IEC 61671

Edition 1.0 2012-06

IEEE Std 1671™

INTERNATIONAL STANDARD

Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML

IEC 61671:2012(E) IEEE Std 1671-2010

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue
New York, NY 10016-5997
United States of America
stds.info@ieee.org
www.ieee.org

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

Useful links:

IEC publications search - www.iec.ch/searchpub

The advanced search enables you to find IEC publications by a variety of criteria (reference number, text, technical committee,…).
It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available on-line and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

IEC 61671

Edition 1.0 2012-06

INTERNATIONAL STANDARD

IEEE Std 1671™

Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PRICE CODE XN

ICS 25.040; 35.060

ISBN 978-2-83220-104-6

Warning! Make sure that you obtained this publication from an authorized distributor.

Contents

Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation.

   IEEE Standards documents are developed within IEEE Societies and Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. IEEE develops its standards through a consensus development process, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of IEEE and serve without compensation. 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 contained in its standards. Use of IEEE Standards documents is wholly voluntary. IEEE documents are made available for use subject to important notices and legal disclaimers (see http://standards.ieee.org/IPR/disclaimers.html for more information).

   IEC collaborates closely with IEEE in accordance with conditions determined by agreement between the two organizations.

2) The formal decisions of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees. The formal decisions of IEEE on technical matters, once consensus within IEEE Societies and Standards Coordinating Committees has been reached, is determined by a balanced ballot of materially interested parties who indicate interest in reviewing the proposed standard. Final approval of the IEEE standards document is given by the IEEE Standards Association (IEEE-SA) Standards Board.

3) IEC/IEEE Publications have the form of recommendations for international use and are accepted by IEC National Committees/IEEE Societies in that sense. While all reasonable efforts are made to ensure that the technical content of IEC/IEEE Publications is accurate, IEC or IEEE cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.

4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications (including IEC/IEEE Publications) transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC/IEEE Publication and the corresponding national or regional publication shall be clearly indicated in the latter.

5) IEC and IEEE do not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC and IEEE are not responsible for any services carried out by independent certification bodies.

6) All users should ensure that they have the latest edition of this publication.

7) No liability shall attach to IEC or IEEE or their directors, employees, servants or agents including individual experts and members of technical committees and IEC National Committees, or volunteers of IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board, for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC/IEEE Publication or any other IEC or IEEE Publications.

8) Attention is drawn to the normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

9) Attention is drawn to the possibility that implementation of this IEC/IEEE Publication may require use of material covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. IEC or IEEE shall not be held responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patent 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.

International Standard IEC 61671/ IEEE Std 1671-2010 has been processed through IEC technical committee 93: Design automation, under the IEC/IEEE Dual Logo Agreement.

The text of this standard is based on the following documents:

IEEE Std	FDIS	Report on voting
IEEE Std 1671-2010	93/323/FDIS	93/330/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

The IEC Technical Committee and IEEE Technical Committee have decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

IEEE Std 1671™-2010
(Revision of
IEEE Std 1671-2006)

IEEE Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML

Sponsor

IEEE Standards Coordinating Committee 20 on Test and Diagnosis for Electronic Systems

Approved 30 September 2010

IEEE-SA Standards Board

Abstract: This document specifies a framework for the automatic test markup language (ATML) family of standards. ATML allows automatic test system (ATS) and test information to be exchanged in a common format adhering to the extensible markup language (XML) standard.

Keywords: ATE description, ATE test results, ATML, ATS, automatic test equipment, automatic test markup language, automatic test system, interface test adapter, ITA, SI, synthetic instrumentation, test configuration, unit under test, UUT description, UUT maintenance, XML instance document, XML schema

IEEE Introduction

This introduction is not part of IEEE Std 1671-2010, IEEE Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML.

Historical background

In 2002, an automatic test markup language (ATML) focus group was formed (outside any formal standardization body) with a mission to "define a collection of XML [extensible markup language] schemas that allows ATE [automatic test equipment] and test information to be exchanged in a common format adhering to the XML standard."

The scope of this effort was the standardization of test information, which would allow for test program (TP) and test asset interoperability, as well as unit under test (UUT) data (including results and diagnostics), to be interchanged between heterogeneous ATE systems.

In 2004, the efforts of the focus group were brought into IEEE Standards Coordinating Committee 20 (SCC20), where the formal standardization process has taken place. Further refinements and updates to the work accomplished by the ATML focus group has (and continues to) take place within both the ATML focus group and IEEE SCC20.

IEEE 1671 ATML family of standards

The ATML family of standards supports TP, test asset, and UUT interoperability within an automatic test environment.

This document provides an overview of the ATML goals, defines the ATML framework, defines the ATML family of standards, and specifies common ATML data elements, and common ATML schemas.

Notice to users

Laws and regulations

Users of these documents should consult all applicable laws and regulations. Compliance with the provisions of this standard 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

This document is copyrighted by the IEEE. It is made available 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 this document available for use and adoption by public authorities and private users, the IEEE does not waive any rights in copyright to this document.

Updating of IEEE documents

Users of IEEE standards 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. An official 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. 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 Standards Association Web site at http://ieeexplore.ieee.org/xpl/standards.jsp, or contact the IEEE at the address listed previously.

For more information about the IEEE Standards Association or the IEEE standards development process, visit the IEEE-SA Web site at http://standards.ieee.org.

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: http://standards.ieee.org/reading/ieee/updates/errata/index.html. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/reading/ieee/interp/index.html.

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 with respect to the existence or validity of any patent rights in connection therewith. 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 nondiscriminatory. 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.

Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML

IMPORTANT NOTICE: This standard is not intended to ensure safety, security, health, or environmental protection. Implementers of the standard are responsible for determining appropriate safety, security, environmental, and health practices or regulatory requirements.

1. Overview

1.1 General

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 standard.[1]

The ATML framework and the ATML family of standards have been developed and are maintained under the guidance of the Test Information Integration (TII) Subcommittee of IEEE Standards Coordinating Committee 20 (SCC20) to 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 (TP), test asset interoperability, and unit under test (UUT) data to be interchanged between heterogeneous systems.

[1] This 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.

1.1.1 ATML framework referenced IEEE standard

The ATML framework can reference IEEE Std 1641™ [B29].[2] This referenced IEEE standard, when utilized, is then considered part of the ATML framework.

1.1.2 Application of this document's annexes

This document includes twelve annexes. Of these twelve, four are normative (Annex A, Annex B, Annex C, and Annex D).

Annex A contains style guidelines for the ATML family XML schemas. Annex A guidelines shall be followed by ATML XML schema developers and maintainers during the development and maintenance of all ATML family XML schemas, including the XML schemas associated with this document.

Annex B contains XML schema element description and definitions for the **ATML common element XML schemas**. Annex B shall be utilized by ATML XML schema developers, maintainers, and ATML users. Annex B shall be referenced during the development and maintenance of all ATML family XML schemas, including the XML schemas associated with this document.

Annex C contains XML schema element description and definitions for the **ATML internal model XML schemas**. Annex C shall be utilized by ATML XML schema developers, maintainers, and ATML users. Annex C shall be referenced during the creation and development of ATML Capabilities or ATML WireLists documents.

Annex D contains guidelines for ATML services. Annex D shall be referenced by ATML users implementing an ATML framework.

Annex E through Annex L are informative and thus are provided strictly as information for both users and maintainers of this document.

1.2 Scope

ATML defines a standard exchange medium for sharing information between components of ATSs. This information includes test data, resource data, diagnostic data, and historic data. The exchange medium is defined using XML. This standard specifies the framework for the family of ATML standards.

1.3 Purpose

The purpose of ATML is to support TP, test asset, and UUT interoperability within an automatic test environment. ATML accomplishes this through a standard medium for exchanging UUT, test, and diagnostic information between components of the test system. The purpose of this standard is to provide an overview of ATML goals, define the ATML family of standards, and specify common data elements for the ATML family of standards.

[2] The numbers in brackets correspond to the numbers of the bibliography in Annex L.

1.4 Application

1.4.1 General

This document should be applied anywhere ATS and test information is to be exchanged. This ATS and test information includes the following:

- Data that will be utilized for the design, development, and utilization of automatic test equipment (ATE).
- Data that will be utilized for the design, development, and utilization of test program sets (TPSs) to test a product (e.g., UUT) on a particular ATE.
- Product design data that will be utilized during the testing of the product (e.g., UUT).
- Shared usage of maintenance data and the results of testing a product (e.g., UUT).
- Testing requirements of a particular product (e.g., UUT).
- Data that will be utilized for the design, development, and utilization of instrumentation that will be utilized within a particular ATS configuration.
- A definition of allowable ATS configurations that can be use to test and evaluate a particular product (e.g., UUT).
- A definition of the capabilities of ATSs as well as the elements of the ATS.

1.4.2 Users

Anticipated users of the ATML family of standards include the following:

- Product (e.g., UUT) developers
- Product (e.g., UUT) maintainers
- TPS developers
- TPS maintainers
- ATE system developers
- ATE system maintainers
- Instrumentation developers
- Developers of ATML-based tools and systems
- Developers of prime mission equipment that use the supported UUT as a component

1.4.3 Precedence

In the event of conflict between this document and an ATML family component standard, this document shall take precedence.

In the event of conflict between this document and a normatively referenced standard (*Extensible Markup Language (XML) 1.0*),[3] the normatively referenced standard, as it applies to the information being produced, shall take precedence.

In the event of conflict between this document's XML schema definition and/or annotations and an ATML family component standard and/or XML schemas, this document's XML schema definition and/or annotations shall take precedence.

1.5 Conventions used in this document

1.5.1 General

Within the body of this document, the conventions defined in Table 1 are utilized.

Table 1—Document conventions

Item	Convention
The use of **bolded** text	Emphasizes a word or concept.
The use of *italics*	Represents bibliography references and quoted text from other documents.
The term "ATML framework"	Represents the sub-domains of an ATS architecture specifically addressed by the ATML family of standards.
The term "ATML family of standards"	Represents the complete set of ATML family component standards and associated XML schemas (see Table 3).
The term "ATML family component standard"	Represents a particular IEEE 1671 series standard (IEEE Std 1671.1™ [B31] through IEEE Std 1671.6™ [B36]).
The term "ATML common element schemas"	Reflects only the ATML Common, ATML HardwareCommon, and ATML TestEquipment XML schemas defined in this document (B.1, B.2, and B.3). These schemas **shall not have** associated XML instance documents.
The term "ATML internal model schemas"	Reflects only the ATML Capabilities and ATML WireLists XML schemas defined in this document (C.1 and C.2). These schemas **shall have** associated XML instance documents.
The term "subdomain"	Represents the complete set of ATML family component standards and associated XML schemas.
The term "subframework"	Represents the ATS or support software.
The term "external interface"	Represents the IEEE 1671 series standards (IEEE Std 1671.1 through IEEE Std 1671.6).
The term "internal model"	Represents the Capabilities and WireLists XML schemas defined in Annex C.
The term "ATML <component name> XML schema"	Represents the XML schema associated with the ATML family component standard. <component name> is defined in Table 3.
The term "ATML <component name> document"	Represents an XML instance document conforming to the ATML <component name> XML schema. For example, an ATML Test Description document is an XML instance document conforming to the ATML Test Description XML schema. <component name> is defined in Table 3.

[3] Information on references can be found in Clause 2.

Annex A, Annex B, Annex C, Annex D, Annex E, Annex F, Annex G, and Annex I present XML schema and XML instance document information. The conventions used in their presentation are defined in Table 2.

Table 2—XML schema and XML instance document conventions

Item	Convention
All specifications in the XML language	Are given in the `Courier` type font where the XML elements are represented outside the XML schema or instance document.
The terms "isRef (0)" and "isRef (1)"	Is a XML boolean indicator used to identify whether an object is a reference. The term isRef (0) indicates it is not a reference; isRef (1) indicates that it is a reference.
The term "final #all"	Is an XML property that prevents all derivation. Used by the Complex Type `c:Extension`.
The use of "—" in tables	Indicates that no information is associated with this table cell or, with respect to attribute usage, implies optional.
The term "content simple"	Indicates that the XML element is not allowed to have attributes or subelements.
The term "content complex"	Indicates that a new complex data type is being defined, which can be used to declare elements to accept attributes and/or subelements.
The use of *italics*	Represents a XML element defined outside the subclause.
The use of "1 … ∞" and "0 … ∞" in tables	Represents the number of times an XML element may appear in an XML instance document. i.e., either one to infinity times or zero to infinity times.
XML snippets of XML instance documents	Are given in the `Courier` type font.
The XML attribute "`xsi:type`"	Explicitly declares the XML element type.
The use of "|" in XML simple types descriptions	Indicates a logical OR.

This document uses the vocabulary and definitions of relevant IEEE standards. In case of conflict of definitions, except for the portions quoted from standards, the following precedence shall be observed: (1) Clause 3, (2) Annex K, and (3) *The IEEE Standards Dictionary: Glossary of Terms & Definitions.*

For clarity, portions of IEEE Std 1641 [B29] have been duplicated within this document. In the event of revision to IEEE Std 1641, the current, approved version of the revised standard shall take precedence.

1.5.2 Word usage

In this document, the word *shall* is used to indicate a mandatory requirement. The word *should* is used to indicate a recommendation. The word *may* is used to indicate a permissible action. The word *can* is used for statements of possibility and capability.

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). The latest edition of the referenced document (including any amendments, corrigenda, and/or working group drafts) applies unless the specific year of publication or edition is referenced.

Extensible Markup Language (XML) 1.0, (Fifth Edition). W3C Proposed Edited Recommendation, 5 Feb. 2008.[4]

[4] This document is available from the World Wide Web Consortium (W3C®) (http://www.w3.org/xml).

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this document, the following terms and definitions apply. *The IEEE Standards Dictionary: Glossary of Terms & Definitions* should be referenced for terms not defined in this clause.[5] In the event a term is explicitly redefined or further defined in an ATML family component standard, that component standard's definition shall be normative only for that particular component standard.

abstract data type: A declared type that can be used to define other types through derivation. Only nonabstract types derived from the declared type can be used in instance documents. When such a type is used, it must be identified by the `xsi:type` attribute.

application: **(A)** The use to which a system is put. **(B)** The use of capabilities provided for by a system specific to the satisfaction of a set of users' requirements.

automatic test equipment (ATE): An integrated assembly of stimulus, measurement, and switching components under computer control that is capable of processing software routines designed specifically to test a particular item or group of items. ATE software includes operating system software, test executive software, and instrument control software.

NOTE—Definition adapted from DoD ATS Selection Process Guide [B7].[6]

automatic test equipment (ATE) control software: Software used during execution of a test program (TP) that controls the nontesting operations of the ATE. This software executes a test procedure but does not contain any of the stimuli or measurement parameters used in testing the unit under test (UUT). Where test software and control software are combined in one inseparable program, that program will be treated as test software, not control software.

NOTE—Definition adapted from MIL-STD-1309D [B52].

automatic test equipment (ATE) oriented language: A computer language used to program an ATE to test units under test (UUTs). The characteristics of this language imply the use of a specific ATE system or family of ATE systems.

NOTE—Definition adapted from MIL-STD-1309D [B52].

automatic test equipment (ATE) support software: Computer programs that aid in preparing, analyzing, and maintaining unit under test (UUT) test programs (TPs). Examples are ATE compilers and translation and analysis programs.

NOTE—Definition adapted from MIL-STD-1309D [B52].

automatic test equipment (ATE) system software: The total software environment of the ATE including operating system, test executives, user interface, system self-test, and other software required to run test programs (TPs). This term does not include TPs for supported end items.

NOTE—Definition adapted from MIL-STD-1309D [B52].

[5] *The IEEE Standards Dictionary: Glossary of Terms & Definitions* is available at http://shop.ieee.org/.
[6] Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement the standard.

automatic test markup language (ATML) capabilities: An extensible markup language (XML) schema that provides for the mapping of tests to instruments (and test systems) in a way that makes it possible to tell whether a test system is able to execute a given test.

automatic test markup language (ATML) instance document: *See*: **instance document**.

automatic test markup language (ATML) namespace: A collection of names, identified by a uniform resource identifier (URI) reference.

NOTE—Definition adapted from *Namespaces in XML 1.0* [B43].

automatic test markup language (ATML) ports, pins, and connectors: The description of instruments, test systems and their capabilities at the instruments, test systems pins, respectively.

automatic test system (ATS): A fully integrated, computer-controlled suite of electronic test equipment hardware, software, documentation, and ancillary items designed to verify at any level of maintenance the functionality of unit under test (UUT) assemblies. An ATS combines the following three elements: automatic test equipment (ATE), test program set (TPS), and test environment.

NOTE—Definition adapted from DoD ATS Selection Process Guide [B7].

D connector: A type of connector so named because one side is shorter (with one less pin) than the other, giving a physical D-shape.

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.

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.

NOTE—Definition adapted from *Extensible Markup Language (XML) 1.0*.

entity: Something that has a distinct separate existence.

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) schema: The definition of a class of XML document, 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.

extensible markup language (XML) style sheet: A description of how an XML document is to be presented on a computer screen or in print.

fault: Degradation in performance due to detuning, misadjustment, misalignment, or failure of part(s).

framework: A real or conceptual structure intended to serve as guidance for building something that expands this structure into something useful.

global attribute: An attribute declaration that is a child of the xs:schema element. A global attribute can be applied to any element.

hypertext markup language (HTML) viewer: A software program that enables a human to read an HTML file in its native format.

instance document: An extensible markup language (XML) document that conforms to a particular XML schema.

isRef: A boolean indicator used to identify that an extensible markup language (XML) object is a reference.

object: An entity that consists of state and behavior. An object stores its states in fields (variables in some programming languages) and exposes its behavior through methods (functions in some programming languages).

signal: **(A)** An electrical impulse controlled or observed by a test resource. **(B)** A visual, audible, or other indication used to convey information.

software tool: A software program that aids in the development of other software programs.

stimulus: Any physical or electrical input applied to a test subject intended to produce a measurable response.

NOTE—Definition adapted from MIL-STD-1309D [B52].

test environment: A description of the automatic test system (ATS) architecture, programming and test specification languages, compiler, development tools, and provisions for capturing and using unit under test (UUT) design requirements and test strategy information in the generation and maintenance of test program set (TPS) software.

NOTE—Definition adapted from DoD ATS Selection Process Guide [B7].

test executive: A software application that controls the execution environment of unit under test (UUT) test programs (TPs). Typical functions include, but are not limited to, verifying hardware/software availability, interpreting and executing operators commands, initializing and controling tests, providing common subprograms for test software usage, providing debug and simulation capabilities, logging test data, and allocating virtual resources.

test program set (TPS): Automatic test equipment (ATE) interface hardware, test program (TP) software, documentation, and other ancillary equipment that connects the unit under test (UUT) to the ATE. The TPS software performs fault isolation and diagnostics and can certify a UUT as ready for issue. Ancillary hardware consists of probes, holding fixtures, and peculiar instrumentation.

NOTE—Definition adapted from DoD ATS Selection Process Guide [B7].

unicode: An industry standard designed to allow text and symbols from all languages to be consistently represented and manipulated.

unicode transformation format (UTF): A variable-length character encoding for unicode.

well-formed: Conforming to all of extensible markup language (XML) syntax rules.

3.2 Acronyms and abbreviations

AI-ESTATE	Artificial Intelligence Exchange and Service Tie to All Test Environments
AM	amplitude modulation
API	application programming interface
ARB	arbitrary waveform generator
ATE	automatic test equipment
ATLAS	Abbreviated Test Language for All Systems
ATML	automatic test markup language
ATS	automatic test system
BIT	built-in test
BNC	Bayonet Neill Concelman connector
BSC	basic signal component
CAGE	commercial and government entity
CORBA	common object request broker architecture
COM	component object module
CSCI	computer software configuration item
DLL	dynamic link library
DMM	digital multimeter
FM	frequency modulation
GUID	globally unique identifier
HTML	hypertext markup language
IC	integrated circuit
ID	identifier or interface device
I/O	input/output
ITA	interface test adapter
IVI®	interchangeable virtual instrumentation
LAN	local area network
LRU	line replaceable unit
PC	personal computer
PCB	printed circuit board
PM	phase modulation
R&D	research and development
RF	radio frequency
RFI	receiver/fixture interface
rms	root mean square

SGML	standard generalized markup language
SCC20	Standards Coordinating Committee 20
SCPI	standard commands for programmable instrumentation
SI	synthetic instrumentation
SIMICA	Software Interface for Maintenance Information Collection and Analysis
SMA	subminiature A connector
SRA	shop replaceable assembly
SRU	shop replaceable unit
STD	signal and test definition
TAR	test accuracy ratio
TII	Test Information Integration
TP	test program
TPS	test program set
TSF	test signal framework
UML	unified modeling language
URI	uniform resource identifier
URN	uniform resource name
URL	universal resource locator
UTC	coordinated universal time
UTF-8	8-bit unicode transformation format
UUID	universal unique identifier
UUT	unit under test
VME	versa module europa
VPP	VXI plug and play
VSWR	voltage standing wave ratio
VXI	VMEbus extensions for instrumentation
W3C®	World Wide Web Consortium
WRA	weapons replaceable assembly
WSDL	web services definition language
XHTML	extensible hypertext markup language
XML	extensible markup language
XSD	XML schema document

4. Automatic test system (ATS) architecture[7]

ATSs are utilized to identify failed electronic components, adjust these components to meet specifications, and assure that an electronic system or electronic component is "ready for issue"; in other words, the item is functioning as it was designed to operate.

An ATS includes the following:

a) ATE hardware and its operating software.

b) TPSs, which include the hardware, software, and documentation required to interface with, and test, individual component items. The associated software development environments required to produce the TPS are also included.

c) Automatic diagnostics and testing.

4.1 Automatic test equipment (ATE)

ATE refers to the test hardware and its accompanying software.

ATE utilizes one or more computers to control test instruments such as digital voltmeters, waveform analyzers, signal generators, and switching assemblies. This equipment operates under control of test software to provide a stimulus to a particular circuit or component in the UUT and then measure the output at various pins, ports, or connections to determine whether the UUT has performed to its specifications. The basic definition of ATE, then, is computer-controlled stimulus and measurement.

ATE is widely used in the electronic manufacturing industry to test electronics components and systems, both before and after they are fabricated. These electronic components and systems include (but are not limited to) avionics systems on commercial and military aircraft, electronic modules in automobiles, and electronic medical devices.

An ATE can be configured to test:

a) Simple components (resistors, capacitors, and inductors).

b) Integrated circuits (ICs).

c) Printed circuit boards (PCBs). PCBs can also be referred to as either shop replaceable units (SRUs) or shop replaceable assemblies (SRAs).

d) Black boxes [sometimes called either line replaceable units (LRUs) or weapons replaceable assemblies (WRAs)].

e) "All Up Round" weapons and weapon sections.

f) Other related electronic components or modules.

4.1.1 ATE hardware

The hardware architecture of an ATS will depend on its planned use, e.g., research and development (R&D), design validation, manufacturing, or field support; on budget and development-time constraints; existing expertise; and measurement throughput requirements.

