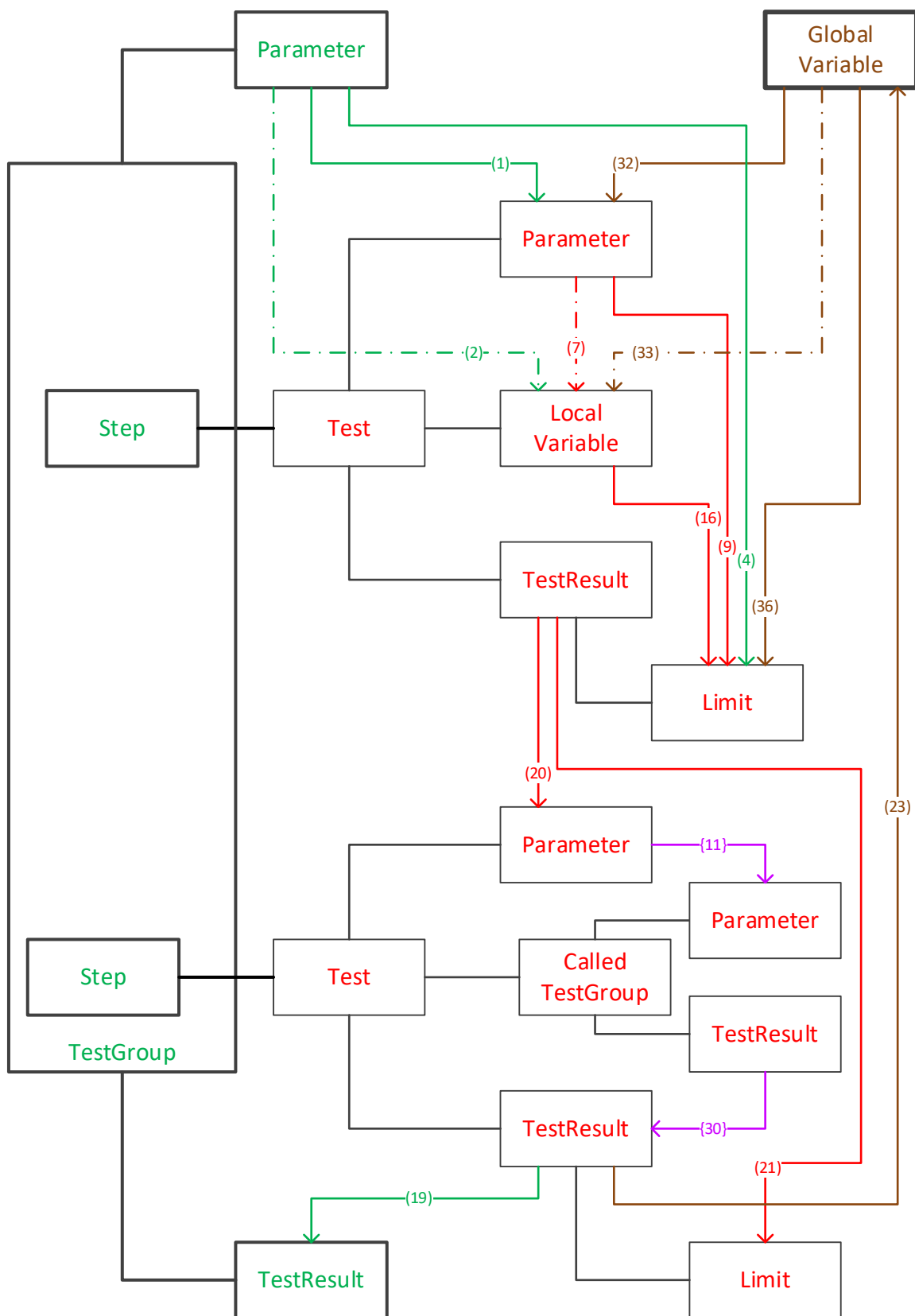


Figure 3—Data flows between TestGroups, Tests, and GlobalVariables

Figure 4 shows examples of data flows to and from Operations, as well as to and from Signal Attributes. The numbers printed on the data flow arrows match the numbers in Table 1.