[7] Concepts described in this clause have, in part, been derived from the DOD ATS Handbook [B5].

In R&D, for example, parametric tests are performed but will not be repeated on hundreds of UUTs. In high-volume manufacturing, for example, hundreds to thousands of UUTs may be tested, and each has to be tested as fast and as inexpensively as possible.

The hardware architecture of the ATE will be different in each of these situations. Typically, the ATE hardware architecture contains six major subsystems:

a) Instrumentation (measuring and stimulus instruments)
b) Computing [computer, software, and input/output (I/O)]
c) Switching (relays that interconnect system instrumentation and loads to the UUT)
d) Mass interconnects (UUT-to-system wiring interface)
e) Power sources (power to the UUT)
f) UUT-specific connections (e.g., loads, serial interfaces)

Each of these six subsystems is depicted by the grey shaded items in Figure 1.

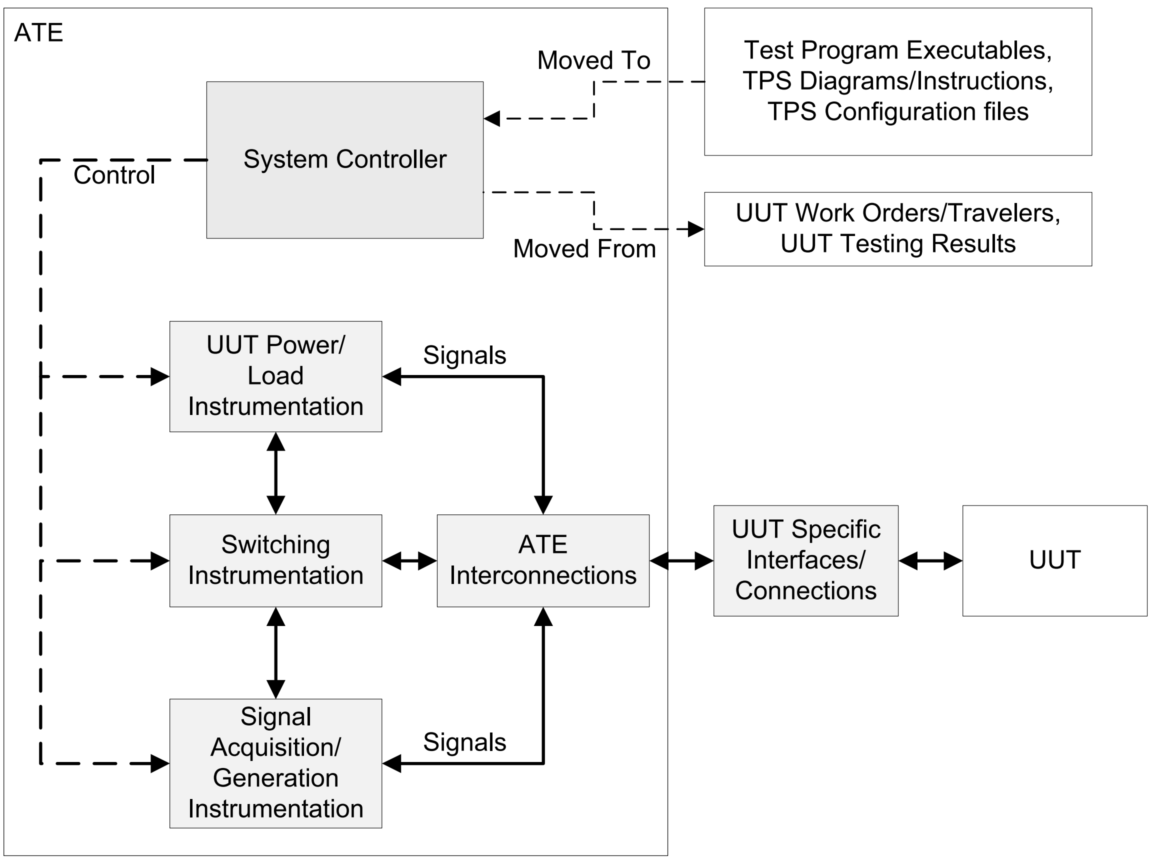


Figure 1—ATS hardware subsystems

4.1.2 ATE software

A "typical" ATE software architecture consists of at least the following two computer software configuration items (CSCIs):

a) ATE support software

b) ATE system software

ATE support software comprises software items typically running on a standalone personal computer (PC) for the purposes of developing a TP, while the ATE system software comprises software items running on the ATE system controller executing the TP. Both the ATE system software and the ATE system software are depicted in Figure 2 (as the "callouts" over the ATS depicted in Figure 1).

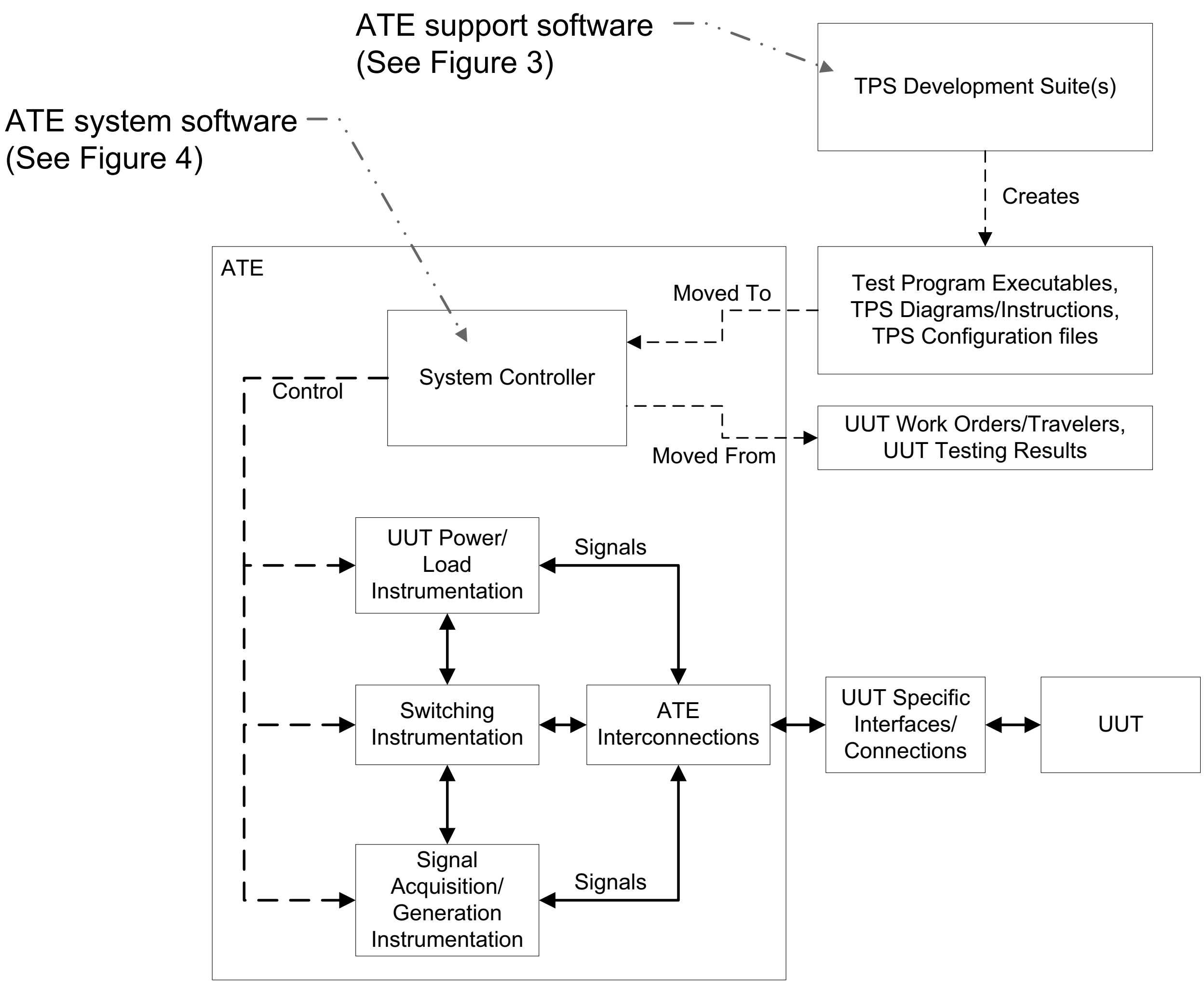


Figure 2—Software associated with a "typical" ATS

4.1.2.1 ATE support software

ATE support software consists of the software that aids in the preparation, analysis, and maintenance of UUT TPs. The ATE support software typically is not available on the ATE station.

In order to actually conduct tests on the UUT utilizing the ATE and interface test adapter (ITA), a UUT TP needs to be developed from the UUT testing strategy so that it may be executed by the ATE station control software.

Historically, the elements of a particular ATE's support software incorporated interpretations by the ATE developers of such items as instrument vendor data sheets/documentation, ATE hardware design material, etc. These interpretations are effectively turning one data format into a second (usually proprietary) format (e.g., an instrument data sheet contents in PDF format, put into a compiler's instrument database's unique file format). This usually always loses something in the translation as information is lost, does not have a home, etc.

"Typical" ATE support software items are depicted in Figure 3 (as the "TPS Development Suite(s)" over the ATS depicted in Figure 1).

4.1.2.2 ATE system software

ATE system software is the total software environment of the ATE including operating system, test executives, user interface, system self-test, and other software required to run UUT TPs.

In order to run the tests on the UUT, the UUT TP created via the ATE support software (see 4.1.2.1) needs an environment to be executed from.

Historically, the elements of a particular ATEs station control software interfaced with and produced data in proprietary formats (TP intermediate programming languages, test results, etc.) This meant that only the matching ATE support software could be utilized, and that test results were represented/store in a format unique to that ATE.

"Typical" ATE system software items are depicted in Figure 4 (as the "System Controller Software" over the ATS depicted in Figure 1).

4.2 Test program set (TPS)

TPSs consist of the test software, interface devices, cabling, and associated documentation.

The computer(s) in the ATE execute the test software, which usually is written in a standard language such as Abbreviated Test Language for All Systems (ATLAS), C, or Visual Basic. The stimulus and measurement instruments in the ATS have the ability to respond as directed by the computer. They send signals where needed and take measurements at the appropriate points. The test software then analyzes the results of the measurements and determines the probable cause of failure. It displays to the technician the component to remove and replace.

Developing the test software requires a series of tools collectively referred to as the software development environment. These tools include ATE and UUT simulators, ATE and UUT description languages, and programming tools such as compilers.

Since each UUT likely has different connections and I/O ports, interfacing the UUT to the ATE normally requires an interconnecting device (known as an ITA) and cables, which physically connect the UUT to the ATE and route signals from the various I/O pins in the ATE to the appropriate I/O pins in the UUT.

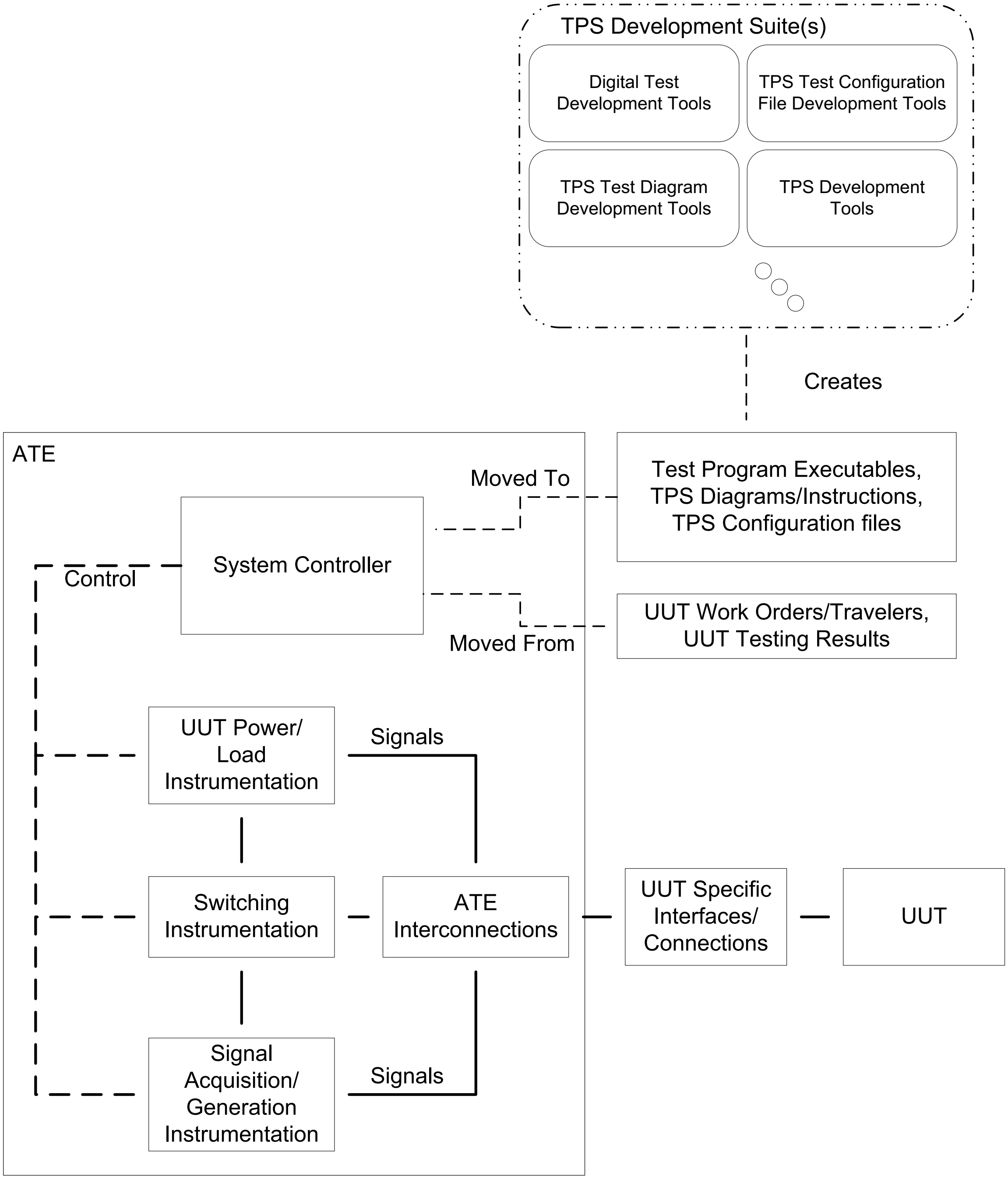


Figure 3—ATE support software

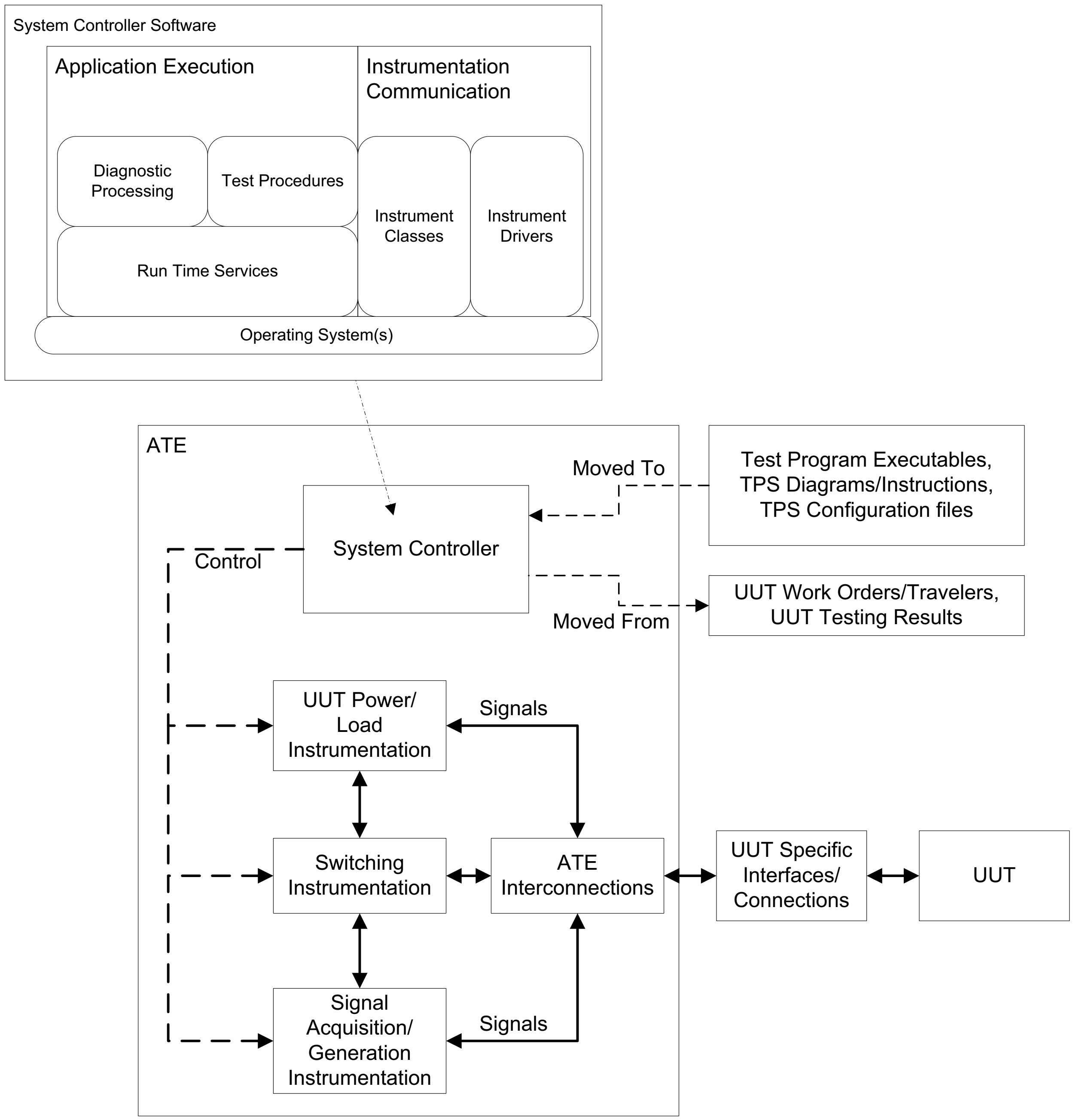


Figure 4—ATE system software

4.3 Automatic diagnosis and testing

Diagnostics is the part of an ATE test that determines the faulty components. ATE tests perform two basic functions. The first is to test whether the UUT is working correctly. The second is to diagnose the reason the UUT is not working correctly. The diagnostic portion can be the most difficult and costly portion of the test. It is typical for ATE to reduce a failure to a cluster or ambiguity group of components.

5. Automatic test markup language (ATML)

The ATML initiative has come about by the desire to standardize on the XML format, rather than the various proprietary tools and formats used within the test industry (as discussed in Clause 4). By using a common format, different tools and systems can exchange information and be brought together to form cooperative heterogeneous systems, which, through the use of ATML, can

- Decrease test times.
- Reduce incidents of **Can Not Duplicate** or **No Fault Found.**
- Reduce the repair cycle.
- Formalize the capture of historic data that have been the preserver of experts in the field to heuristically identify faulty components.
- Close the loop on diagnostic systems.

The goals of ATML are to

- Establish an industry standard for test information exchange.
- Develop a exchange format that can be understood by man or machine.
- Allow, and design for, user extensibility.
- Establish a process for managing extensibility.
- Ensure acceptance within the user community.

The general uses cases ATML supports are to

- Support dynamic test sequences that can change with historical data.
- Support instrument setup directly.
- Support instrument setup using signal descriptions.
- Support parallel/simultaneous testing and complex timing relationships.
- Capture test results.
- Capture TP information and sequencing.
- Capture instrument specifications and capabilities.
- Capture test station specifications and capabilities.
- Capture test setup and test configurations.
- Capture UUT specifications and requirements.
- Capture test support hardware and software.
- Capture UUT diagnostic and maintenance information.

5.1 ATS architecture elements addressed by ATML

ATML formally standardizes the following distinct subdomains under the ATML framework:

- The descriptions of how a test signal description "maps" to a ATE station (e.g., capabilities).
- The descriptions of test instrumentation.
- The descriptions of ITAs.
- The descriptions of the ATS configurations under which a UUT can be tested.
- The descriptions of UUT tests.
- The descriptions of test stations.
- The descriptions of a UUT.
- The descriptions of the elecrical paths from the UUT to the instrument in the ATE, on a per-test basis.

These sub-domains are depicted as the "callouts" in Figure 5, which are depicted "over the top" of the ATS architecture described in Clause 4.

A complete ATS architecture, including these ATML framework subdomains, is provided by Annex H.

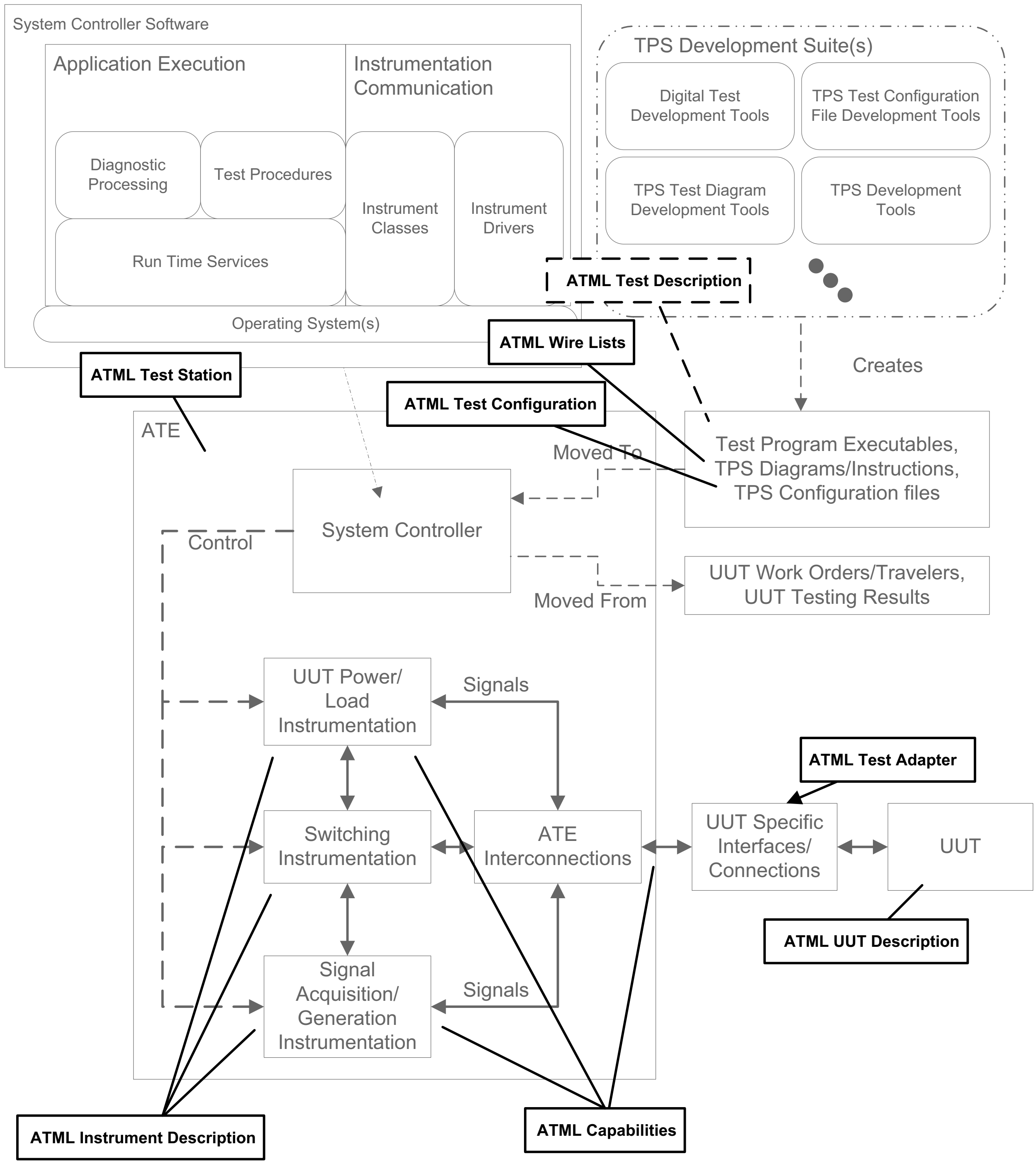


Figure 5—The ATML family component standards associated with a "typical" ATS architecture

NOTE—The ATML Test Description in Figure 5 is represented as a "dashed line" because the Test Program Executables are to be "derived from" the ATML Test Description.