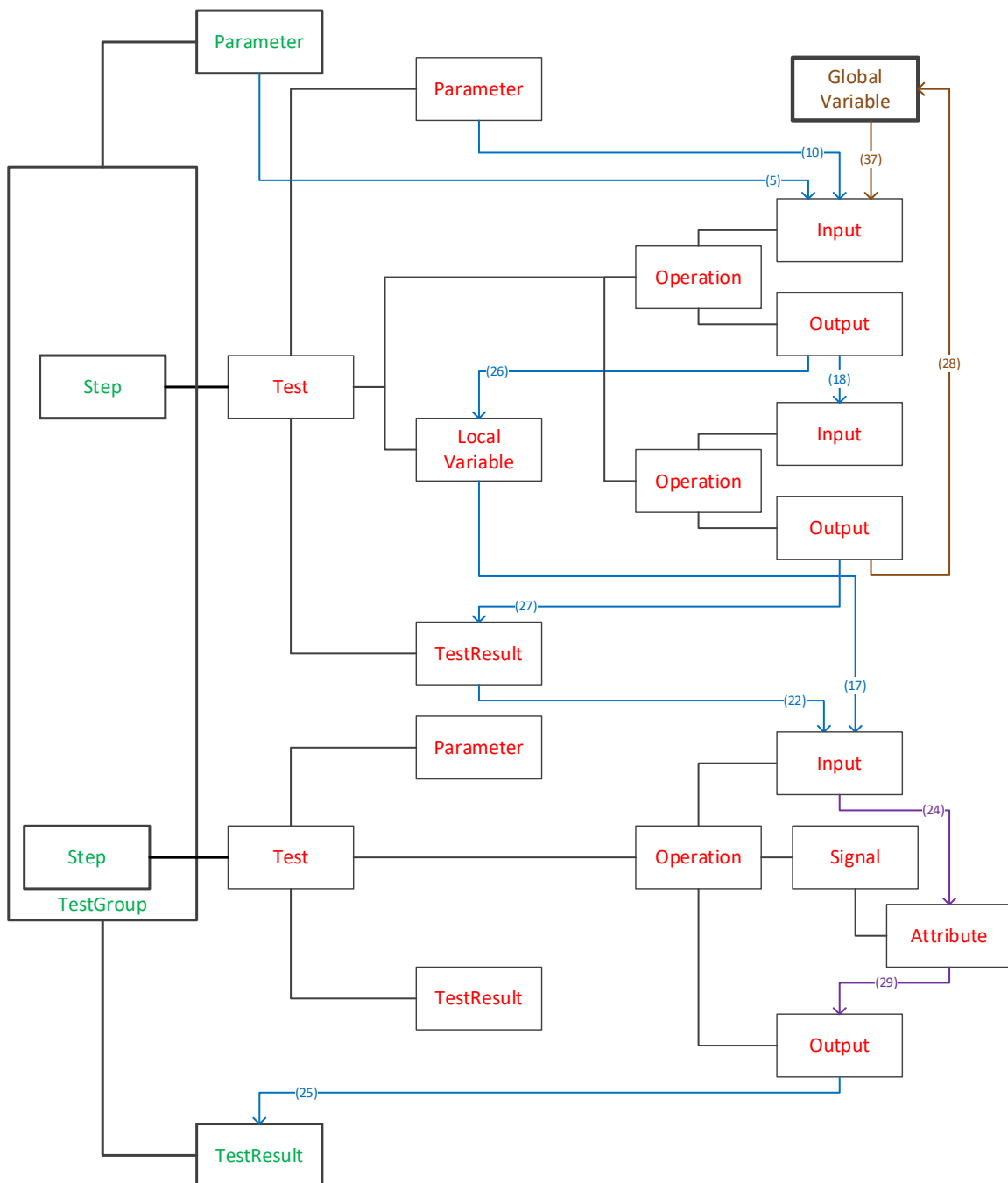


Figure 4—Data flows between Operations and Signal Attributes

Figure 5 shows examples of data flows that can be represented indirectly, through Operations. The numbers printed on the data flow arrows match the numbers in Table 1.