6. The ATML framework

The ATML framework is based upon XML. XML describes a class of data objects called XML documents and partially describes the behavior of computer programs, which process them. XML is an application profile or restricted form of the standard generalized markup language (SGML) (see ISO 8879:1986 [B39]). By construction, XML documents are conforming SGML documents.

The ATML framework has been developed to

— Summarize and organize the essential elements of an ATS.
— Provide a common frame of reference.
— Eliminate the need to use a variety of custom file formats.
— Provide compliance with the World Wide Web Consortium (W3C)[8] standards.
— Be based upon standards.
— Be extensible.
— Enable the creation of modular ATS architectures (components based upon the ATML family component standards can easily be substituted, and data can be shared between the components).

The ATML framework is defined in the form of three distinct approaches:

— External interfaces
— Internal models
— Services

All external interfaces and internal models shall be with reference to the ATS and UUT. The ATML framework expects, although will not define, services to be available to generate, consume, and manipulate this information.

6.1 External interfaces

The external interfaces represent information that is exchanged between two or more distinct components in a typical ATS being used for testing of an UUT.

As defined in 5.1, and depicted by Figure 5 and Figure J.1, ATML formally standardizes six distinct external interfaces as ATML framework subdomains:

— Instrument descriptions
— Test adapter descriptions
— Test configurations

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

— Test descriptions

— Test station descriptions

— UUT descriptions

Collectively, these six ATML framework subdomains were deemed to be the interfaces that offer the largest potential of reducing the cost to rehost or replace ATS components (up to as much as the entire ATE or the entire TPS).

NOTE—One with experience in ATE and/or TPSs might recognize that **a TP is not one of the ATML framework subdomains**. This omission is not an oversight. The ATML philosophy is that, since a TP is typically developed from test strategies and test requirements, the description of these test strategies and test requirements should be standardized as opposed to standardizing on the implementation of them (e.g., the TP). This approach allows for scenarios where the TPs are written in different languages. An example of this ATML philosophy is provided in I.2. This philosophy permits test strategies and test requirements to be implemented in the software programming language of choice, with the added benefit of potentially accessing a newer instrument's added capabilities in the future. ATML standardizes these test strategies and test requirements as part of the ATML Test Description component.

6.2 Internal models

Internal models ensure a consistent approach to defining elements that need common semantics. Within ATML there are several such models:

— Signal definitions using IEEE Std 1641 [B29]

— ATML capabilities

— ATML wire lists and network lists

As defined in 5.1, and depicted by Figure 5, ATML formally standardizes two distinct internal models as ATML framework subdomains:

— ATML capabilities

— ATML wire lists

The use of these items within the ATML framework ensures that different elements interpret the same information in the same way.

6.3 Services

The definition of the external interfaces (6.1) and internal models (6.2) is generally not enough to achieve interoperability. A simple scenario of **Tell me your test configuration?** or **What is the next test?** requires not only the definition of the format of the information, but also a definition of how the questions should be asked.

The web services infrastructure is founded on communication via XML-based messages that comply with a published web service description. The service description is an XML document written in an XML grammar called the web services definition language (WSDL) that defines the format of messages the web service understands. The service description serves as an agreement that defines the behavior of a web service and instructs potential clients in how to interact with it.

ATML does not formally define specific services; however, ATML implementations should define services using WSDL. An example WSDL service is provided in Annex D.

7. ATML specification techniques

The ATML framework is defined in terms of subframeworks and subdomains.

There are two major subframeworks, the ATS subframework and the support subframework.

The ATS sub-framework is analogous to the ATE system software defined in 4.1.2.2, and the support subframework is analogous to the ATE support software defined in 4.1.2.1.

The ATS subframework is further subdivided into following subframeworks:

- UUT & TPS
- Support software
- System software.

The support subframework is not subdivided.

Each of these four (i.e., UUT & TPS, support software, system software, and support) subframeworks contains one or more ATML subdomains. ATML subdomains are analogous to the following:

a) The six external interfaces described in 6.1.
    1) Each of the six are IEEE 1671 series 'dot' standards (e.g., IEEE Std 1671.1 [B31] through IEEE Std 1671.6 [B36], inclusive).
b) The two internal models described in 6.2.
    1) Both are defined in Annex C of this standard.
c) The three ATML common element schemas derived from the partitioning described in 7.1.
    1) Each of the three are defined in Annex B of this standard.

The ATML framework, the subframeworks, and the ATML subdomains are depicted by Figure 6.

7.1 ATML common element partitioning

Common elements provide for the definition of XML types and attributes that are utilized within more than one ATML subdomain XML schemas.

Common element XML schemas are reference XML schemas containing only type definitions that may be used in other XML schemas. **They have no root element, and there will be no instance documents directly validated against them.**

Having each ATML subdomain XML schema include ATML common elements allows for a consistent definition of shared XML types and prevents each XML schema from defining XML types used by other ATML subdomain XML schemas (which would have had to also define that XML type, possibly differently).

Common elements, as a result, is simply a **toolbox** for the ATML subdomain XML schemas to include.

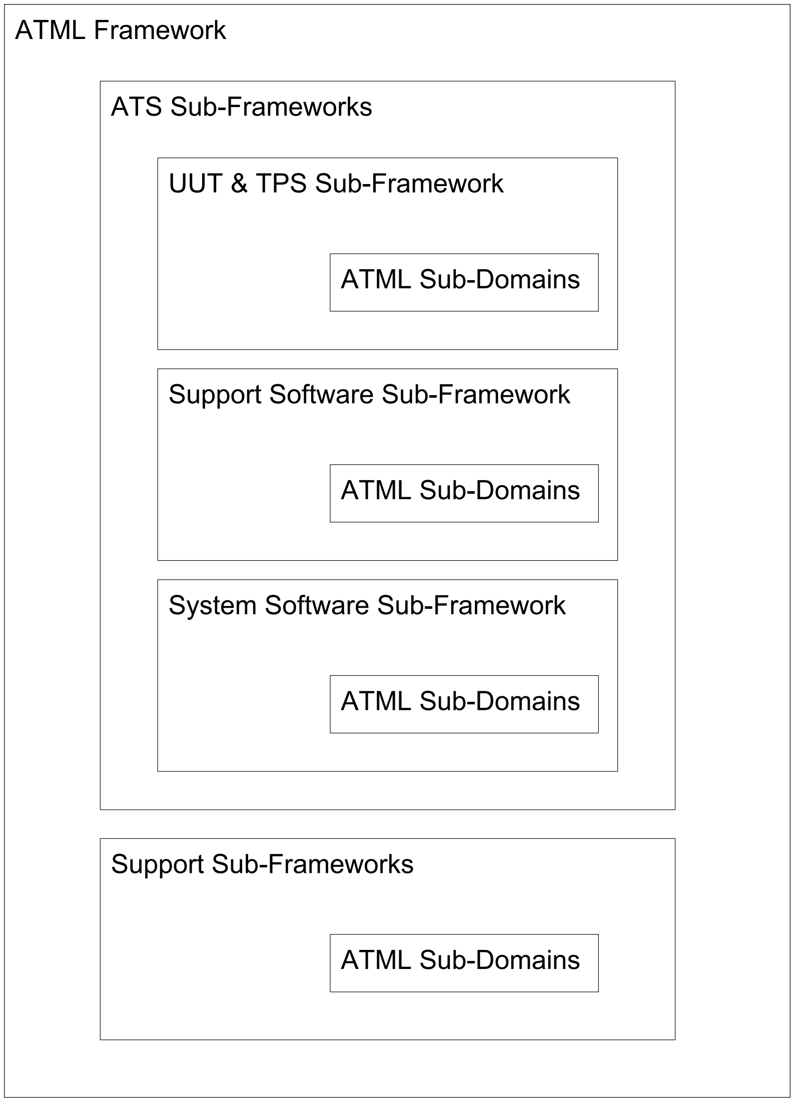


Figure 6—The ATML framework, subframeworks, and ATML subdomains

Figure 7 represents an example of a common element XML type (in this case, a NonBlankString) being inherited by two different XML schemas complex types (in this case, ExampleA and ExampleB), which are then used to develop two separate XML instance documents.

The fact that the XML schemas complex type's attribute is inherited is irrelevant to both the XML instance documents content and to how the XML instance document was or is generated.

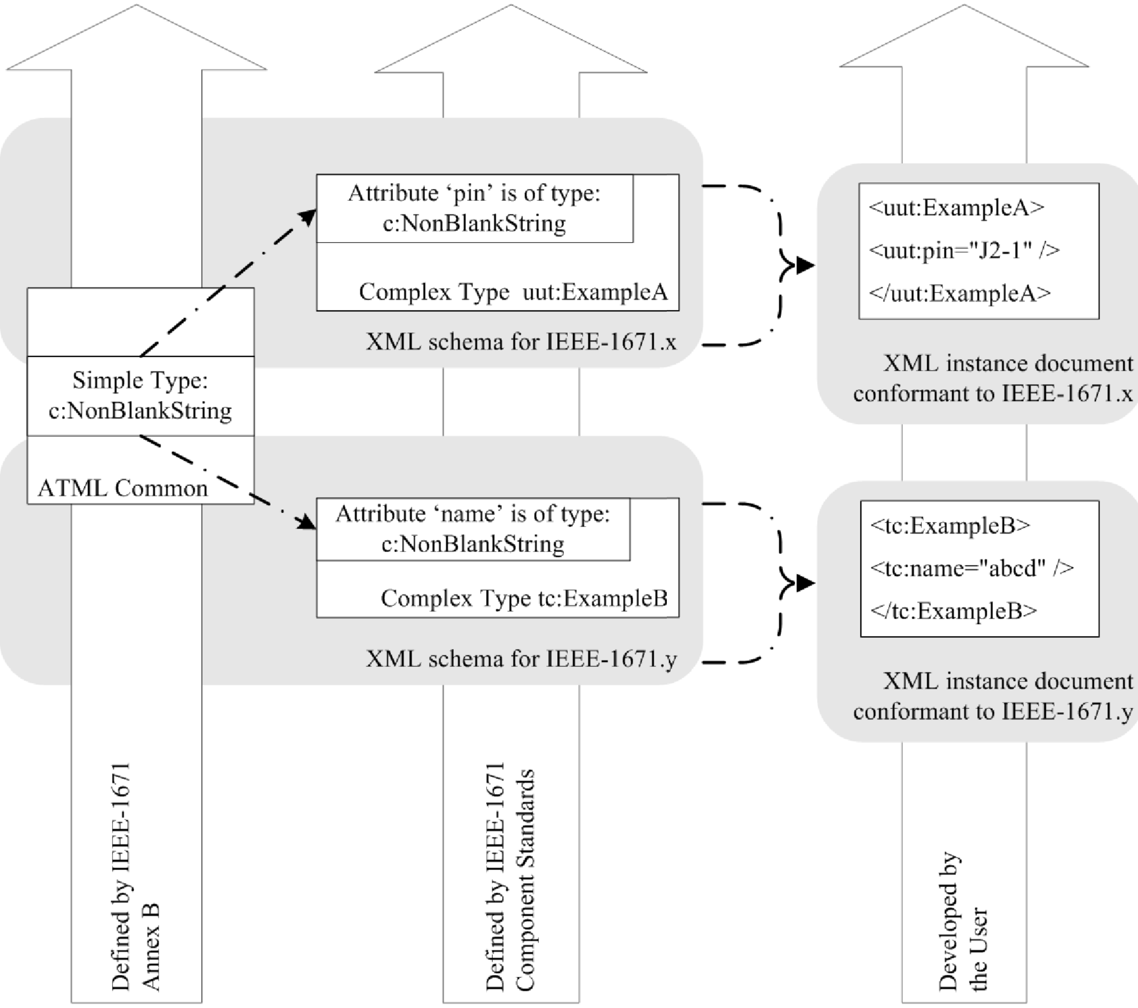


Figure 7—Example ATML common element usage

ATML defines three common element XML schemas:

a) **Common.xsd**, which provides ATML-unique types and attributes.

b) **HardwareCommon.xsd**, which provides ATML-hardware-unique types and attributes as well as includes Common.xsd.

c) **TestEquipment.xsd**, which provides test-equipment-unique types and attributes as well as includes both Common.xsd and HardwareCommon.xsd.

These three common element XML schemas are defined in B.1, B.2, and B.3, respectively.

Every ATML subdomain component XML schema (with the exception of Common.xsd) shall include at least one of the three common element XML schemas.

7.2 ATML XML schemas

ATML defines a collection of XML schemas that allow ATE and test information to be exchanged in a common format adhering to the XML standard. These XML schemas have been developed, and continue to be maintained, through the use of modeling tools and use cases.

NOTE—It is expected that ATML users will utilize one or more XML-based tools to aid in the development of XML instance documents and/or to graphically view the XML schemas.

7.3 XML schemas and their use in ATML

The XML language focuses on the definition of entities, which are the objects of interest. The entities are defined in terms of their elements and attributes, which are the traits or characteristics considered to be important for using and understanding the entities. These elements or attributes have a representation, which might be a simple data type (such as integer) or another entity type. The XML schema also specifies constraints, rules, and relationships between entities.

ATML uses XML schemas to precisely specify the data that can reside in an ATML framework. XML schemas shall be specified for the categories of test information where different sets of data can be instantiated and exchanged between ATML implementations (as depicted in Clause 7 and defined in Clause 8). Test information that conforms to the ATML family XML schemas should be accessed and manipulated by software tools in an ATML test environment. The guidelines for the development of ATML XML schemas are provided in Annex A.

NOTE—**Some constraints cannot be represented by an XML schema; consequently these constraints are specified in the corresponding ATML family standard's textual content (e.g., the published standard)**. Thus, the sources for the complete set of requirements are the ATML family of standards and their associated XML schemas. Moreover, **validation of instance documents against XML schemas does not guarantee that the instance documents satisfy ATML compliance requirements; additional compliance verification may be necessary** for constraints that are not expressed in the XML schema.

7.4 UML models

This document includes, in Annex J, informational unified modeling language (UML)[9] models. One represents a generic ATS testing an UUT (Figure J.1). The second (represented by Figure J.2, Figure J.3, Figure J.4, and Figure J.5) represents the relationship between components of an ATS and UUT.

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

8. The ATML framework subdomains

Each of the ATML subdomains' external interfaces (see 6.1) is formally defined by an ATML family component standard (e.g., IEEE 1671 series "dot" standard). The ATML subdomain internal models and common element schemas are defined within this document (see 6.2 and 7.1), which is also an ATML family component standard.

XML instance documents of these ATML family component standards make up the core elements of an ATML framework. Figure 8 portrays the ATML family of standards (which would actually be XML instance documents valid against the associated family component) making up a fully populated ATML framework.

8.1 The ATML framework and ATML family component standards

The ATML family component standards define the external interfaces (see 6.1), internal models (see 6.2), and common elements (see 7.1). ATML family component standards are segmented into the following:

a) The ATML family component standard document (in the form of a formal IEEE standard)

b) The ATML family component's associated XML schemas

c) References to ATML instance documents (however, specific instance documents shall not be part of the ATML framework)

8.1.1 ATML family component standards

The ATML family components each shall have an associated IEEE published standard. Each of these standards shall contain the definition, description, and use of each element of the ATML family component as well as define the conformance to that standard.

8.1.2 ATML family XML schemas

The ATML family of standards shall have associated XML schemas. XML schemas are described in 7.2.

These XML schemas shall be located on the World Wide Web at the locations specified in Clause 9.

8.1.3 ATML instance documents

ATML instance documents are a collection of specific information defined and organized by the referenced XML schema (e.g., a particular instrument's instance document shall contain the definition of the particular instrument, in accordance with the instrument description XML schema specification).

8.2 ATML subdomains

The ATML family of standards (along with their associated XML schemas) defines a logically related set of ATML information (e.g., ATML subdomain). These ATML family component standards elaborate on information that only appears as place holders in this document. The ATML family of standards is defined in Table 3.

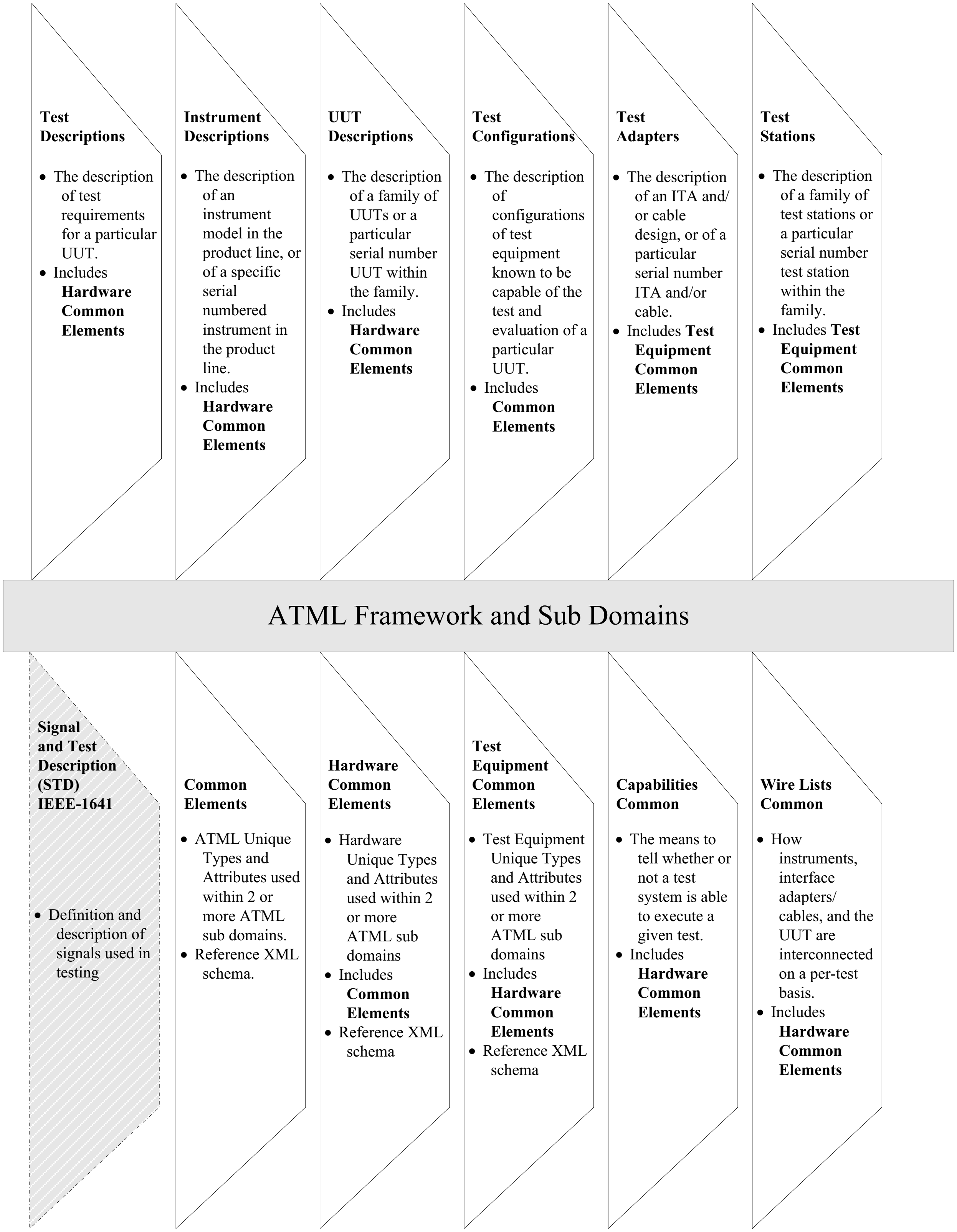


Figure 8—The ATML framework and subdomains

Table 3—ATML subdomains

Sub-domain name	Brief description	Standard
Standard Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML	This standard.	IEEE Std 1671 (ATML)
Capabilities	Annex F of this standard contains the necessary information to allow software to determine whether a given test system can run a given test. C.1 of this standard contains the Capabilities XML schema definition.	IEEE Std 1671 (ATML)
Common Elements	B.1, B.2, and B.3 of this standard contain the shared type definitions utilized within two or more ATML family components. There are three common element schemas: Common, HardwareCommon, and TestEquipment.	IEEE Std 1671 (ATML)
Instrument Description	Provides for the description of an test instrument.	IEEE Std 1671.2™ [B32] (ATML: Instrument Description)
Ports, Pins, Connectors, and Wire Lists	Annex E of this standard describes instruments, test systems, and their capabilities at the instruments' pins as "ports, pins connectors" and describes how ATS elements are interconnected as "wire lists." C.2 of this standard contains the WireLists XML schema definition.	IEEE Std 1671 (ATML)
Signal Definitions	Provides for the definition of a test.	IEEE Std 1641 [B29] (Signal and Test Definition)
Test Adapter	Provides for the description of an ITA and/or associated cables and other interface hardware.	IEEE Std 1671.5™ [B35] (ATML: Test Adapter)
Test Configuration	Provides for the description of the testing configuration.	IEEE Std 1671.4™ [B34] (ATML: Test Configuration)
Test Description	Provides for the description of the test subjects test requirements and a default test flow.	IEEE Std 1671.1 [B31] (ATML: Test Description)
Test Station	Provides for the description of a test station.	IEEE Std 1671.6 [B36] (ATML: Test Station)
UUT Description	Provides for the description of a test subject.	IEEE Std 1671.3™ [B33] (ATML: UUT Description)

8.2.1 Capabilities

An ATML framework may require a means to tell whether a test system is able to execute a given test.

ATML Capabilities provides a mechanism to allow tests to be mapped onto instruments (and test systems) in a way that makes it possible to tell whether a test system is able to execute a given test.

Any signal descriptions within a Capabilities description should be represented utilizing the signal and test definition (STD) standard (see IEEE Std 1641 [B29]).

ATML Capabilities is defined in Annex F. The Capabilities XML schema is defined in C.1.

In the event ATML Capabilities is found to be insufficient or an error is identified, a change proposal to C.1 of this document should be directed to the Secretary, IEEE-SA Standards Board.

8.2.2 Common elements

Common elements provide for the definition of XML types and attributes that are utilized within more than one of the ATML family XML schemas. Common elements are described in 7.1.

In the event ATML Common is found to be insufficient or an error is identified, a change proposal to Annex B of this document should be directed to the Secretary, IEEE-SA Standards Board.

8.2.3 Instrument description

An ATML framework may require the description of an instrument model (e.g., a particular company's digital multimeter (DMM) model 123) or a specific occurrence of the instrument in the product line (e.g., a particular company's DMM model 123, serial number 1). Additionally, an ATML framework implementation may include a synthetic arbitrary waveform generator (ARB), synthetic digitizer, synthetic up-converter, or synthetic down-converter. Synthetic instrumentation (SI) requires descriptions like any other instrument.

The Instrument Description standard defines an exchange format for the static description of an nstrument. Instances of Instrument Description will be utilized in conjunction with instances of other Instrument Description in support of the execution of test programs in an automatic test environment. The standard promotes and facilitates interoperability between components of automatic test systems (e.g., between a test executive and a resource allocator) where instrument descriptions need to be shared.

SI is part of IEEE Std 1671.2 [B32], as both an example of Instrument Description instances as well as to provide a definition of the necessary parameters/attributes to document an SI. Template instance documents shall be used by vendors developing/providing SI, as the basis for documenting the SI. The XML template instance document provides examples for each instrument vendor to follow.

Any signal descriptions within an Instrument Description should be represented utilizing STD (see IEEE Std-1641 [B29]).

In the event ATML Instrument Description is found to be insufficient or an error is identified, a change proposal to IEEE Std 1671.2 [B32] and/or Annex B of this document should be directed to the Secretary, IEEE-SA Standards Board.

8.2.4 Ports, pins, connectors, and wire lists

An ATML framework may require a means to describe instruments, test systems, and their capabilities at the instruments' pins. ATML ports, pins, and connectors provide an explanation of the techniques used to map the capabilities to the instruments' pins.

ATML ports, pins, and connectors are described in Annex E.

An ATML framework may require a means to describe how instruments, test systems, interface adapters, and UUTs are interconnected. ATML wire lists provide an explanation of the techniques used to describe these interconnections. The interconnections are typically defined on a test-by-test basis, the test being an ATML Test Description document definition. Therefore, there shall be a direct mapping between wire list and test description.

ATML wire lists are described in Annex E. The WireLists XML schema is defined in C.2.

In the event ATML WireLists is found to be insufficient or an error is identified, a change proposal to C.2 of this document should be directed to the Secretary, IEEE-SA Standards Board.

8.2.5 Signal definitions

IEEE Std 1641 [B29] is a culmination of a radical review of the ATLAS test programming language and the requirement to create truly portable test requirements. STD allows test information to pass more freely between the design, test, and maintenance phases of a project and enables the same information to be used directly across project phases. This more efficient use of information will lead to reduced life-cycle costs. STD provides the capability to describe and control signals, while permitting a choice of operating environment, including the choice of carrier language. STD permits signal operations to be embedded in any object-oriented environment and thus to be used by the architecture standards of various ATSs. Portability is extended beyond that of test specifications by virtue of a layered architecture. STD defines a collection of objects and their interfaces. These objects describe signal components relevant to test requirements. The STD standard defines how to interconnect these objects using interfaces through which the objects exchange information so that a test model may be defined that describes actual test requirements. Finally, the link to published ATLAS standards (such as IEEE Std 716™-1995 [B12]) is preserved in that the user can describe signal operations using very similar test-signal-related keywords. These keywords now have formal definitions. Furthermore, the parameters of the signals themselves also have a rigorous formal behavioral description.

Signal definitions are defined by IEEE Std 1641 [B29].

8.2.6 Test adapter

An ATML framework may require the description of an ITA and/or cable design or of a particular serial number ITA and/or cable. ITAs (also sometimes referred to as interface devices or IDs) are the interface between the UUT and the Test Station. Cables are the interface between the UUT, the Test Station, and ITA. What typically needs to be documented is the physical and electrical characteristics, the capabilities/performance, the identification and classification, etc. This information includes the connectors, wires, contacts, etc.

IEEE Std 1671.5 [B35] defines an exchange format for exchanging the test adapter information by defining the interface between the UUT and the test station, which includes the description of the test adapter, test adapter and/or ancillary cables[10], and any ancillary equipment required to interface the UUT to the ATE, in order to perform any test(s) on the subject UUT. These descriptions include the physical and electrical characteristics, capabilities/performance, and identification/classification. The standard provides a standardized format to promote and facilitate interoperability between components of automatic test systems by allowing the exchange of test adapter information. Each instance document contains the definition of a single test adapter or cable model. The test adapter schemas provide a structure for describing test adapter capabilities and structure.

Any signal descriptions within a Test Adapter description should be represented utilizing STD (see IEEE Std 1641 [B29]).

In the event ATML Test Adapter Description is found to be insufficient or an error is identified, a change proposal to IEEE Std 1671.5 [B35] and/or Annex B of this document should be directed to the Secretary, IEEE-SA Standards Board.

[10] When utilizing the Test Adapter XML schema for the description of any cable(s), the length of the cable is represented by depth.

8.2.7 Test configuration

An ATML framework may require the description of configurations of test equipment known to be capable of the test and evaluation of a particular UUT.

IEEE Std 1671.4 [B34] defines an exchange format for identifying all of the hardware, software, and documentation that may be used to test and diagnose a UUT on an ATS. The data support the acquisition and itemization of test assets required to be in place prior to testing a UUT on the test system.

In the event ATML Test Configuration is found to be insufficient or an error is identified, a change proposal to IEEE Std 1671.4 [B34] and/or Annex B of this document should be directed to the Secretary, IEEE-SA Standards Board.

8.2.8 Test description

An ATML framework may require the description of tests requirements for a particular UUT.

IEEE Std 1671.1 [B31] defines an exchange format for exchanging the test description information defining test performance, test conditions, diagnostic requirements, and support equipment to locate, align, and verify the proper operation of a UUT. This information shall be utilized in the development of TPSs that will be ultimately used in an automatic test environment. The standard promotes and facilitates interoperability between components of ATSs (e.g., rehosting test requirements between ATS platforms) where UUT test requirement definitions need to be shared.

Any signal descriptions within a Test Description should be represented utilizing STD (see IEEE Std 1641 [B29]).

In the event ATML Test Description is found to be insufficient or an error is identified, a change proposal to IEEE Std 1671.1 [B31] and/or Annex B of this document should be directed to the Secretary, IEEE-SA Standards Board.

8.2.9 Test station

An ATML framework may require the description a family of test stations or a particular test station within the family (e.g., by serial number). This description includes the physical and electrical characteristics; the paths between test system ports and the instrument; tolerances and accuracy of the test station; test station identification information such as part number, serial number, nomenclature, location; status information such as calibration data, dates, and self-test status; operational history, such as power-on time; external interfaces; power requirements; controller definitions; etc.

IEEE Std 1671.6 [B36] defines an exchange format for exchanging the test station information by defining the description of the test station (e.g., physical and electrical characteristics, components, capabilities/performance, identification/classification). The standard provides a standardized format to promote and facilitate interoperability between components of automatic test systems by allowing exchange of test station information. Each instance document contains the definition of a single test station model. The Test Station XML schema provides a structure for describing test station capabilities and structure.

Any signal descriptions within a Test Station description should be represented utilizing STD (see IEEE Std 1641 [B29]).

In the event ATML Test Station is found to be insufficient or an error is identified, a change proposal to IEEE Std 1671.6 [B36] and/or Annex B of this document should be directed to the Secretary, IEEE-SA Standards Board.

8.2.10 UUT description

An ATML framework may require the description of a family of UUTs or a particular UUT within the family (e.g., by serial number). This description includes information such as the name, part number, model number, type, power requirements, interfaces, physical properties, and operational requirements, i.e., the information about a UUT that is required to implement and execute tests on and diagnose the UUT itself.

IEEE Std 1671.3 [B33] defines an exchange format for information that uniquely describes a category or type of UUT. The format will include the ability to specify multiple manufacturers for each UUT, as there may be cases where a single UUT is supplied by a variety of manufacturers. This information is intended to support all aspects of the test and maintenance environment. The standard promotes and facilitates interoperability between components of test and maintenance support systems by defining a common set of identification information for UUTs.

In the event ATML UUT Description is found to be insufficient or an error is identified, a change proposal to IEEE Std 1671.3 [B33] and/or Annex B of this document should be directed to the Secretary, IEEE-SA Standards Board.

9. ATML XML schema names and locations

The IEEE provides a download Web site for material associated with published IEEE standards; the material is presented in machine-friendly format. This material is digital rights management restricted use material. The ATML family of standards utilizes this download Web site to allow easy accessibility to all of the ATML family XML schemas (and in some cases, example XML instance documents). As depicted by Figure 9, the IEEE download Web site (http://standards.ieee.org/downloads/) contains several folders, each folder labeled by an associated IEEE standards number (e.g., IEEE 1671 series standards are in the 1671 folder). Each folder under this base IEEE standard number contains the material (e.g., XML schemas) for that ATML family component standard. ATML family component standards are identified by their IEEE 1671 series 'dot' standard number and the year in which that standard was published by the IEEE.

NOTE 1—Standards that are revised will contain a folder for the year in which the standard is reissued. Both folders (for each year the standard was published) will be present on the IEEE download Web site.
NOTE 2—Providing a particular standard has associated material that is to be made available via the download Web site, folders for that standard are not available until the standard is published by the IEEE.

Figure 9 depicts a portion of the entire IEEE download Web site, as it pertains to the ATML family of standards.

The IEEE SCC20 TII Subcommittee's Web site (http://grouper.ieee.org/groups/scc20/tii) provides access to material not yet published as an IEEE standard. This material is also digital rights management restricted use material. The ATML family of standards utilizes this site to allow easy accessibility to any of the ATML family XML schemas (and in some cases, example XML instance documents) not yet approved by the IEEE Standards Board.

The ATML family component standards (where the component is defined), their associated XML schemas' names, and the IEEE download Web site folder name (where the XML schemas shall be located) are as defined in Table 4.

Each of the XML schemas identified in Table 4 and Table 6 may include one or more of the ATML common element XML schemas defined in Annex A. The ATML common element (e.g., component) (where the component is defined), the associated XML schema's name, and the IEEE download Web site folder name (where the XML schema shall be located) are as defined in Table 5.

The XML schemas identified in Table 4 may utilize one or more of the ATML common XML schemas defined in Annex C. The ATML common schema (e.g., component) (where the component is defined), the associated XML schema's name, and the IEEE download Web site folder name (where the XML schema shall be located) are as defined in Table 6.

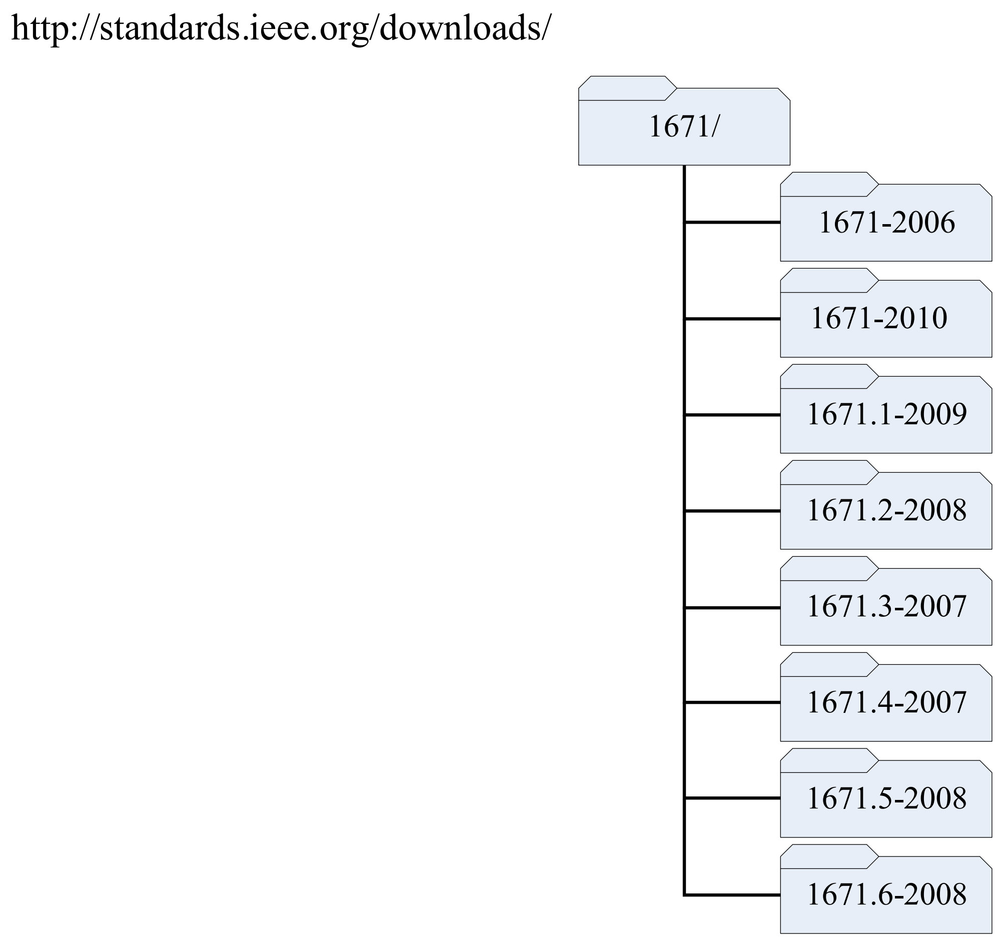


Figure 9—ATML-related IEEE download Web site structure

Table 4—ATML family XML schema names and folder locations

Component	Defined in	XML schema name	IEEE download Web site folder (see Figure 9)
Instrument Description	IEEE Std 1671.2	InstrumentDescription.xsd	1671.2-2008
		InstrumentInstance.xsd	1671.2-2008
		Digitizer.xml	1671.2-2008
		DownConverter.xml	1671.2-2008
		SyntheticWaveformGenerator.xml	1671.2-2008
		UpConverter.xml	1671.2-2008
Test Adapter	IEEE Std 1671.5	TestAdapterDescription.xsd	1671.5-2008
		TestAdapterInstance.xsd	1671.5-2008
Test Configuration	IEEE Std 1671.4	TestConfiguration.xsd	1671.4-2007
Test Description	IEEE Std 1671.1	TestDescription.xsd	1671.1-2009
Test Station	IEEE Std 1671.6	TestStationDescription.xsd	1671.6-2008
		TestStationInstance.xsd	1671.6-2008
UUT Description	IEEE Std 1671.3	UUTDescription.xsd	1671.3-2007
		UUTInstance.xsd	1671.3-2007

Table 5—ATML Common element XML schema names and locations

Component	Defined in	XML schema name	IEEE download Web site folder (see Figure 9)
Common	B.1	Common.xsd	1671-2010
Hardware Common	B.2	HardwareCommon.xsd	1671-2010
Test Equipment	B.3	TestEquipment.xsd	1671-2010

Table 6—ATML Common XML schema names and locations

Component	Defined in	XML schema name	IEEE download Web site folder (see Figure 9)
Capabilities	C.1	Capabilities.xsd	1671-2010
Wire Lists	C.2	WireLists.xsd	1671-2010

10. ATML XML schema extensibility

The provision of an extension mechanism is necessary to ensure the viability of the specification and allow producers and consumers of ATML instance documents to interoperate in cases where there is a requirement to exchange relevant data that are not included in the ATML family XML schemas. The use of the extensions shall be done in a way that ensures that a conformant consumer can utilize the extended file without error, discard or otherwise sidestep the extended data, and use the nonextended portions of the data as they are intended, without error or loss of functionality.

Extensions shall be additional information added to the content model of the element being extended.

Extensions shall not repackage existing information entities that are already supported by the ATML family of standards.

Extensions shall always be associated with a user-defined namespace and should be identified with a namespace prefix (see Table A.1).

An extended instance document shall be accompanied by the extension XML schema and documentation sufficient to explain the need for the extension as well as the underlying semantics and relationship(s) to the base ATML family XML schema.

The ATML family XML schemas allow for three forms of extension:

a) Wildcard-based extensions allow for the extension of the XML schemas with additional elements.

b) Type derivation allows for extending the set of data types by deriving a new type from an existing common element type.

c) Lists derived from `c:NamedValues` allowing user-defined properties with attached values.

The ATML family XML schemas control the location and type of extension allowed. A.6.7 describes how to specify the extension points for an ATML family XML schema.

11. Conformance

ATML conformance has two facets. The first (11.1) shall apply only to the development and maintenance of the ATML family of standards and their associated XML schemas. The second (11.2) shall apply only to implementers and implementations of ATML (this standard).

11.1 ATML family XML schemas

Each of the ATML family XML schemas shall be developed and maintained to be conformant with the XML schema style guidelines defined in Annex A.

11.2 The ATML framework

Conformance to the ATML framework shall be achieved as specified by the requirements defined by either of the following:

a) Any combination of Table 7, and/or Table 8, and/or Table 9 (see 11.2.1 through 11.2.1.3)

b) Table 10 (see 11.2.2)

For each ATML family component standard utilized in either item a) or item b), the conformance requirements of the utilized component standard shall also be satisfied.

11.2.1 ATS subframework

Figure 10 illustrates three ATS subframeworks that may be incorporated within an ATS:

a) UUT & TPS (see 11.2.1.1)

b) ATE support software (see 11.2.1.2)

c) ATE System software (see 11.2.1.3)

An ATS shall include at least one of these subframeworks and may include all three.

Each of the ATS subframework components utilize ATML Common element schemas (see 7.1 and Table 5). The same ATML Common element schemas should be used for each ATS framework component; however, different versions of Common element schemas are allowed. It is also allowable to edit ATML family XML schemas, solely for the purpose of permitting the use of the same Common element schema version, within an ATS framework.

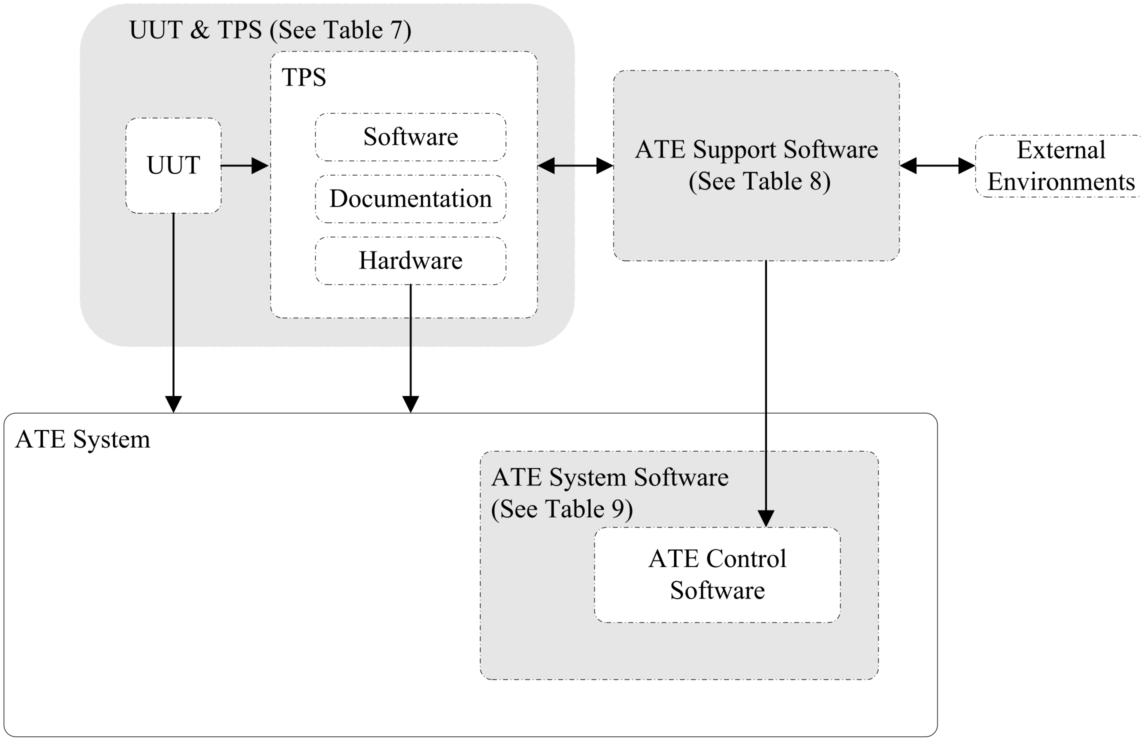


Figure 10 —ATML ATS subframework

11.2.1.1 ATML UUT & TPS subframework

Conformance to an ATML TPS subframework shall be achieved as defined in Table 7.

Table 7— ATML UUT & TPS subframework conformance table

#	Requirement	Clause	Requirement type		Compliance
			shall	should	
1	The specification / definition of a signal.	8.2.5		●	IEEE Std 1641 [B29] (i.e., STD) should be utilized where signal descriptions are to be included.
2	Common types and attributes that are used by more than one of the XML schemas.	8.2.2	●		IEEE Std 1671-2010 B.1, B.2, and B.3 shall be utilized as required by the XML schemas being used.
3	The description of an ITA and/or cable set design, or the specific occurrence a particular ITA and/or cable set.	8.2.6	○ [a]		Should the description of an ITA be included, IEEE Std 1671.5 [B35] shall be utilized.
4	Configurations of test equipment known to be capable of the test and evaluation of a particular UUT.	8.2.7		●	Should test configuration be included, IEEE Std 1671.4 [B34] should be utilized.

Table 7 — ATML UUT & TPS subframework conformance table *(continued)*

#	Requirement	Clause	Requirement type		Compliance
			shall	should	
5	The definition of test performance, test conditions, diagnostic requirements, and support equipment needed to verify the proper operation of a UUT.	8.2.8	○ [a]		Should test descriptions be included, IEEE Std 1671.1 [B31] shall be utilized.
6	The description of a family of UUTs or the specific occurrence of a particular UUT.	8.2.10		●	Should UUT descriptions be included, IEEE Std 1671.3 [B33] should be utilized.
7	XML schema names and IEEE folder locations.	Clause 9	●		The XML schemas utilized shall originate from the locations defined in Table 2, Table 3, and Table 4.
8	Extensibility	Clause 10	●		Clause 6 shall be strictly adhered to.