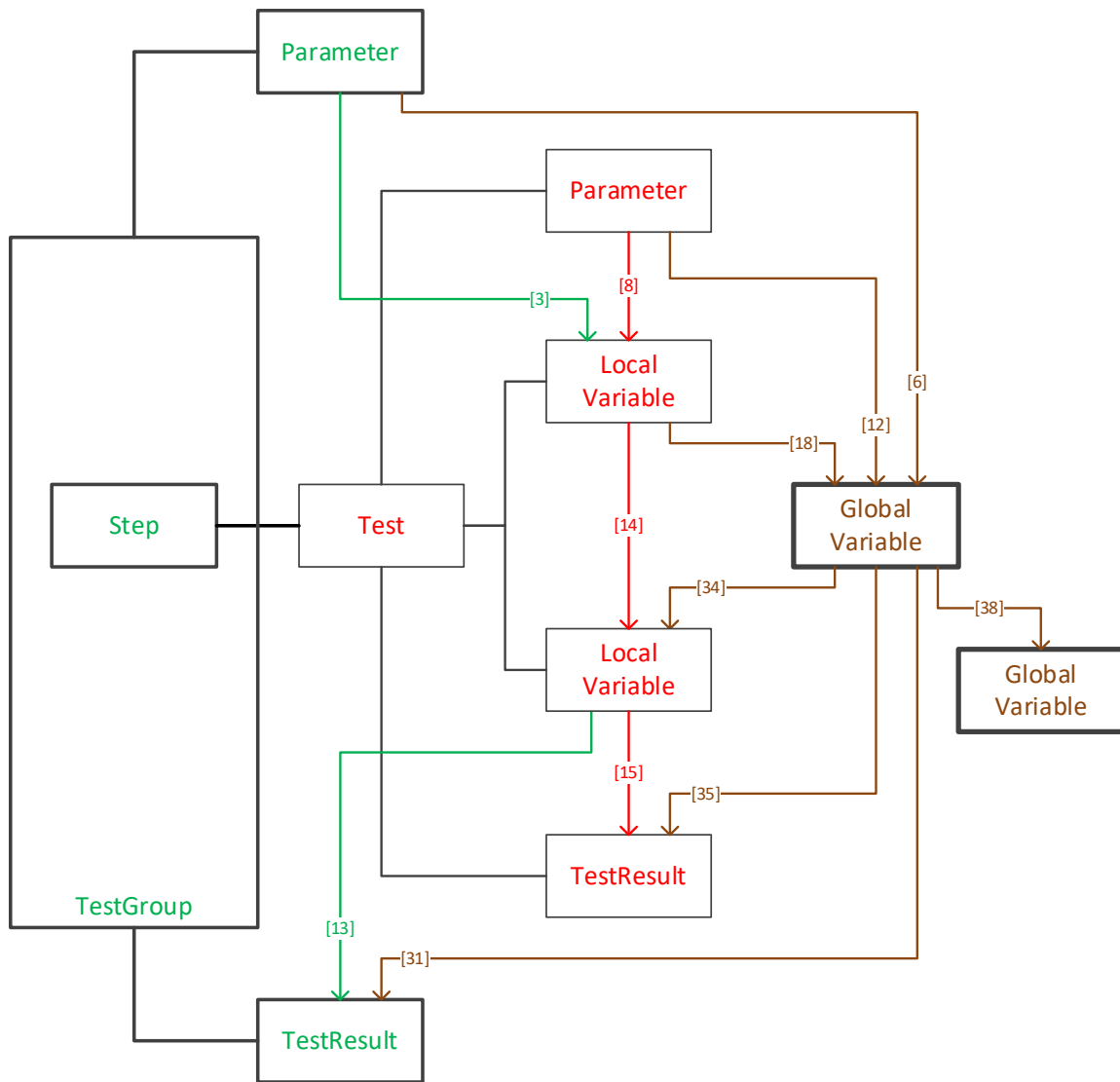


Figure 5—Data flows that can be represented indirectly, through Operations

7.3 Describing data items

7.3.1 Using the *c:Value* Type

In the *TestDescription* schema the type *c:Value* is assigned to data items (or their descendants) to describe their data type and to specify a value. Refer to IEEE Std 1671-2010 for detailed information on the *c:Value* type.

To specify a **scalar** data item, use the *c:Datum* child element of *c:Value* and assign to it one of the *c:DatumType*-derived types defined in the *Common* schema. For example, the following XML code

describes a scalar *td:Parameter* of type “double”, with a value of 1.1 V. Note the use of the ‘*xsi:type*’ attribute to identify the type assigned to *c:Datum*.

```
<td:Parameter ID="1001" name="Voltage">
  <td:Value>
    <c:Datum xsi:type="c:double" standardUnit="V" value="1.1"/>
  </td:Value>
</td:Parameter>
```

To specify an **array** data item, use the *c:IndexedArray* child element of *c:Value* and assign to it one of the *c:IndexedArrayType*-derived types defined in the *Common* schema. For example, the following XML code describes an array *td:Parameter* with elements of type “double”. Note the use of the ‘*xsi:type*’ attribute to identify the type assigned to *c:IndexedArray*.

```
<td:Parameter ID="1002" name="FrequencyHops">
  <td:Value>
    <c:IndexedArray xsi:type="c:doubleArray" dimensions="[3]"
      standardUnit="MHz">
      <c:Element position="[0]" value="1.0"/>
      <c:Element position="[1]" value="1.1"/>
      <c:Element position="[2]" value="1.2"/>
    </c:IndexedArray>
  </td:Value>
</td:Parameter>
```