[a] A conforming UUT & TPS subframework shall include either requirement #3 or #5 and could include both.

11.2.1.2 ATML ATE support software subframework

Conformance to an ATML ATE support software subframework shall be achieved as defined in Table 8.

Table 8 — ATML ATE support software subframework conformance table

#	Requirement	Clause	Requirement type		Compliance
			shall	should	
1	Determining whether a test system is able to execute a given test.	8.2.1		●	Should capabilities be included, IEEE Std 1671 C.1 should be utilized.
2	Describe how ATS elements are interconnected.	8.2.4		●	Should how ATS elements are interconnected (either by test or in its entirely) be included, IEEE Std 1671 C.2 should be utilized.
3	Common types and attributes that are used by more than one of the XML schemas.	8.2.2	●		IEEE Std 1671 B.1, B.2, and B.3 shall be utilized as required by the XML schemas being used.
4	The description of an instrument model or the specific occurrence of the instrument.	8.2.3	○ [a]		Should instrument descriptions be included, IEEE Std 1671.2 [B32] shall be utilized.
5	SI.	8.2.3		●	Should SI be included, IEEE Std 1671.2 [B32] should be utilized.
6	The description of an ITA and/or cable set design or the specific occurrence a particular ITA and/or cable set.	8.2.6	○ [a]		Should test adapter descriptions be included, IEEE Std 1671.5 [B35] shall be utilized.
7	The description of a family of test stations or the specific occurrence of a particular test station.	8.2.8	○ [a]		Should test station descriptions be included, IEEE Std 1671.6 [B36] shall be utilized.

Table 8 — ATML ATE support software subframework conformance table *(continued)*

#	Requirement	Clause	Requirement type		Compliance
			shall	should	
8	XML schema names and IEEE folder locations.	Clause 9	●		The XML schemas utilized shall originate from the locations defined in Table 2, Table 3, and Table 4.
9	Extensibility.	Clause 10	●		Clause 6 shall be strictly adhered to.

[a] A conforming ATE support software subframework shall include requirement #4 or #6 or #7 and could include more than one.

11.2.1.3 ATML system software subframework

Conformance to an ATML ATE system software subframework shall be achieved as defined in Table 9.

Table 9—ATML ATE system software subframework conformance table

#	Requirement	Clause	Requirement type		Compliance
			shall	should	
1	Determining whether a test system is able to execute a given test.	8.2.1		●	Should capabilities be included, IEEE Std 1671 C.1 should be utilized.
2	Describe how ATS elements are interconnected.	8.2.4		●	Should how ATS elements are interconnected (either by test or in its entirely) be included, IEEE Std 1671 C.2 should be utilized.
3	Common types and attributes that are used by more than one of the XML schemas.	8.2.2	●		IEEE Std 1671 B.1, B.2, and B.3 shall be utilized as required by the XML schemas being used.
4	The description of an instrument model or the specific occurrence of the instrument.	8.2.3	○ [a]		Should instrument descriptions be included, IEEE Std 1671.2 [B32] shall be utilized
5	SI.	8.2.3		●	Should SI be included, IEEE Std 1671.2 [B32] should be utilized.
6	The description of an ITA and/or cable set design or the specific occurrence a particular ITA and/or cable set.	8.2.6	○ [a]		Should test adapter descriptions be included, IEEE Std 1671.5 [B35] shall be utilized.
7	Configurations of test equipment known to be capable of the test and evaluation of a particular UUT.	8.2.7	○ [a]		Should test configuration be included, IEEE Std 1671.4 [B34] should be utilized.
8	The description of a family of test stations or the specific occurrence of a particular test station.	8.2.9	○ [a]		Should test station descriptions be included, IEEE Std 1671.6 [B36] shall be utilized.
9	XML schema names and IEEE folder locations.	Clause 9	●		The XML schemas utilized shall originate from the locations defined in Table 2, Table 3, and Table 4.
10	Extensibility	Clause 10	●		Clause 6 shall be strictly adhered to.

[a] A conforming ATE system software subframework shall include requirement #4 or #6 or #7 or #8 and could include more than one.

11.2.2 ATML support subframework

The ATML support subframework encompasses the utilization of one or more ATML family component standards outside the formal definition of an ATS. This utilization may be in **support** of ATS elements, such as procuring an instrument, specifying a requirement (e.g., I need a station with the following capabilities), or developing a ATML **tool** or **tools** (e.g., a **tool** to aide in the development of ATML Instrument Description documents based upon the IEEE 1671.2 XML schemas).

ATML family components utilize ATML **common element components**. When more than one ATML family component is included in an ATML support framework implementation, the ATML **common element components shall be identical between the ATML family components,** e.g., when utilizing the ATML Instrument Description component and the Test Station Description component, the same Common.xsd shall be used. **Note that this requirement may require editing of one or more ATML family XML schemas to include the same chosen ATML common element component**.

Figure 11 illustrates the ATML support subframework and its optional interfaces to external inputs and environments.

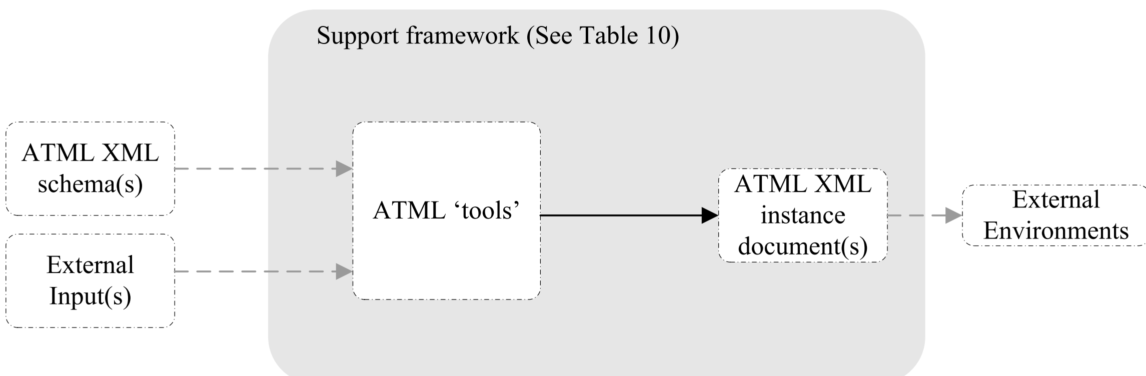


Figure 11 —ATML support subframework

Conformance to an ATML support subframework shall be achieved as defined in Table 10.

Table 10 —ATML support subframework conformance table

#	Requirement	Clause	Requirement type		Compliance
			shall	should	
1	The specification /definition of a signal.	8.2.5		●	IEEE Std 1641 [B29] (i.e., STD) should be utilized where signal descriptions are to be included.
2	Common types and attributes that are used by more than one of the XML schemas.	8.2.2	●		IEEE Std 1671 B.1, B.2, and B.3 shall be utilized as required by the XML schemas being used.
3	The definition of test performance, test conditions, diagnostic requirements, and support equipment needed to verify the proper operation of a UUT.	8.2.8	○ [a]		Should test descriptions be included, IEEE Std 1671.1 [B31] shall be utilized.
4	The description of an instrument model or the specific occurrence of the instrument.	8.2.3	○ [a]		Should instrument descriptions be included, IEEE Std 1671.2 [B32] shall be utilized.
5	SI.	8.2.3		●	Should SI be included, IEEE Std 1671.2 [B32] should be utilized.
6	The description of a family of UUTs or the specific occurrence of a particular UUT.	8.2.10	○ [a]		Should UUT descriptions be included, IEEE Std 1671.3 [B33] shall be utilized.
7	Configurations of test equipment known to be capable of the test and evaluation of a particular UUT.	8.2.7	○ [a]		Should test configurations be included, IEEE Std 1671.4 [B34] shall be utilized.
8	The description of an ITA and/or cable set design or the specific occurrence a particular ITA and/or cable set.	8.2.6	○ [a]		Should ITAs be included,IEEE Std 1671.5 [B35] shall be utilized.
9	The description of a family of test stations or the specific occurrence of a particular test station.	8.2.9	○ [a]		Should test station descriptions be included, IEEE Std 1671.6 [B36] shall be utilized.
10	XML schema names and IEEE folder locations.	Clause 9	●		The XML schemas utilized shall originate from the locations defined in Table 2, Table 3, and Table 4.
11	Extensibility	Clause 10	●		Clause 6 shall be strictly adhered to.

[a] An conforming support subframework shall include requirement # 3 or #4 or #6 or #7 or #8 or #9 and can include more than one.

Annex A

(normative)

XML schema style guidelines

XML is a simple, flexible text format derived from SGML (ISO 8879:1986 [B39]). The W3C created, developed, and continues to maintain the XML specifications (see *Extensible Markup Language (XML) 1.0*, *Namespaces in XML 1.0* [B43], W3C Technical Reports and Publications [B56], XHTML 1.1 [B57], XSD 1.1 Part 1 [B58], and XSD 1.1 Part 2 [B59]).

The style guidelines presented in this annex shall be followed during the development or maintenance of each of the XML schemas associated with this document (e.g., the schemas defined in Annex B and Annex C) and the schemas associated with each of the ATML family of standards.

A.1 Naming conventions

A.1.1 Capitalization conventions

A.1.1.1 Pascal case

The first letter in the identifier and the first letter of each subsequent concatenated word are capitalized.

A.1.1.2 Camel case

The first letter of an identifier is lowercase, and the first letter of each subsequent concatenated word is capitalized.

A.1.1.3 Uppercase

All letters in the identifier are capitalized.

A.1.1.4 Lowercase

All letters in the identifier are lowercase.

A.1.2 Naming guidelines

a) All words shall be spelled using U.S. English in accordance with the latest edition of *Webster's New Collegiate Dictionary* [B55]. The use of abbreviations and acronyms shall be avoided.
    1) As a general rule, acronyms should not be used in XML element and attribute names. When it is necessary to use an acronym, acronyms with three or more characters shall use Pascal case. Acronyms with two characters shall use Uppercase.
    2) Abbreviations shall not be used in XML element and/or attribute names.

3) For XML schema data types, abbreviations shall be avoided while acronyms should not be used.

b) XML element and XML schema data types shall use Pascal case.

c) Except for XML schema abstract data types, XML schema data type names shall not have the word 'Type' appended.

d) XML attributes should use Camel case. There is one exception to this rule: if an element has an ID attribute, that attribute may use the Uppercase naming convention and be of type `NonBlankString` defined in Common.xsd (B.1).

e) Namespace names shall use Pascal case.

f) Namespace prefixes shall use Lowercase.

1) Prefixes for each of the ATML family XML schemas shall as defined in Table A.1[11]:

Table A.1—ATML family XML schema namespaces: sorted by prefix

XML Schema	Prefix
Common.xsd	c:
Capabilities.xsd	ca:
HardwareCommon.xsd	hc:
InstrumentDescription.xsd	inst:
InstrumentInstance.xsd	insti:
IEEE 1641:STDBSC (see NOTE 1)	std:
TestAdapterDescription.xsd	ta:
TestAdapterInstance.xsd	tai:
TestConfiguration.xsd	tc:
TestDescription.xsd	td:
TestEquipment.xsd	te:
TestStationDescription.xsd	ts:
IEEE 1641:STDTSF (see NOTE 2)	tsf:
TestStationInstance.xsd	tsi:
UUTDescription.xsd	uut:
UUTInstance.xsd	uuti:
WireLists.xsd	w:
NOTE 1—STDBSC is the basic signal components (BSCs) layer of IEEE Std 1641 [B29]. NOTE 2—STDTSF is the test signal framework (TSF) layer of IEEE Std 1641.	

g) Name segments should be distinguished by the use of mixed case (instead of underscores).

1) Underscores, periods, and dashes shall not be used in XML element, schema data type, or attribute names.

h) An element that represents a collection shall be named using a plural name.

i) An element that represents a single collection element (or member) shall be named using a singular name.

[11] These prefixes may be used from the date of this document forward. Previously published ATML family trial-use standards, when implemented by ATML framework developers/users, may utilize these prefixes. Should IEEE Std 1641 [B29] be utilized within the ATML framework, the listed IEEE 1641 prefixes may be used.

A.2 XML declaration

All ATML family XML schema and instance documents shall use an explicit XML declaration as the first line of a file. This declaration shall follow the form `<?xml version opt._encoding opt._standalone?>`. In general, it is expected that all ATML documents will use UTF-8 encoding and will not use the standalone option. Thus, the XML declaration for ATML documents shall be

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

A.3 ATML namespaces

A.3.1 Approved XML schema namespaces—IEEE approved standard

The namespace uniform resource name (URN) (see URN Syntax [B51]) for approved XML schemas, which have been published by the IEEE, shall be

```
URN:IEEE-<ieee_standard_number>-<release_year>:<schema_name>
```

where

`<ieee_standard_number>` is the standard number assigned by the IEEE.
`<release_year>` is the year in which the standard and XML schema were approved by the IEEE or developed for the purpose of updating or revising a published IEEE standard.
`<schema_name>` is the XML schema name identified in Clause 9.

A.3.2 Approved XML schema namespaces—IEEE standard in revision

The namespace URN for approved XML schemas, which are associated with an IEEE standard in revision, shall be

```
URN:IEEE-<ieee_standard_number>-<release_year>:<release>:<schema_name>
```

where

`<ieee_standard_number>` is the standard number assigned by the IEEE.
`<release_year>` is the year in which the standard and XML schema were approved by the IEEE or developed for the purpose of updating or revising a published IEEE standard.
`<release>` is an integer that indicates the release number of the XML schema. The release number starts at 01 and increments each time a new release is made available; this approach incorporates invalidating (e.g., breaking) changes from the previous release.
`<schema_name>` is the XML schema name identified in Clause 9.

A.3.3 Preapproved XML schema namespaces

The namespace URN for preapproved (draft, candidate, and recommendation) XML schemas, which have not been previously published as an IEEE standard, shall be

```
URN:P-IEEE-<ieee_standard_number>-<posting_year>:<release>:<schema_name>
```

where

`<eee_standard_number>` is the standard number assigned by the IEEE.
`<posting_year>` is the year in which the prereleased version of the XML schema is made available

<release> is an integer that indicates the release number of the preapproved XML schema. The release number starts at 01 and increments each time a new preapproved release is made available for evaluation; this approach incorporates invalidating (e.g., breaking) changes from the previous release.

<schema_name> is the XML schema name identified in Clause 9.

The namespace shall be modified whenever one of the following conditions occurs:

— A change is made to a XML schema that invalidates existing instance documents (i.e., a major revision update).

— The XML schemas state changes from **preapproved** to **approved**.

— A new **preapproved** version is made available for evaluation.

The use of the XML schema is controlled through their namespaces so that any XML instance document refers to the namespace when describing one of these components.

A.3.4 Target namespace

Every XML schema shall define a target namespace. The namespace shall be defined as a URN as described in A.3. Each ATML family XML schema has its own namespace. This approach provides a standard way to avoid name collisions between XML schemas.

A.3.5 Default namespace

The default namespace shall be the target namespace.

A.3.6 XML schema namespace reference

The namespace prefix for the XML schema namespace shall be **`xs`**

`xmlns:xs:="http://www.w3.org/2001/XMLSchema"`

The XML schema namespace shall not be the default namespace.

A.3.7 Qualified and unqualified

There are two attributes of the `xs:schema` element that shall be specified for every XML schema: `elementFormDefault` and `attributeFormDefault`. These attributes specify whether elements and attributes in XML instance documents need to be qualified with the namespace of the XML schema in which they are defined.

The value of `attributeFormDefault` specifies whether attributes in XML instance documents are qualified with the namespace of the XML schema in which they are defined. Since an attribute is always defined and used in the context of an element, it is not necessary to qualify the attribute as well as the element.

The value of `attributeFormDefault` shall be `unqualified`.

The value of `elementFormDefault` specifies whether elements in XML instance documents are qualified with the namespace of the XML schema in which they are defined. A value of `qualified` indicates that if the root element is qualified, then all subelements must be qualified as well. A value of `unqualified` indicates that only global elements need to be qualified. Using a value of `unqualified` allows for inconsistent qualification of elements in instance documents.

Given the following example XML schema with `elementFormDefault` set to `qualified`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:http://mynamespace.com/MySchema
  xmlns:xs=http://www.w3.org/2001/XMLSchema
  targetNamespace="http://mynamespace.com/MySchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

  <xs:element name="GlobalElement">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ChildElement" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

The following are valid XML instance documents:

```
<?xml version="1.0" encoding="UTF-8"?>
<GlobalElement xmlns="http://mynamespace.com/MySchema">
  <ChildElement/>
</GlobalElement>
```

and

```
<?xml version="1.0" encoding="UTF-8"?>
<my:GlobalElement xmlns:my="http://mynamespace.com/MySchema">
  <my:ChildElement/>
</my:GlobalElement>
```

The following is not a valid XML instance document:

```
<?xml version="1.0" encoding="UTF-8"?>
<my:GlobalElement xmlns:my="http://mynamespace.com/MySchema">
  <ChildElement/>
</my:GlobalElement>
```

The value of `elementFormDefault` shall be `qualified`.

A.4 Versioning

The XML schema version shall be captured in the XML schema using the version attribute of the XML schema element.

The format of the XML schema version shall be `<major>.<minor>`, where the major portion shall always begin at **0** and the minor portion shall be a two-digit number beginning from **00**.

Previously released versions of each XML schema shall be made available on the SCC20 TII Subcommittee's Web site (http://grouper.ieee.org/groups/scc20/tii/).

Changes made to an XML schema fall into two categories:

a) A *non-invalidating change* does not invalidate existing instance documents. In other words, existing instance documents will continue to validate against the new version of the XML schema. Examples include correcting or adding annotation data, adding an optional element, adding an optional attribute, or adding an enumeration item. For this type of change, it is sufficient to increment the `<minor>` portion of the version. While adding optional attributes and elements does not invalidate existing instance documents, new instance documents that take advantage of these new optional elements will not validate against earlier versions of the XML schema even though the namespace has not changed.

b) An *invalidating change* invalidates existing instance documents. In other words, existing instance documents will no longer validate against the new version of the XML schema. Examples include adding required elements or attributes, changing the structure of an element, or renaming an element or attribute. For this type of change, the `<major>` portion of the version must be incremented, and the minor portion of the version will be reset to zero (00). Also in this case, the namespace of the XML schema must be changed by incrementing the `<release>` as described in A.3.2.

A.4.1 Versioning process for non-invalidating change

a) Change the XML schema version number within the XML schema (`<minor>` portion is incremented by 1).

b) Document the change in the XML schema change history.

c) Make the new and previous version of the XML schema available.

A.4.2 Versioning process for an invalidating change

a) Change the namespace.

b) Change the XML schema version number within the XML schema (<major> portion is incremented by 1, <minor> portion is reset to 00).

c) Document the change in the XML schema change history.

d) Make the new and previous version of the XML schema available.

A.4.3 Version process releasing an approved schema

a) Change the namespace to replace `<posting_year>:<release>` with `<release_year>`, and replace `P-IEEE` with `IEEE` for any XML schema being transitioned from a preapproved status (see A.3.2).

b) Make the new and previous version of the XML schema available to the IEEE SCC20 TII Subcommittee.

A.5 Documentation

A.5.1 Documenting the XML schema

The XML annotation element shall be used. The `<xs:annotation><xs:documentation>…</xs:documentation></xs:annotation>` elements shall contain information targeted at human readers of the XML schema. Annotations shall be used to capture semantics, definitions, and other explanatory information.

A.5.2 XML schema annotations

All ATML family XML schema elements and nonobvious attributes should include annotations as a documentation aid.

A.5.3 Acknowledging the XML schema

All ATML family XML schemas shall include the following text near the beginning of the XML schema:

```
<xs:annotation>
  <xs:documentation xml:lang="en">This schema is specified in IEEE <insert number>,
    "<insert title>." This schema is a World Wide Web Consortium (W3C) Extensible Markup
    Language (XML) binding of the ATML component defined in IEEE <insert number>,
    "<insert title>." The purpose of this schema is to allow the creation of IEEE <insert
    number> instance documents. This schema uses the W3C XML Schema definition language
    as the encoding. This allows for interoperability and the exchange of ATML component
    instances between various systems. This schema may be modified and may be
    included in derivative works. Copyright (c) <year> Institute of Electrical and
    Electronics Engineers, Inc. USE AT YOUR OWN RISK
  </xs:documentation>
</xs:annotation>
```

`<insert number>`, `<insert title>`, and `<year>` shall be replaced with the IEEE standard number, the title of the standard, and the year the standard was approved by the IEEE, respectively.

A.6 Design

A.6.1 Element versus type

A XML schema should declare a type and should avoid the declaration of element(s). Declaring a type permits reuse.

A.6.2 Global elements

A XML schema shall define at most one global element. A global element is an element declaration that is an immediate child of the `<schema>` element.

A.6.3 Global element attributes

The global element shall include the attribute group `DocumentRootAttributes` defined in Common.xsd (see B.1).

A.6.4 Type definitions

A XML schema may define one or more global type definitions.

All elements shall be defined using type definitions. This approach maximizes reuse and namespace control.

A.6.5 Global attributes

The use of global attributes should be avoided.

A.6.6 Element versus attribute

As a general convention, elements are the real containers of data. Attributes are used to annotate elements with metadata describing the content of the element. Perhaps the biggest advantage of using element content to represent information in the document and using attributes for annotation is extensibility. The decision to use elements versus attributes should never be made to optimize document size.

A.6.7 Extensibility

An element has an extensible content model if, in instance documents, that element can contain elements and data beyond that specified by the XML schema. ATML family XML schemas should explicitly identify where they can be extended. Only elements from a namespace different from the document namespace shall be allowed in an extension. The XML schema shall use the ATML Common `<Extension>` type to identify where extension is allowed.

Allowing the extension of a XML schema using type substitution should be avoided. Schemas should mark elements defined via a simple or complex type with the `block` attribute set to `#all` if type substitution is to be avoided. Elements that use type substitution as their means of definition should set the `abstract` attribute to `true`.

A.6.8 Defining uniqueness and references

When defining a XML schema for which validation of references is desired, `xs:key` and `xs:keyref` shall be used instead of `xs:ID` and `xs:IDREF`.

When defining a XML schema for which validation of unique identifiers is desired, **`xs:unique`** shall be used instead of `xs:ID`.

These requirements arise from the fact that there is no limitation on the values or types that can be used as part of an identity constraint that uses `xs:unique`, `xs:key`, and `xs:keyref`, whereas `xs:ID` can be only of a specific range of values (for example, 7 is not a valid `xs:ID`). In addition, the scope of `xs:ID` and `xs:IDREF` is the entire document. The scope of `xs:unique`, `xs:key`, and `xs:keyref` is the target scope of the XPath expression included in the `xs:keyref` definition.

A.6.9 Default and fixed values

Default or fixed values should not be specified for attributes.

A.6.10 Collections

A collection is a list item of the same type. When specifying a collection, a containing element should be included. The `minOccurs` attribute of the containing element should be set to 1 if the collection is required and set to 0 if the collection is optional. The `maxOccurs` attribute of the containing element should always be set to 1. This value implies that if the containing element exists, then the collection has at least one item.

The following is an example of the recommended method for defining a collection of items of the same type:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:http://mynamespace.com/MySchema
  xmlns:xs=http://www.w3.org/2001/XMLSchema
```

```
  targetNamespace="http://mynamespace.com/MySchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

  <xs:element name="MyElement">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Items" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Item" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="OtherElement"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

The above XML schema validates the following XML snippet:

```
<MyElement xmlns="http://mynamespace.com/MySchema">
  <Items>
    <Item/>
    <Item/>
    <Item/>
  </Items>
  <OtherElement/>
</MyElement>
```

Further, the `minOccurs` and `maxOccurs` values of an `xs:sequence` element in an XML schema should be set to 1, the default value.

A.6.11 minOccurs and maxOccurs

The default value for both of these attributes is 1. If the default value is to be used, the attributes should not be explicitly set.

A.6.12 Additions to ATML family XML schemas

Effective with the date of publication of this document, any addition to an ATML family of standards XML schema shall be optional. This requirement assures ATML implementations prior to the date of this document continue to be valid.

Annex B

(normative)

ATML common element schemas

Should the reader not have a general understanding of XML schemas, a XML Schema Tutorial [B60] is available for reference. This tutorial will help with the understanding of the contents of this annex as well as the ATML Common, ATML HardwareCommon, ATML TestEquipment, ATML Capabilities, and ATML WireLists XML schemas for which Annex B and Annex C define the elements.

These ATML common XML schemas may utilize IEEE Std 1641 [B29] for all signal descriptions. When utilized, IEEE Std 1641 shall be referenced for a complete understanding of the ATML common XML schemas and the implementation of any ATML family component standard that includes one or more of the ATML common XML schemas.

B.1 Common element schema—Common.xsd

target namespace	urn:IEEE-1671:2010:Common
version	3.17
imported schema	—

A standard XML schema document (XSD) intended as the source of an instance XML document shall contain a single root element. The **Common XML schema is a reference XML schema containing only type definitions** that may be used in other XML schemas. **It has no root element, and there will be no instance documents directly validated against the Common XML schema**.

B.1.1 Elements

None

B.1.2 Complex types

B.1.2.1 binary

Base type: Extension of *c:DatumType*

Properties: base *c:DatumType*

The *binary* complex type shall be the "`xsi:type`" of any element of type *c:DatumType* that contains a binary value.

B.1.2.1.1 Attributes

binary contains the following attribute, in addition to those inherited from *c:DatumType* (*nonStandardUnit*, *standardUnit*, and *unitQualifier*):

Name	Type	Description	Use
value	xs:string	A finite-length sequence of characters 0 and 1.	Required

B.1.2.1.2 Child elements

binary inherits the child elements of *c:DatumType* (the group *c:DatumQuality*).

B.1.2.2 binaryArray

Base type: Extension of *c:IndexedArrayType*

Properties: base *c:IndexedArrayType*

The *binaryArray* complex type shall be the "`xsi:type`" of any element of type *c:IndexedArrayType* that contains an array of binary values.

B.1.2.2.1 Attributes

binaryArray inherits the attributes of *c:IndexedArrayType* (*dimensions*, *nonStandardUnit*, *standardUnit*, and *unitQualifier*).

B.1.2.2.2 Child elements

binaryArray contains the following child elements, in addition to those inherited from *c:IndexedArrayType* (the group *c:DatumQuality*):

Name	Subclause	Type	Use
DefaultElementValue	B.1.2.3	*c:binary*	Optional
Element	B.1.2.4	*c:binary*	0 …∞

B.1.2.3 binaryArray/DefaultElementValue

Base type: *c:binary*

Properties: isRef 0, content complex

The *binaryArray/DefaultElementValue* child element shall contain the default binary value of the array element.

B.1.2.3.1 Attributes

binaryArray/DefaultElementValue inherits the attributes of *c:binary* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.1.2.3.2 Child elements

binaryArray/DefaultElementValue inherits the child elements of *c:binary* (the group *c:DatumQuality*).

B.1.2.4 binaryArray/Element

Base type: Extension of *c:binary*

Properties: isRef 0, content complex

The *binaryArray/Element* child element shall contain the binary value of the array element.

B.1.2.4.1 Attributes

binaryArray/Element contains the following attribute, in addition to those inherited from *c:binary* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*):

Name	Type	Description	Use
position	*c:ArrayIndexor*	The element value's index within the array.	Required

B.1.2.4.2 Child elements

binaryArray/Element inherits the child elements of *c:binary* (the group *c:DatumQuality*).

B.1.2.5 boolean

Base type: Extension of *c:DatumType*

Properties: base *c:DatumType*

The *boolean* complex type shall be the "`xsi:type`" of any element of type *c:DatumType* that contains a boolean value.

B.1.2.5.1 Attributes

boolean contains the following attribute, in addition to those inherited from *c:DatumType* (*nonStandardUnit*, *standardUnit*, and *unitQualifier*):

Name	Type	Description	Use
value	xs:boolean	A finite-length sequence of characters 0 and 1.	Required

B.1.2.5.2 Child elements

boolean inherits the child elements of *c:DatumType* (the group *c:DatumQuality*).

B.1.2.6 booleanArray

Base type: Extension of *c:IndexedArrayType*

Properties: base *c:IndexedArrayType*

The *booleanArray* complex type shall be the "xsi:type" of any element of type *c:IndexedArrayType* that contains an array of boolean values.

B.1.2.6.1 Attributes

booleanArray inherits the attributes of *c:IndexedArrayType* (*dimensions*, *nonStandardUnit*, *standardUnit*, and *unitQualifier*).

B.1.2.6.2 Child elements

booleanArray contains the following child elements, in addition to those inherited from *c:IndexedArrayType* (the group *c:DatumQuality*):

Name	Subclause	Type	Use
DefaultElementValue	B.1.2.7	*c:boolean*	Optional
Element	B.1.2.8	*c:boolean*	0 …∞

B.1.2.7 booleanArray/DefaultElementValue

Base type: *c:boolean*

Properties: isRef 0, content complex

The *booleanArray/DefaultElementValue* child element shall contain the default boolean value of the array element.

B.1.2.7.1 Attributes

booleanArray/DefaultElementValue inherits the attributes of *c:boolean* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.1.2.7.2 Child elements

booleanArray/DefaultElementValue inherits the child elements of *c:boolean* (the group *c:DatumQuality*).

B.1.2.8 booleanArray/Element

Base type: Extension of *c:boolean*

Properties: isRef 0, content complex

The *booleanArray/Element* child element shall contain the boolean value of the array element.

B.1.2.8.1 Attributes

booleanArray/Element contains the following attribute, in addition to those inherited from *c:boolean* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*):

Name	Type	Description	Use
position	*c:ArrayIndexor*	The element value's index within the array.	Required

B.1.2.8.2 Child elements

booleanArray/Element inherits the child elements of *c:boolean* (the group *c:DatumQuality*).

B.1.2.9 Collection

The *Collection* complex type shall be the base type for XML schema elements intended to contain multiple data values, i.e., unordered sets of values, ordered vectors of values (with the order of items in the vector being represented by the order of *c:Collection/Item* child elements), or collections of named values, also known as records (with the names being represented by the `name` attribute of the *c:Collection/Item* child element).

B.1.2.9.1 Attributes

Collection contains the following attributes:

Name	Type	Description	Use
defaultStandardUnit	*c:StandardUnit*	This attribute shall contain a unit of measure as defined in IEEE Std 260.1™ [B11].	Optional
defaultNonStandardUnit	*c:NonBlankString*	This attribute shall contain any nonstandard unit, not already defined in IEEE Std 260.1.	Optional
defaultUnitQualifier	*c:NonBlankString*	A textual qualifier that is to be applied to the attribute of either the standardUnit or nonStandardUnit. Examples include RMS and Peak-to-Peak for a unit of volts.	Optional

B.1.2.9.2 Child elements

Collection contains the following child elements, in addition to those inherited from the group *c:DatumQuality* (*Confidence*, *ErrorLimits, Range*, and *Resolution*):

Name	Subclause	Type	Use
Item	B.1.2.10	*c:Value*	0 …∞

B.1.2.10 Collection/Item

Base type: Extension of *c:Value*

Properties: isRef 0, content complex

The *Collection/Item* child element shall contain an individual data value or vector. This child element is recursive; thus a *Collection/Item* may be a collection of data values or vectors.

B.1.2.10.1 Attributes

Collection/Item contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the individual data value or vector.	Optional

B.1.2.10.2 Child elements

Collection/Item inherits the child elements of *c:Value* (*Collection*, *Datum*, and *IndexedArray*).

B.1.2.11 CollectionArray

Base type: Extension of *c:IndexedArrayType*

Properties: base *c:IndexedArrayType*

The *CollectionArray* complex type shall be the "`xsi:type`" of any element of type *c:IndexedArrayType* that contains an array of boolean values.

B.1.2.11.1 Attributes

CollectionArray inherits the attributes of *c:IndexedArrayType* (*dimensions*, *nonStandardUnit*, *standardUnit*, and *unitQualifier*).

B.1.2.11.2 Child elements

CollectionArray contains the following child elements, in addition to those inherited from *c:IndexedArrayType* (the group *c:DatumQuality*):

Name	Subclause	Type	Use
DefaultElementValue	B.1.2.12	*c:Collection*	Optional
Element	B.1.2.13	*c:Collection*	0 …∞

B.1.2.12 CollectionArray/DefaultElementValue

Base type: *c:Collection*

Properties: isRef 0, content complex

The *CollectionArray/DefaultElementValue* child element shall contain the default value of the collection array element.

B.1.2.12.1 Attributes

CollectionArray/DefaultElementValue inherits the attributes of *c:Collection* (*defaultNonStandardUnit*, *defaultStandardUnit*, and *defaultUnitQualifier*).

B.1.2.12.2 Child elements

CollectionArray/DefaultElementValue inherits the child elements of *c:Collection* (the group *c:DatumQuality* and the element *c:Item*).

B.1.2.13 CollectionArray/Element

Base type: Extension of *c:Collection*

Properties: isRef 0, content complex

The *CollectionArray/Element* child element shall contain the value of the collection array element.

B.1.2.13.1 Attributes

CollectionArray/Element contains the following attribute, in addition to those inherited from *c:Collection* (*defaultNonStandardUnit*, *defaultStandardUnit*, and *defaultUnitQualifier*):

Name	Type	Description	Use
position	*c:ArrayIndexor*	The element value's index within the array.	Required

B.1.2.13.2 Child elements

CollectionArray/Element inherits the child elements of *c:Collection* (the group *c:DatumQuality* and the element *c:Item*).

B.1.2.14 complex

Base type: Extension of *c:DatumType*

Properties: base *c:DatumType*

The *complex* complex type shall be the "`xsi:type`" for any element of type *c:DatumType* that will contain complex numbers (i.e., with real and imaginary components).

B.1.2.14.1 Attributes

complex contains the following attributes, in addition to those inherited from *c:DatumType* (*nonStandardUnit*, *standardUnit*, and *unitQualifier*):

Name	Type	Description	Use
imaginary	xs:double	The imaginary part of the complex value.	Required
real	xs:double	The real part of the complex value.	Required

B.1.2.14.2 Child elements

complex inherits the child elements of *c:DatumType* (the group *c:DatumQuality*).

B.1.2.15 complexArray

Base type: Extension of *c:IndexedArrayType*

Properties: base *c:IndexedArrayType*

The *complexArray* complex type shall be the base type of any XML schema element that will contain an array of complex numbers (i.e., with real and imaginary components).

B.1.2.15.1 Attributes

complexArray inherits the attributes of *c:IndexedArrayType* (*dimensions*, *nonStandardUnit*, *standardUnit*, and *unitQualifier*).

B.1.2.15.2 Child elements

complexArray contains the following child elements, in addition to those inherited from *c:IndexedArrayType* (the group *c:DatumQuality*):

Name	Subclause	Type	Use
DefaultElementValue	B.1.2.16	*c:complex*	Optional
Element	B.1.2.17	*c:complex*	0 …∞

B.1.2.16 complexArray/DefaultElementValue

Base type: *c:complex*

Properties: isRef 0, content complex

The *complexArray/DefaultElementValue* child element shall contain the default value of the complex array element.

B.1.2.16.1 Attributes

complexArray/DefaultElementValue inherits the attributes of *c:complex* (*imaginary*, *nonStandardUnit*, *real*, *standardUnit*, and *unitQualifier*).

B.1.2.16.2 Child elements

complexArray/DefaultElementValue inherits the child elements of *c:complex* (the group *c:DatumQuality*).

B.1.2.17 complexArray/Element

Base type: Extension of *c:complex*

Properties: isRef 0, content complex

The *complexArray/Element* child element shall contain the value of the complex array element.

B.1.2.17.1 Attributes

complexArray/Element contains the following attribute, in addition to those inherited from *c:complex* (*imaginary*, *nonStandardUnit*, *real*, *standardUnit*, and *unitQualifier*):

Name	Type	Description	Use
position	*c:ArrayIndexor*	The element value's index within the array.	Required

B.1.2.17.2 Child elements

complexArray/Element inherits the child elements of *c:complex* (the group *c:DatumQuality*).

B.1.2.18 Connector

Base type: Extension of *c:ItemDescription*

Properties: base *c:ItemDescription*

The *Connector* complex type shall be the base type of any XML schema element that will contain connector information.

B.1.2.18.1 Attributes

Connector contains the following attributes, in addition to those inherited from *c:ItemDescription* (*name* and *version*):

Name	Type	Description	Use
ID	*c:NonBlankString*	A user-defined string uniquely identifying the connector. Example: J1.	Required
location	—	A descriptive or common name of where the connector is located. Example: Front Panel.	Required
matingConnectorType	*c:NonBlankString*	A descriptive or common name for the mating connector. Example: The mating connector for a 15-pin d-shell connector (male) is a 15-pin d-shell connector (female).	Optional
type	*c:NonBlankString*	A descriptive or common name for the type of connector. Example: MIL-C-38999.	Required

B.1.2.18.2 Child elements

Connector contains the following child elements, in addition to those inherited from *c:ItemDescription* (*Description*, *Extension*, and *Identification*):

Name	Subclause	Type	Use
Pins	B.1.2.19	—	Required

B.1.2.19 Connector/Pins

Properties: isRef 0, content complex

The *Connector/Pins* child element shall contain descriptive information for each of the pins in the connector.

B.1.2.19.1 Attributes

Connector/Pins contains no attributes.

B.1.2.19.2 Child elements

Connector/Pins contains the following child elements:

Name	Subclause	Type	Use
Pin	B.1.2.20	*c:ConnectorPin*	Required

B.1.2.20 Connector/Pins/Pin

Base type: *c:ConnectorPin*

Properties: isRef 0, content complex

The *Connector/Pins/Pin* child element shall contain descriptive information of a particular pin in the connector.

B.1.2.20.1 Attributes

Connector/Pins/Pin inherits the attributes of *c:ConnectorPin* (*baseIndex*, *count*, *ID*, *incrementedBy*, *name*, and *replacementCharacter*).

B.1.2.20.2 Child elements

Connector/Pins/Pin inherits the child elements of *c:ConnectorPin* (*Definition*).

B.1.2.21 ConnectorLocation

The *ConnectorLocation* complex type shall be the base type of any XML schema element that will contain information associated with the location of an electrical connector.

B.1.2.21.1 Attributes

ConnectorLocation contains the following attributes:

Name	Type	Description	Use
connectorID	*c:NonBlankString*	A user-defined string uniquely identifying the connector.	Required
pinID	*c:NonBlankString*	A user-defined string uniquely identifying the pin within the connector.	Optional

B.1.2.21.2 Child elements

ConnectorLocation contains no child elements.

B.1.2.22 ConnectorPin

Properties: isRef 0, content complex

The *ConnectorPin* complex type shall be the base type of any XML schema element that will contain connector pin information.

B.1.2.22.1 Attributes

ConnectorPin contains the following attributes, in addition to those inherited from the *c:RepeatedItemAttributes* attribute group (*baseIndex*, *count*, *incrementedBy*, and *replacementCharacter*):

Name	Type	Description	Use
ID	*c:NonBlankString*	A user-defined string uniquely identifying the connector pin.	Required
name	*c:NonBlankString*	A descriptive or common name for the connector pin.	Optional

B.1.2.22.2 Child elements

ConnectorPin contains the following child elements:

Name	Subclause	Type	Use
Definition	B.1.2.23	*c:ItemDescription*	Optional

B.1.2.23 ConnectorPin/Definition

Base type: *c:ItemDescription*

Properties: isRef 0, content complex

The *ConnectorPin/Definition* child element shall define a particular pin in the connector.

B.1.2.23.1 Attributes

ConnectorPin/Definition inherits the attributes of *c:ItemDescription* (*name* and *version*).

B.1.2.23.2 Child elements

ConnectorPin/Definition inherits the child elements of *c:ItemDescription* (*Definition*, *Extension*, and *Identification*).

B.1.2.24 dateTime

Base type: Extension of *c:DatumType*

Properties: base *c:DatumType*

The *dateTime* complex type shall be the "`xsi:type`" of any XML schema element of *c:DatumType* that contains a date-time value.

The specific format for *dateTime* data shall follow the ISO 8601 [B37] variable-length character form: [YYYY]-[MM]-[DD]T[hh:mm:ss(.s)][TZD], where **.s** represents optional fractional seconds and **TZD** must be **Z** or **+hh:mm** or **-hh:mm**. By default, all dateTime elements are assumed to represent coordinated universal time (UTC). If a different time zone is represented by the literal value of the data element, the specific UTC offset must be appended to the literal. For example, 2009-07-08T12:00:00+05:00 is 2009-07-08T07:00:00Z.

B.1.2.24.1 Attributes

dateTime contains the following attribute, in addition to those inherited from *c:DatumType* (*nonStandardUnit*, *standardUnit*, and *unitQualifier*). **The attributes inherited from *c:DatumType* are meaningless for data of this type and shall not be used**.

Name	Type	Description	Use
value	xs:dateTime	The *dateTime* value as described.	Required

B.1.2.24.2 Child elements

dateTime inherits the child elements of *c:DatumType* (the group *c:DatumQuality*).

B.1.2.25 dateTimeArray

Base type: Extension of *c:IndexedArrayType*

Properties: base *c:IndexedArrayType*

The *dateTimeArray* complex type shall be the "`xsi:type`" of any element of type *c:IndexedArrayType* that contains an array of date-time values.

B.1.2.25.1 Attributes

dateTimeArray inherits the attributes of *c:IndexedArrayType* (*dimensions*, *nonStandardUnit*, *standardUnit*, and *unitQualifier*). The attributes inherited are meaningless for data of this type and shall not be used.

B.1.2.25.2 Child elements

dateTimeArray contains the following child elements, in addition to those inherited from *c:IndexedArrayType* (the group *c:DatumQuality*):

Name	Subclause	Type	Use
DefaultElementValue	B.1.2.26	*c:dateTime*	Optional
Element	B.1.2.27	*c:dateTime*	0 …∞

B.1.2.26 dateTimeArray/DefaultElementValue

Base type: *c:dateTime*

Properties: isRef 0, content complex

The *dateTimeArray/DefaultElementValue* child element shall contain the default date and time value of the array element.

B.1.2.26.1 Attributes

dateTimeArray/DefaultElementValue inherits the attributes of *c:dateTime* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.1.2.26.2 Child elements

dateTimeArray/DefaultElementValue inherits the child elements of *c:dateTime* (the group *c:DatumQuality*).

B.1.2.27 dateTimeArray/Element

Base type: Extension of *c:dateTime*

Properties: isRef 0, content complex

The *dateTimeArray/Element* child element shall contain the date and time value of the array element.

B.1.2.27.1 Attributes

dateTimeArray/Element contains the following attribute, in addition to those inherited from *c:dateTime* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*):

Name	Type	Description	Use
position	*c:ArrayIndexor*	The element value's index within the array.	Required

B.1.2.27.2 Child elements

dateTimeArray/Element inherits the child elements of *c:dateTime* (the group *c:DatumQuality*).

B.1.2.28 DatumType

Properties: abstract True

The *DatumType* complex type shall be the base type for XML schema elements that contain a numeric, boolean, string, or a date-time data value, each with an optional unit.

B.1.2.28.1 Attributes

DatumType inherits the attributes of the *c:UnitAttributes* attribute group (*nonStandardUnit*, *standardUnit*, and *unitQualifier*).

B.1.2.28.2 Child elements

DatumType inherits the child elements of the group *c:DatumQuality*.

B.1.2.29 Document

The *Document* complex type shall be the base type for any XML schema element that will capture identification information for a document. This information may be in the form of a universal unique identifier (UUID) and the name of the document, a universal resource locator (URL), or the contents of the document. For documents that consist only of short strings, the *Text* element may be used to capture the entire contents of the document.

B.1.2.29.1 Attributes

Document contains the following attributes:

Name	Type	Description	Use
controlNumber	*c:NonBlankString*	A unique identifier for the document.	Optional
name	*c:NonBlankString*	A descriptive or common name for the document	Required
uuid	*c:Uuid*	The universal unique identifier for the document.	Required
version	*c:NonBlankString*	The version identification of the document.	Optional

B.1.2.29.2 Child elements

Document contains the following child elements:

	Name	Subclause	Type	Use
	Extension	B.1.2.30	*c:Extension*	Optional
Choice	Text	B.1.2.31	*c:NonBlankString*	Optional
	URL	B.1.2.32	*c:NonBlankURI*	
NOTE—Choice indicates that only one of these elements may be specified.				

B.1.2.30 Document/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *Document/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.1.2.30.1 Attributes

Document/Extension contains no attributes.

B.1.2.30.2 Child elements

Document/Extension inherits the child element of *c:Extension* (*##other*).

B.1.2.31 Document/Text

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Document/Text* child element shall contain the actual text of the document.

B.1.2.31.1 Attributes

Document/Text contains no attributes.

B.1.2.31.2 Child elements

Document/Text contains no child elements.

B.1.2.32 Document/URL

Base type: *c:NonBlankURI*

Properties: isRef 0, content simple

Facets: minLength 1

The *Document/URL* child element shall contain the URL of the Web site where the document is located.

B.1.2.32.1 Attributes

Document/URL contains no attributes.

B.1.2.32.2 Child elements

Document/URL contains no child elements.

B.1.2.33 DocumentList

The *DocumentList* complex type shall be the base type for any XML schema element that will identify one or more documents.

B.1.2.33.1 Attributes

DocumentList contains no attributes.