To specify a **collection** data item, use the *c:Collection* child element of *c:Value* and assign to each collection item one of the *c:DatumType*-derived types defined in the *Common* schema. For example, the following XML code describes a *td:Parameter* of type collection, with two properties of type “double” whose values are 2.55 mm and 3.46 mm.

```
<td:Parameter ID="1003" name="Coordinates">
  <td:Value>
    <c:Collection>
      <c:Item name="x">
        <c:Datum xsi:type="c:double" standardUnit="mm"
          value="2.55"/>
      </c:Item>
      <c:Item name="y">
        <c:Datum xsi:type="c:double" standardUnit="mm"
          value="3.46"/>
      </c:Item>
    </c:Collection>
  </td:Value>
</td:Parameter>
```

To specify a **connection** data item, use the *c:Datum* child element of *c:Value* and assign to it the type *td:ConnectionDatum* defined in file “DatumTypes.xsd” of the *TestDescription* schema. For example, the following XML code describes a connection *td:Parameter* that references a UUT port. Note the use of the ‘*xsi:type*’ attribute to identify the type assigned to *c:Datum*.

```
<td:Parameter ID="1004" name=" SerialPortConnection ">
  <td:Value>
    <c:Datum xsi:type="td:ConnectionDatum">
      <td:Network>
        <hc:Node>
          <hc:Path>/td:TestDescription/td:InterfaceRequirements/c:Ports/
```

```
c:Port[@name="COM1"]</hc:Path>
    </hc:Node>
  </td:Network>
</c:Datum>
</td:Value>
</td:Parameter>
```

To specify a **signal reference** data item, use the *td:Datum* child element of *td:Value* and assign to it the type *td:SignalReference* defined in file “DatumTypes.xsd” of the *TestDescription* schema. For example, the following XML code describes a *td:Variable* that stores a signal reference. Refer to 8.3.4.1 for detailed information on signal references. Note the use of the ‘*xsi:type*’ attribute to identify the type assigned to *c:Datum*.

```
<td:Variable ID="var1" name="PowerSignal">
  <td:Value>
    <c:Datum xsi:type="td:SignalReference"/>
  </td:Value>
</td:Variable>
```

7.3.2 Using the *td:ValueDescription* type

In the *TestDescription* schema the type *td:ValueDescription* is assigned to data items (or their descendants) to describe their data type, without specifying an actual value. The actual value is produced at run time, for example through measurement.

To describe a **scalar** data item, use the *td:DatumDescription* child element of *td:ValueDescription* and assign to it one of the *td:DatumDescriptionType*-derived types defined in file “DatumTypes.xsd” of the *TestDescription* schema. For example, the following XML code describes a scalar *td:TestResult* of type “double”. Note the use of the ‘*xsi:type*’ attribute to identify the type assigned to *td:DatumDescription*.

```
<td:TestResult ID="2001" name="td:Voltage">
  <td:ValueDescription>
    <td:DatumDescription
      xsi:type="td:doubleDescription" standardUnit="V"/>
    </td:DatumDescription>
  </td:ValueDescription>
</td:TestResult>
```

To describe an **array** data item, use the *td:IndexedArrayDescription* child element of *td:ValueDescription* and assign to it one of the *td:IndexedArrayDescriptionType*-derived types defined in file “DatumTypes.xsd” of the *TestDescription* schema. For example, the following XML code describes an array *td:TestResult* with elements of type “double” and unknown size. Note the use of the ‘*xsi:type*’ attribute to identify the type assigned to *td:IndexedArrayDescription*.

```
<td:TestResult ID="2002" name="PeakFrequencies ">
  <td:ValueDescription>
    <td:IndexedArrayDescription
      xsi:type="td:doubleArrayDescription" standardUnit="GHz"/>
    </td:ValueDescription>
</td:TestResult>
```

If the size of the array is known, then the attribute ‘*dimensions*’ of the *td:IndexedArrayDescriptionType*-derived types can be used to describe it.

A nominal value can optionally be specified for data items described using the *td:ValueDescription* type. The nominal value represents the value of the data item (for example, *TestResult*) obtained for a UUT that