B.1.2.33.2 Child elements

DocumentList contains the following child element:

Name	Subclause	Type	Use
Document	B.1.2.34	*c:Document*	1 …∞

B.1.2.34 DocumentList/Document

The *DocumentList/Document* child element shall capture identification information for a document. This information may be in the form of a UUID and the name of the document, a URL, or the contents of the document.

B.1.2.34.1 Attributes

DocumentList/Document inherits the attributes of *c:Document* (*name* and *uuid*).

B.1.2.34.2 Child elements

DocumentList/Document inherits the child elements of *c:Document* (*Extension*, *Text*, and *URL*).

B.1.2.35 DocumentReference

The *DocumentReference* complex type shall be the base type for any XML schema element that will identify an external document.

B.1.2.35.1 Attributes

DocumentReference contains the following attributes:

Name	Type	Description	Use
ID	*c:NonBlankString*	A user-defined string uniquely identifying the document.	Required
uuid	*c:Uuid*	The universal unique identifier for the document.	Required

B.1.2.35.2 Child elements

DocumentReference contains no child elements.

B.1.2.36 double

Base type: Extension of *c:DatumType*

Properties: base *c:DatumType*

The *double* complex type shall be the base type for any XML schema element, including elements of type *c:DatumType*, that contains a numeric value that corresponds to the IEEE 754 double precision 64-bit floating point type.

B.1.2.36.1 Attributes

double contains the following attribute, in addition to those inherited from *c:DatumType* (*nonStandardUnit*, *standardUnit*, and *unitQualifier*):

Name	Type	Description	Use
value	xs:double	The numeric value of the element.	Required

B.1.2.36.2 Child elements

double inherits the child elements of *c:DatumType* (the group *c:DatumQuality*).

B.1.2.37 doubleArray

Base type: Extension of *c:IndexedArrayType*

Properties: base *c:IndexedArrayType*

The *doubleArray* complex type shall be the “`xsi:type`” of any element of type *c:IndexedArrayType* that contains an array of numeric values that correspond to the IEEE 754 double precision 64-bit floating point type.

B.1.2.37.1 Attributes

doubleArray inherits the attributes of *c:IndexedArrayType* (*dimensions*, *nonStandardUnit*, *standardUnit*, and *unitQualifier*).

B.1.2.37.2 Child elements

doubleArray contains the following child elements, in addition to those inherited from *c:IndexedArrayType* (the group *c:DatumQuality*):

Name	Subclause	Type	Use
DefaultElementValue	B.1.2.38	*c:dateTime*	Optional
Element	B.1.2.39	*c:dateTime*	0 …∞

B.1.2.38 doubleArray/DefaultElementValue

Base type: *c:double*

Properties: isRef 0, content complex

The *doubleArray/DefaultElementValue* child element shall contain the default double precision 64-bit floating point value of the array element.

B.1.2.38.1 Attributes

doubleArray/DefaultElementValue inherits the attributes of *c:double* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.1.2.38.2 Child elements

doubleArray/DefaultElementValue inherits the child elements of *c:double* (the group *c:DatumQuality*).

B.1.2.39 doubleArray/Element

Base type: Extension of *c:double*

Properties: isRef 0, content complex

The *doubleArray/Element* child element shall contain the double precision 64-bit floating point value of the array element.

B.1.2.39.1 Attributes

doubleArray/Element contains the following attribute, in addition to those inherited from *c:double* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*):

Name	Type	Description	Use
position	*c:ArrayIndexor*	The element value's index within the array.	Required

B.1.2.39.2 Child elements

doubleArray/Element inherits the child elements of *c:double* (the group *c:DatumQuality*).

B.1.2.40 EnvironmentalElements

The *EnvironmentalElements* complex type shall be the base type for any XML schema element that requires the statement of environmental specifications or values.

B.1.2.40.1 Attributes

EnvironmentalElements contains no attributes.

B.1.2.40.2 Child elements

EnvironmentalElements contains the following child elements:

Name	Subclause	Type	Use
Altitude	B.1.2.41	*c:Limit*	Optional
Humidity	B.1.2.42	*c:Limit*	Optional
Shock	B.1.2.43	*c:Limit*	Optional
Temperature	B.1.2.44	*c:Limit*	Optional
Vibration	B.1.2.45	—	Optional

B.1.2.41 EnvironmentalElements/Altitude

Base type: *c:Limit*

Properties: isRef 0, content complex

The *EnvironmentalElements/Altitude* child element shall contain an altitude value.

B.1.2.41.1 Attributes

EnvironmentalElements/Altitude inherits the attributes of *c:Limit* (*name* and *operator*).

B.1.2.41.2 Child elements

EnvironmentalElements/Altitude inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.1.2.42 EnvironmentalElements/Humidity

Base type: *c:Limit*

Properties: isRef 0, content complex

The *EnvironmentalElements/Humidity* child element shall contain the relative humidity value.

B.1.2.42.1 Attributes

EnvironmentalElements/Humidity inherits the attributes of *c:Limit* (*name* and *operator*).

B.1.2.42.2 Child elements

EnvironmentalElements/Humidity inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.1.2.43 EnvironmentalElements/Shock

Base type: *c:Limit*

Properties: isRef 0, content complex

The *EnvironmentalElements/Shock* child element shall contain the physical shock value.

B.1.2.43.1 Attributes

EnvironmentalElements/Shock inherits the attributes of *c:Limit* (*name* and *operator*).

B.1.2.43.2 Child elements

EnvironmentalElements/Shock inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.1.2.44 EnvironmentalElements/Temperature

Base type: *c:Limit*

Properties: isRef 0, content complex

The *EnvironmentalElements/Temperature* child element shall contain the temperature value.

B.1.2.44.1 Attributes

EnvironmentalElements/Temperature inherits the attributes of *c:Limit* (*name* and *operator*).

B.1.2.44.2 Child elements

EnvironmentalElements/Temperature inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.1.2.45 EnvironmentalElements/Vibration

The *EnvironmentalElements/Vibration* child element shall contain the physical vibration value.

B.1.2.45.1 Attributes

EnvironmentalElements/Vibration contains no attributes.

B.1.2.45.2 Child elements

EnvironmentalElements/Vibration contains the following child elements:

Name	Subclause	Type	Use
Displacement	B.1.2.46	*c:Limit*	Optional
Frequency	B.1.2.47	*c:Limit*	Optional
Velocity	B.1.2.48	*c:Limit*	Optional

B.1.2.46 EnvironmentalElements/Vibration/Displacement

Base type: *c:Limit*

Properties: isRef 0, content complex

The *EnvironmentalElements/Vibration/Displacement* child element shall contain the displacement (the amplitude of a point on the item) value.

B.1.2.46.1 Attributes

EnvironmentalElements/Vibration/Displacement inherits the attributes of *c:Limit* (*name* and *operator*).

B.1.2.46.2 Child elements

EnvironmentalElements/Vibration/Displacement inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.1.2.47 EnvironmentalElements/Vibration/Frequency

Base type: *c:Limit*

Properties: isRef 0, content complex

The *EnvironmentalElements/Vibration/Frequency* child element shall contain the natural resonance frequency value.

B.1.2.47.1 Attributes

EnvironmentalElements/Vibration/Frequency inherits the attributes of *c:Limit* (*name* and *operator*).

B.1.2.47.2 Child elements

EnvironmentalElements/Vibration/Frequency inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.1.2.48 EnvironmentalElements/Vibration/Velocity

Base type: *c:Limit*

Properties: isRef 0, content complex

The *EnvironmentalElements/Vibration/Velocity* child element shall contain the acceleration (rate of change of velocity of a point in an item) value.

B.1.2.48.1 Attributes

EnvironmentalElements/Vibration/Velocity inherits the attributes of *c:Limit* (*name* and *operator*).

B.1.2.48.2 Child elements

EnvironmentalElements/Vibration/Velocity inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.1.2.49 EnvironmentalRequirements

The *EnvironmentalRequirements* complex type shall be the base type for any XML schema element that requires the statement of operational and/or of storage and transport environmental requirements. Typically, this element would be used as part of a XML schema describing hardware.

B.1.2.49.1 Attributes

EnvironmentalRequirements contains no attributes.

B.1.2.49.2 Child elements

EnvironmentalRequirements contains the following child elements:

Name	Subclause	Type	Use
Operation	B.1.2.50	*c:EnvironmentalElements*	Optional
StorageTransport	B.1.2.51	*c:EnvironmentalElements*	Optional

B.1.2.50 EnvironmentalRequirements/Operation

Base type: *c:EnvironmentalElements*

Properties: isRef 0, content complex

The *EnvironmentalRequirements/Operation* child element shall contain operational environmental requirements.

B.1.2.50.1 Attributes

There are no attributes associated with *EnvironmentalRequirements/Operation*.

B.1.2.50.2 Child elements

EnvironmentalRequirements/Operation inherits the child elements of *c:EnvironmentalElements* (*Altitude*, *Humidity*, *Shock*, *Temperature*, and *Vibration*).

B.1.2.51 EnvironmentalRequirements/StorageTransport

Base type: *c:EnvironmentalElements*

Properties: isRef 0, content complex

The *EnvironmentalRequirements/StorageTransport* child element shall contain storage or transport environmental requirements.

B.1.2.51.1 Attributes

EnvironmentalRequirements/StorageTransport contains no attributes.

B.1.2.51.2 Child elements

EnvironmentalRequirements/StorageTransport inherits the child elements of *c:EnvironmentalElements* (*Altitude*, *Humidity*, *Shock*, *Temperature*, and *Vibration*).

B.1.2.52 Extension

Properties: final #all

The *Extension* complex type is provided for the convenience of XML schema developers. The *Extension* type shall be used only as the base type of extension elements in XML schemas. Such elements are provided to permit implementers to extend a XML schema as required to meet the unique needs of their use case. Use follows the W3C standard XML extension mechanism.

B.1.2.52.1 Attributes

Extension contains the XML standard attribute of

```
<xs:any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
```

B.1.2.52.2 Child elements

Extension contains no child elements.

B.1.2.53 HardwareInstance

Base type: Extension of *c:ItemInstance*

Properties: base *c:ItemInstance*

The *HardwareInstance* complex type shall be the base type for any XML schema element that is intended to capture data describing or identifying a specific instance of physical hardware.

B.1.2.53.1 Attributes

HardwareInstance contains no attributes.

B.1.2.53.2 Child elements

HardwareInstance contains the following child elements, in addition to those inherited from *c:ItemInstance* (*Definition*, *DescriptionDocumentReference*, and *SerialNumber*):

Name	Subclause	Type	Use
Calibration	B.1.2.54	—	Optional
Components	B.1.2.55	—	Optional
ManufactureDate	B.1.2.57	xs:dateTime	Optional
ParentComponent	B.1.2.58	*c:HardwareInstance*	Optional
PowerOn	B.1.2.59	—	Optional

B.1.2.54 HardwareInstance/Calibration

Properties: isRef 0, content complex

The *HardwareInstance/Calibration* child element shall contain the date and time the hardware item was last calibrated.

B.1.2.54.1 Attributes

HardwareInstance/Calibration contains the following attribute:

Name	Type	Description	Use
time	xs:dateTime	The date and time value.	Required

B.1.2.54.2 Child elements

HardwareInstance/Calibration contains no child elements.

B.1.2.55 HardwareInstance/Components

Properties: isRef 0, content complex

The *HardwareInstance/Components* child element shall identify the next-lower assembly belonging to the parent hardware item.

B.1.2.55.1 Attributes

HardwareInstance/Components contains no attributes.

B.1.2.55.2 Child elements

HardwareInstance/Components contains the following child element:

Name	Subclause	Type	Use
Component	B.1.2.56	*c:ItemInstanceReference*	1 …∞

B.1.2.56 HardwareInstance/Components/Component

Base type: *c:ItemInstanceReference*

Properties: isRef 0, content complex

The *HardwareInstance/Components/Component* child element shall identify each next-lower assembly belonging to the parent hardware item.

B.1.2.56.1 Attributes

HardwareInstance/Components/Component contains no attributes.

B.1.2.56.2 Child elements

HardwareInstance/Components/Component inherits the child elements of *c:ItemInstanceReference* (*Definition* and *InstanceDocumentReference*).

B.1.2.57 HardwareInstance/ManufactureDate

Base type: xs:dateTime

Properties: isRef 0, content simple

The *HardwareInstance/ManufactureDate* child element shall identify the date the hardware item was manufactured.

B.1.2.57.1 Attributes

HardwareInstance/ManufactureDate contains no attributes.

B.1.2.57.2 Child elements

HardwareInstance/ManufactureDate contains no child elements.

B.1.2.58 HardwareInstance/ParentComponent

Base type: *c:HardwareInstance*

Properties: isRef 0, content complex

The *HardwareInstance/ParentComponent* child element shall identify the next-higher assembly to which the parent hardware item belongs.

B.1.2.58.1 Attributes

HardwareInstance/ParentComponent contains no attributes.

B.1.2.58.2 Child elements

HardwareInstance/ParentComponent inherits the child elements of *c:HardwareInstance* (*Calibration*, *Components*, *Definition*, *DescriptionDocumentReference*, *ParentComponent*, *PowerOn*, and *SerialNumber*).

B.1.2.59 HardwareInstance/PowerOn

Properties: isRef 0, content complex

The *HardwareInstance/PowerOn* child element shall indicate the number of power-on cycles and the total power-on time experienced by the hardware item at the time of creation of the XML instance document.

B.1.2.59.1 Attributes

HardwareInstance/PowerOn contains the following attributes:

Name	Type	Description	Use
count	xs:int	The number of power-on cycles.	Required
time	xs:duration	The total power-on time.	Required

B.1.2.59.2 Child elements

HardwareInstance/PowerOn contains no child elements.

B.1.2.60 hexadecimal

Base type: Extension of *c:DatumType*

Properties: base *c:DatumType*

The *hexadecimal* complex type shall be the "`xsi:type`" of any element of type *c:DatumType* that contains a hex-encoded binary value.

B.1.2.60.1 Attributes

hexadecimal contains the following attribute, in addition to those inherited from *c:DatumType* (*nonStandardUnit*, *standardUnit*, and *unitQualifier*):

Name	Type	Description	Use
value	*c:HexValue*	The numeric value of the element. Hexadecimal digits shall be formatted as **0x** followed by a finite-length sequence of characters 0–9 and a–f. Letters may be either lowercase or uppercase.	Required

B.1.2.60.2 Child elements

hexadecimal inherits the child elements of *c:DatumType* (*Confidence*, *ErrorLimits*, *Range*, and *Resolution*).

B.1.2.61 hexadecimalArray

Base type: Extension of *c:IndexedArrayType*

Properties: base *c:IndexedArrayType*

The *hexadecimalArray* complex type shall be the “`xsi:type`” of any element of type *c:IndexedArrayType* that contains an array of hex-encoded binary values.

B.1.2.61.1 Attributes

hexadecimalArray inherits the attributes of *c:IndexedArrayType* (*dimensions*, *nonStandardUnit*, *standardUnit*, and *unitQualifier*).

B.1.2.61.2 Child elements

hexadecimalArray contains the following child elements, in addition to those inherited from *c:IndexedArrayType* (*Confidence*, *ErrorLimits*, *Range*, and *Resolution*):

Name	Subclause	Type	Use
DefaultElementValue	B.1.2.62	*c:hexadecimal*	Optional
Element	B.1.2.63	*c:hexadecimal*	0 … ∞

B.1.2.62 hexadecimalArray/DefaultElementValue

Base type: *c:hexadecimal*

Properties: isRef 0, content complex

The *hexadecimalArray/DefaultElementValue* child element shall contain the default hexadecimal value of the array element.

B.1.2.62.1 Attributes

hexadecimalArray/DefaultElementValue inherits the attributes of *c:hexadecimal* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.1.2.62.2 Child elements

hexadecimalArray/DefaultElementValue inherits the attributes of *c:hexadecimal* (*Confidence*, *ErrorLimits*, *Range*, and *Resolution*).

B.1.2.63 hexadecimalArray/Element

Base type: Extension of *c:hexadecimal*

Properties: isRef 0, content complex

The *hexadecimalArray/Element* child element shall contain the hexadecimal value of the array element.

B.1.2.63.1 Attributes

hexadecimalArray/Element contains the following attribute, in addition to those inherited from *c:hexadecimal* (*Confidence*, *ErrorLimits*, *Range*, and *Resolution*):

Name	Type	Description	Use
position	*c:ArrayIndexor*	The element value's index within the array.	Required

B.1.2.63.2 Child elements

hexadecimalArray/Element inherits the attributes of *c:hexadecimal* (*Confidence*, *ErrorLimits*, *Range*, and *Resolution*).

B.1.2.64 IdentificationNumber

The *IdentificationNumber* complex type shall be the base type of any XML schema element that will contain entity identification (such as hardware part number).

B.1.2.64.1 Attributes

IdentificationNumber contains the following attributes:

Name	Type	Description	Use
number	*c:NonBlankString*	The part number of the entity.	Required
type	—	An indication of whether the *c:IdentificationNumber* is a part number, model number, or other.	Required

B.1.2.64.2 Child elements

IdentificationNumber contains no child elements.

B.1.2.65 IndexedArrayType

Properties: abstract true

The *IndexedArrayType* complex type shall be the base type for any XML schema element that will contain an array of numeric, boolean, string, or date-time data values, or an array of collections, with an optional unit. The array may be sparse.

B.1.2.65.1 Attributes

IndexedArrayType contains the following attribute, in addition to those inherited from the *c:UnitAttributes* Attribute Group (*nonStandardUnit*, *standardUnit*, and *unitQualifier*):

Name	Type	Description	Use
dimensions	*c:ArrayIndexor*	A string designating an *n*-dimensional array index or array dimension, with the format [*a*,*b*,*c*,…,*n*], where *a*,*b*,*c*,…*n* are numeric indices. Example: [3,4] specifies a 3-by-4 two-dimensional array.	Required

B.1.2.65.2 Child elements

IndexedArrayType inherits the child elements of group *c:DatumQuality* (*Confidence*, *ErrorLimits*, *Range*, and *Resolution*).

B.1.2.66 integer

Base type: Extension of *c:DatumType*

Properties: base *c:DatumType*

The *integer* complex type shall be the "`xsi:type`" for elements of type *c:DatumType* that contain a 32-bit signed integer value.

B.1.2.66.1 Attributes

integer contains the following attribute, in addition to those inherited from *c:DatumType* (*nonStandardUnit*, *standardUnit*, and *unitQualifier*):

Name	Type	Description	Use
value	xs:int	The numeric value of the element, between +2 147 483 647 and -2 147 483 648 (inclusive).	Required

B.1.2.66.2 Child elements

integer inherits the child elements of *c:DatumType* (the group *c:DatumQuality*).

B.1.2.67 integerArray

Base type: Extension of *c:IndexedArrayType*

Properties: base *c:IndexedArrayType*

The *integerArray* complex type shall be the "`xsi:type`" of any element(s) of type *c:IndexedArrayType* that contain an array of 32-bit signed integer values.

B.1.2.67.1 Attributes

integerArray inherits the attributes of *c:IndexedArrayType* (*dimensions*, *nonStandardUnit*, *standardUnit*, and *unitQualifier*).

B.1.2.67.2 Child elements

integerArray contains the following child elements, in addition to those inherited from *c:IndexedArrayType* (the group *c:DatumQuality*):

Name	Subclause	Type	Use
DefaultElementValue	B.1.2.68	*c:integer*	Optional
Element	B.1.2.69	*c:integer*	0 … ∞

B.1.2.68 integerArray/DefaultElementValue

Base type: *c:integer*

Properties: isRef 0, content complex

The *integerArray/DefaultElementValue* child element shall contain the default integer value of the array element.

B.1.2.68.1 Attributes

integerArray/DefaultElementValue inherits the attributes of *c:integer* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.1.2.68.2 Child elements

integerArray/DefaultElementValue inherits the child elements of *c:integer* (the group *c:DatumQuality*).

B.1.2.69 integerArray/Element

Base type: Extension of *c:integer*

Properties: isRef 0, content complex

The *integerArray/Element* child element shall contain the integer value of the array element.

B.1.2.69.1 Attributes

integerArray/Element contains the following attribute, in addition to those inherited from *c:integer* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*):

Name	Type	Description	Use
position	*c:ArrayIndexor*	The element value's index within the array.	Required

B.1.2.69.2 Child elements

integerArray/Element inherits the child elements of *c:integer* (the group *c:DatumQuality*).

B.1.2.70 Interface

The *Interface* complex type shall be the base type for any XML schema element that describes electrical interfaces to a device.

B.1.2.70.1 Attributes

Interface contains no attributes.

B.1.2.70.2 Child elements

Interface contains the following child element:

Name	Subclause	Type	Use
Ports	B.1.2.71	—	1 … ∞

B.1.2.71 Interface/Ports

Properties: isRef 0, content complex

The *Interface/Ports* child element shall serve as a collector element of an unbounded set of *c:Port* elements.

B.1.2.71.1 Attributes

Interface/Ports contains no attributes.

B.1.2.71.2 Child elements

Interface/Ports contains the following child element:

Name	Subclause	Type	Use
Port	B.1.2.72	*c:Port*	1 … ∞

B.1.2.72 Interface/Ports/Port

Base type: *c:Port*

Properties: isRef 0, content complex

The *Interface/Ports/Port* child element shall contain the name of the depicted port.

B.1.2.72.1 Attributes

Interface/Ports/Port inherits the attributes of *c:Port* (*direction*, *name*, and *type*).

B.1.2.72.2 Child elements

Interface/Ports/Port inherits the child element of *c:Port* (*Extension*).

B.1.2.73 ItemDescription

The *ItemDescription* complex type shall be the base type for any XML schema element that is intended to contain descriptive and identification information for any entity.

B.1.2.73.1 Attributes

ItemDescription contains the following attributes:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the described item.	Optional
version	*c:NonBlankString*	A string designating the version of the described item.	Optional

B.1.2.73.2 Child elements

ItemDescription contains the following child elements:

Name	Subclause	Type	Use
Description	B.1.2.74	*c:NonBlankString*	Optional
Extension	B.1.2.75	*c:Extension*	Optional
Identification	B.1.2.76	—	Required

B.1.2.74 ItemDescription/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *ItemDescription/Description* child element shall contain a free-form textual description of the item described.

B.1.2.74.1 Attributes

ItemDescription/Description contains no attributes.

B.1.2.74.2 Child elements

ItemDescription/Description contains no child elements.

B.1.2.75 ItemDescription/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *ItemDescription/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.1.2.75.1 Attributes

ItemDescription/Extension contains no attributes.

B.1.2.75.2 Child elements

ItemDescription/Extension inherits the child element of *c:Extension* (*##other*).

B.1.2.76 ItemDescription/Identification

Properties: isRef 0, content complex

The *ItemDescription/Identification* child element shall identify a class of the described item.

B.1.2.76.1 Attributes

ItemDescription/Identification contains the following attribute:

Name	Type	Description	Use
designator	*c:NonBlankString*	An alphanumeric string that identifies an item within a larger assembly. For example, a reference designator such as **A25** to indicate a circuit card number.	Optional

B.1.2.76.2 Child elements

ItemDescription/Identification contains the following child elements:

Name	Subclause	Type	Use
Extension	B.1.2.77	*c:Extension*	Optional
IdentificationNumbers	B.1.2.78	—	Optional
Manufacturers	B.1.2.79	—	Optional
ModelName	B.1.2.83	*c:NonBlankString*	Required
Version	B.1.2.84	*c:NonBlankString*	Optional

B.1.2.77 ItemDescription/Identification/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *ItemDescription/Identification/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.1.2.77.1 Attributes

ItemDescription/Identification/Extension contains no attributes.

B.1.2.77.2 Child elements

ItemDescription/Identification/Extension inherits the child element of *c:Extension* (*##other*).

B.1.2.78 ItemDescription/Identification/IdentificationNumbers

Properties: isRef 0, content complex

The *ItemDescription/Identification/IdentificationNumbers* child element shall be a collector for an unbounded set of *IdentificationNumber* or *ManufacturerIdentificationNumber* child elements. This element identifies multiple part or model numbers for the described item (such as a user and/or manufacturer part number).

B.1.2.78.1 Attributes

ItemDescription/Identification/IdentificationNumbers contains no attributes.

B.1.2.78.2 Child elements

ItemDescription/Identification/IdentificationNumbers contains one of the following child elements:

	Name	Subclause	Type	Use
Choice	IdentificationNumber	B.1.2.79	*c:UserDefinedIdentificationNumber*	1.. ∞
	ManufacturerIdentificationNumber	B.1.2.80	*c:ManufacturerIdentificationNumber*	
NOTE—Choice indicates that only one of these elements may be specified.				

B.1.2.79 ItemDescription/Identification/IdentificationNumbers/IdentificationNumber

Base type: *c:UserDefinedIdentificationNumber*

Properties: isRef 0, content complex

The *ItemDescription/Identification/IdentificationNumbers/IdentificationNumber* child element shall provide for multiple end-user-assigned part or model numbers for the described item.

B.1.2.79.1 Attributes

ItemDescription/Identification/IdentificationNumbers/IdentificationNumber inherits the attributes from *c:UserDefinedIdentificationNumber* (*number, qualifier*, and *type*).

B.1.2.79.2 Child elements

ItemDescription/Identification/IdentificationNumbers/IdentificationNumber contains no child elements.

B.1.2.80 ItemDescription/Identification/IdentificationNumbers/ManufacturerIdentificationNumber

Base type: *c:ManufacturerIdentificationNumber*

Properties: isRef 0, content complex

The *ItemDescription/Identification/IdentificationNumbers/ManufacturerIdentificationNumber* child element shall provide for multiple manufacturers' assigned part or model numbers, which are not the end-users' assigned part number, for the described item.

B.1.2.80.1 Attributes

ItemDescription/Identification/IdentificationNumbers/ManufacturerIdentificationNumber inherits the attributes from *c:ManufacturerIdentificationNumber* (*manufacturerName*, *number* and *type*).

B.1.2.80.2 Child elements

ItemDescription/Identification/IdentificationNumbers/ManufacturerIdentificationNumber contains no child elements.

B.1.2.81 ItemDescription/Identification/Manufacturers

Properties: isRef 0, content complex

The *ItemDescription/Identification/Manufacturers* child element shall identify the manufacturers of the item.

B.1.2.81.1 Attributes

ItemDescription/Identification/Manufacturers contains no attributes.

B.1.2.81.2 Child elements

ItemDescription/Identification/Manufacturers contains the following child element:

Name	Subclause	Type	Use
Manufacturer	B.1.2.82	*c:ManufacturerData*	1 … ∞

B.1.2.82 ItemDescription/Identification/Manufacturers/Manufacturer

Base type: *c:ManufacturerData*

Properties: isRef 0, content complex

The *ItemDescription/Identification/Manufacturers/Manufacturer* child element shall identify the manufacturer of the item.

B.1.2.82.1 Attributes

ItemDescription/Identification/Manufacturers/Manufacturer inherits the attributes of *c:ManufacturerData* (*cageCode* and *name*).

B.1.2.82.2 Child elements

ItemDescription/Identification/Manufacturers/Manufacturer inherits the child elements of *c:ManufacturerData* (*Contacts*, *FaxNumber*, *MailingAddress*, and *URL*).

B.1.2.83 ItemDescription/Identification/ModelName

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *ItemDescription/Identification/ModelName* child element shall contain the model name of the item.

B.1.2.83.1 Attributes

ItemDescription/Identification/ModelName contains no attributes.

B.1.2.83.2 Child elements

ItemDescription/Identification/ModelName contains no child elements.

B.1.2.84 ItemDescription/Identification/Version

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *ItemDescription/Identification/Version* child element shall contain a textual description of the version of the item.

B.1.2.84.1 Attributes

ItemDescription/Identification/Version contains no attributes.

B.1.2.84.2 Child elements

ItemDescription/Identification/Version contains no child elements.

B.1.2.85 ItemDescriptionReference

The *ItemDescriptionReference* complex type shall be the base type for any XML schema element that requires element(s) referencing *c:ItemDescription* element(s).

B.1.2.85.1 Attributes

ItemDescriptionReference contains no attributes.

B.1.2.85.2 Child elements

ItemDescriptionReference contains one of the following child elements:

	Name	Subclause	Type	Use
Choice	Definition	B.1.2.86	*c:ItemDescription*	Required
	DescriptionDocumentReference	B.1.2.87	*c:DocumentReference*	
NOTE—Choice indicates that only one of these elements may be specified.				

B.1.2.86 ItemDescriptionReference/Definition

Base type: *c:ItemDescription*

Properties: isRef 0, content complex

The *ItemDescriptionReference/Definition* child element shall uniquely identify a specific description of an item.

B.1.2.86.1 Attributes

ItemDescriptionReference/Definition inherits the attributes of *c:ItemDescription* (*name* and *version*).

B.1.2.86.2 Child elements

ItemDescriptionReference/Definition inherits the child elements of *c:ItemDescription* (*Description*, *Extension*, and *Identification*).

B.1.2.87 ItemDescriptionReference/DescriptionDocumentReference

Base type: *c:DocumentReference*

Properties: isRef 0, content complex

The *ItemDescriptionReference/DescriptionDocumentReference* child element shall identify the UUID corresponding to the specific instance document.

B.1.2.87.1 Attributes

ItemDescriptionReference/DescriptionDocumentReference inherits the attributes of *c:DocumentReference* (*ID* and *uuid*).

B.1.2.87.2 Child elements

ItemDescriptionReference/DescriptionDocumentReference contains no child elements.

B.1.2.88 ItemInstance

Base type: Extension of *c:ItemDescriptionReference*

Properties: base *c:ItemDescriptionReference*

The *ItemInstance* complex type shall be the base type for any XML schema element that is intended to capture identification information specifying a single instance of an item.

B.1.2.88.1 Attributes

ItemInstance contains no attributes.

B.1.2.88.2 Child elements

ItemInstance contains the following child elements, in addition to those inherited from *c:ItemDescriptionReference* (*Definition* and *DescriptionDocumentReference*):

Name	Subclause	Type	Use
SerialNumber	B.1.2.89	*c:NonBlankString*	Required

B.1.2.89 ItemInstance/SerialNumber

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *ItemInstance/SerialNumber* child element shall uniquely identify a specific instance of an item.

B.1.2.89.1 Attributes

ItemInstance/SerialNumber contains no attributes.

B.1.2.89.2 Child elements

ItemInstance/SerialNumber contains no child elements.

B.1.2.90 ItemInstanceReference

The *ItemInstanceReference* complex type shall be the base type for any XML schema element that requires an element to reference a *c:ItemInstance* that has no serial number.

B.1.2.90.1 Attributes

ItemInstanceReference contains no attributes.

B.1.2.90.2 Child elements

ItemInstanceReference contains one of the following child elements:

	Name	Subclause	Type	Use
Choice	Definition	B.1.2.91	*c:ItemDescription*	Required
	InstanceDocumentReference	B.1.2.92	*c:DocumentReference*	
NOTE—Choice indicates that only one of these elements may be specified.				

B.1.2.91 ItemInstanceReference/Definition

Base type: *c:ItemInstance*

Properties: isRef 0, content complex

The *ItemInstanceReference/Definition* child element shall uniquely identify a specific instance of an item.

B.1.2.91.1 Attributes

ItemInstanceReference/Definition contains no attributes.

B.1.2.91.2 Child elements

ItemInstanceReference/Definition inherits the child elements of *c:ItemInstance* (*Definition*, *DescriptionDocumentReference*, and *SerialNumber*).

B.1.2.92 ItemInstanceReference/InstanceDocumentReference

Base type: *c:DocumentReference*

Properties: isRef 0, content complex

The *ItemInstanceReference/InstanceDocumentReference* child element shall identify the UUID corresponding to the specific instance document.

B.1.2.92.1 Attributes

ItemInstanceReference/InstanceDocumentReference inherits the attributes of *c:DocumentReference* (*ID* and *uuid*).

B.1.2.92.2 Child elements

ItemInstanceReference/InstanceDocumentReference contains no child elements.

B.1.2.93 Limit

The *Limit* complex type shall be the base type for any element that contains limit data where such data are a comparison to a single value. The datatypes must be consistent for the purposes of comparison, e.g., should a limit be represented as a string, then strings shall be used through the entire limit description so that strings can be compared to strings.

B.1.2.93.1 Attributes

Limit contains the following attributes:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the limit expressed in the element.	Optional
operator	*c:LogicalOperator*	The comparison with the two boundary limits may be for a value **between** the limits or **outside** the limits. The LogicalOperator AND explicitly indicates a **between** comparison; OR explicitly indicates an **outside** comparison. Example: GT 3 AND LT 7 (between) vs. GT 10 OR LT 3 or GT 5 OR GT 10 (outside). While the logical operator may be inferred from the combination of limit values and comparison types, the *c:LogicalOperator* attribute permits better definition and less possibility for misinterpretation.	Optional

B.1.2.93.2 Child elements

Limit contains the following child elements:

	Name	Subclause	Type	Use
	Description	B.1.2.94	*c:NonBlankString*	Optional
	Extension	B.1.2.96	*c:Extension*	Optional
Choice	Expected	B.1.2.95	*c:LimitExpected*	Required
	LimitPair	B.1.2.97	*c:LimitPair*	
	Mask	B.1.2.98	*c:LimitMask*	
	SingleLimit	B.1.2.99	*c:SingleLimit*	
NOTE—Choice indicates that only one of these elements may be specified.				

B.1.2.94 Limit/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Limit/Description* child element shall contain a textual description of the limit being described.

B.1.2.94.1 Attributes

Limit/Description contains no attributes.

B.1.2.94.2 Child elements

Limit/Description contains no child elements.

B.1.2.95 Limit/Expected

Base type: *c:LimitExpected*

Properties: isRef 0, content complex

The *Limit/Expected* child element shall identify the desired or expected value that will be used for the purposes of limit comparison.

B.1.2.95.1 Attributes

Limit/Expected inherits the attribute of *c:LimitExpected* (*comparator*).

B.1.2.95.2 Child elements

Limit/Expected inherits the child elements of *c:LimitExpected* (*Collection*, *Datum*, and *IndexedArray*).

B.1.2.96 Limit/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *Limit/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.1.2.96.1 Attributes

Limit/Extension contains no attributes.

B.1.2.96.2 Child elements

Limit/Extension inherits the child element of *c:Extension* (*##other*).

B.1.2.97 Limit/LimitPair

Base type: *c:LimitPair*

Properties: isRef 0, content complex

The *Limit/LimitPair* child element shall contain the pair of limit values for the use cases where the limit is bounded by a pair of values.

B.1.2.97.1 Attributes

Limit/LimitPair inherits the attributes of *c:LimitPair* (*name* and *operator*).

B.1.2.97.2 Child elements

Limit/LimitPair inherits the child elements of *c:LimitPair* (*Limit* and *Nominal*).

B.1.2.98 Limit/Mask

Base type: *c:LimitMask*

Properties: isRef 0, content complex

The *Limit/Mask* child element shall contain the numeric mask value.

B.1.2.98.1 Attributes

Limit/Mask contains no attributes.

B.1.2.98.2 Child elements

Limit/Mask inherits the child elements of *c:LimitMask* (*Expected* and *MaskValue*).

B.1.2.99 Limit/SingleLimit

Base type: *c:SingleLimit*

Properties: isRef 0, content complex

The *Limit/SingleLimit* child element shall contain the value being used for the purposes of limit comparison.

B.1.2.99.1 Attributes

Limit/SingleLimit inherits the attribute of *c:SingleLimit* (*comparator*).

B.1.2.99.2 Child elements

Limit/SingleLimit inherits the child elements of *c:SingleLimit* (*Collection*, *Datum*, and *IndexedArray*).

B.1.2.100 LimitExpected

The *LimitExpected* complex type shall be the base type for any XML schema element that requires identification of the desired or expected value that will be used for the purposes of limit comparison.

B.1.2.100.1 Attributes

LimitExpected contains the following attribute:

Name	Type	Description	Use
comparator	*c:EqualityComparisonOperator*	The comparison logic to be applied to the limit. Examples: EQ or NE.	Required

B.1.2.100.2 Child elements

LimitExpected inherits the child elements of *c:Value* (*Collection*, *Datum*, and *IndexedArray*).

B.1.2.101 LimitMask

The *LimitMask* complex type shall be the base type for any XML schema element that requires identification of a numeric mask value.

B.1.2.101.1 Attributes

LimitMask contains no attributes

B.1.2.101.2 Child elements

LimitMask contains the following child elements:

Name	Subclause	Type	Use
Expected	B.1.2.102	*c:Value*	Required
MaskValue	B.1.2.103	*c:Value*	1 … ∞

B.1.2.102 LimitMask/Expected

Base type: *c:Value*

Properties: isRef 0, content complex

The *LimitMask/Expected* child element shall contain the expected pattern.

B.1.2.102.1 Attributes

LimitMask/Expected contains no attributes.

B.1.2.102.2 Child elements

LimitMask/Expected inherits the child elements of *c:Value* (*Collection*, *Datum*, and *IndexedArray*).

B.1.2.103 LimitMask/MaskValue

Base type: Extension of *c:Value*

Properties: isRef 0, content complex

The *LimitMask/MaskValue* child element shall contain the mask pattern.

B.1.2.103.1 Attributes

LimitMask/MaskValue contains the following attributes:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the limit.	Optional
operation	*c:MaskOperator*	The logical operation that is to be applied (AND, OR, or XOR) to the mask and the value.	Required

B.1.2.103.2 Child elements

LimitMask/MaskValue inherits the child elements of *c:Value* (*Collection*, *Datum*, and *IndexedArray*).

B.1.2.104 LimitPair

The *LimitPair* complex type shall be the base type for any element that captures paired boundary condition data used in a comparison or evaluation.

B.1.2.104.1 Attributes

LimitPair contains the following attributes:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the limit pair expressed in the element.	Optional
operator	*c:LogicalOperator*	The comparison with the two boundary limits may be for a value **between** the limits or **outside** the limits. The LogicalOperator AND explicitly indicates a **between** comparison; OR explicitly indicates an **outside** comparison. Example: GT 3 AND LT 7 (between) vs. GT 10 OR LT 3 (outside). While the logical operator may be inferred from the combination of limit values and comparison types, the LogicalOperator attribute permits better definition and less possibility for misinterpretation.	Required

B.1.2.104.2 Child elements

LimitPair contains the following child elements:

Name	Subclause	Type	Use
Limit	B.1.2.105	*c:SingleLimit*	2 Required
Nominal	B.1.2.106	*c:Value*	Optional

B.1.2.105 LimitPair/Limit

Base type: *c:SingleLimit*

Properties: isRef 0, content complex

The *LimitPair/Limit* child element shall contain two (and only two) limit values.

B.1.2.105.1 Attributes

LimitPair/Limit inherits the attribute of *c:SingleLimit* (*comparator*).

B.1.2.105.2 Child elements

LimitPair/Limit inherits the child elements of *c:SingleLimit* (*Collection*, *Datum*, and *IndexedArray*).

B.1.2.106 LimitPair/Nominal

Base type: *c:Value*

Properties: isRef 0, content complex

The *LimitPair/Nominal* child element shall contain the expected or preferred value to be captured.

B.1.2.106.1 Attributes

LimitPair/Nominal contains no attributes.

B.1.2.106.2 Child elements

LimitPair/Nominal inherits the child elements of *c:Value* (*Collection*, *Datum*, and *IndexedArray*).

B.1.2.107 long

Base type: Extension of *c:DatumType*

Properties: base *c:DatumType*

The *long* complex type shall be the "`xsi:type`" for elements of type *c:DatumType* that contain a 64-bit signed integer value.

B.1.2.107.1 Attributes

long contains the following attribute, in addition to those inherited from c:DatumType (the group *DatumQuality*).

Name	Type	Description	Use
value	xs:long	The numeric value, between +9 223 372 036 854 755 807 and -9 223 372 036 854 755 808 (inclusive).	Required

B.1.2.107.2 Child elements

long inherits the child elements of *c:DatumType* (*Confidence*, *ErrorLimits, Range,* and *Resolution*).

B.1.2.108 longArray

Base type: Extension of *c:IndexedArrayType*

Properties: base *c:IndexedArrayType*

The *longArray* complex type shall be the "`xsi:type`" for elements of type *c:IndexedArrayType* that contain an array of 32-bit signed integer value.

B.1.2.108.1 Attributes

longArray inherits the attributes from *c:IndexedArrayType* (*dimensions*, *standardUnit*, *nonStandardUnit*, and *unitQualifier*).

B.1.2.108.2 Child elements

longArray contains the following child elements, in addition to those inherited from *c:IndexedArrayType* (the group *c:DatumQuality*).

Name	Subclause	Type	Use
DefaultElementValue	B.1.2.109	*c:long*	Optional
Element	B.1.2.110	*c:long*	0 … ∞

B.1.2.109 longArray/DefaultElementValue

Base type: *c:long*

Properties: isRef 0, content complex

The *longArray/DefaultElementValue* child element shall contain the default integer value of the array element.

B.1.2.109.1 Attributes

longArray/DefaultElementValue inherits the attributes of *c:long* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.1.2.109.2 Child elements

longArray/DefaultElementValue inherits the child elements of *c:long* (the group *c:DatumQuality*).

B.1.2.110 longArray/Element

Base type: Extension of *c:long*

Properties: isRef 0, content complex

The *longArray/Element* child element shall contain the integer value of the array element.

B.1.2.110.1 Attributes

longArray/Element contains the following attribute, in addition to those inherited from *c:long* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*):

Name	Type	Description	Use
position	*c:ArrayIndexor*	The element value's index within the array.	Required

B.1.2.110.2 Child elements

longArray/Element inherits the child elements of *c:long* (the group *c:DatumQuality*).

B.1.2.111 MailingAddress

The *MailingAddress* complex type shall be the base type for any XML schema element that will contain a street or mailing address. An example is the mailing address information for a manufacturer.

B.1.2.111.1 Attributes

MailingAddress contains no attributes.

B.1.2.111.2 Child elements

MailingAddress contains the following child elements:

Name	Subclause	Type	Use
Address1	B.1.2.112	*c:NonBlankString*	Required
Address2	B.1.2.113	*c:NonBlankString*	Optional
City	B.1.2.114	*c:NonBlankString*	Required
Country	B.1.2.115	*c:NonBlankString*	Required
PostalCode	B.1.2.116	*c:NonBlankString*	Required
State	B.1.2.117	*c:NonBlankString*	Optional

B.1.2.112 MailingAddress/Address1

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *MailingAddress/Address1* child element shall contain a textual description of the physical street address.

B.1.2.112.1 Attributes

MailingAddress/Address1 contains no attributes.

B.1.2.112.2 Child elements

MailingAddress/Address1 contains no child elements.

B.1.2.113 MailingAddress/Address2

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *MailingAddress/Address2* child element shall contain a textual description of additional street address information (e.g., suite number, mail stop) that shall be associated with *c:MailingAddress/Address1*.

B.1.2.113.1 Attributes

MailingAddress/Address2 contains no attributes.

B.1.2.113.2 Child elements

MailingAddress/Address2 contains no child elements.

B.1.2.114 MailingAddress/City

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *MailingAddress/City* child element shall contain a textual description of the city that shall be associated with *c:MailingAddress/Address1*.

B.1.2.114.1 Attributes

MailingAddress/City contains no attributes.

B.1.2.114.2 Child elements

MailingAddress/City contains no child elements.

B.1.2.115 MailingAddress/Country

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *MailingAddress/Country* child element shall contain a textual description of the territory occupied by a nation.

B.1.2.115.1 Attributes

MailingAddress/Country contains no attributes.

B.1.2.115.2 Child elements

MailingAddress/Country contains no child elements.

B.1.2.116 MailingAddress/PostalCode

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *MailingAddress/PostalCode* child element shall contain a series of letters and/or digits typically appended to the postal address for the purposes of sorting mail (Example: U.S. Postal Service ZIP code).

B.1.2.116.1 Attributes

MailingAddress/PostalCode contains no attributes.

B.1.2.116.2 Child elements

MailingAddress/PostalCode contains no child elements.

B.1.2.117 MailingAddress/State

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *MailingAddress/State* child element shall contain the U.S. state (Examples: Florida and Hawaii) typically appended to an U.S. postal address.

B.1.2.117.1 Attributes

MailingAddress/State contains no attributes.

B.1.2.117.2 Child elements

MailingAddress/State contains no child elements.

B.1.2.118 ManufacturerData

The *ManufacturerData* complex type shall be the base type for any XML schema element that is intended to contain information identifying the manufacturer of an item.

B.1.2.118.1 Attributes

ManufacturerData contains the following attributes:

Name	Type	Description	Use
cageCode	*c:NonBlankString*	The commercial and government entity (CAGE) code for the company indicated by the `name` attribute.	Optional
name	*c:NonBlankString*	A descriptive or common name for the manufacturer.	Required

B.1.2.118.2 Child elements

ManufacturerData contains the following child elements:

Name	Subclause	Type	Use
Contacts	B.1.2.119	—	Optional
FaxNumber	B.1.2.121	*c:NonBlankString*	Optional
MailingAddress	B.1.2.122	*c:NonBlankString*	Optional
URL	B.1.2.123	*c:NonBlankURI*	Optional

B.1.2.119 ManufacturerData/Contacts

Properties: isRef 0, content complex

The *ManufacturerData/Contacts* child element shall be a collector for an unbounded set of child *ManufacturerData/Contacts/Contact* elements.

B.1.2.119.1 Attributes

ManufacturerData/Contacts contains no attributes.

B.1.2.119.2 Child elements

ManufacturerData/Contacts contains the following child element:

Name	Subclause	Type	Use
Contact	B.1.2.120	—	1 .. ∞

B.1.2.120 ManufacturerData/Contacts/Contact

Properties: isRef 0, content complex

The *ManufacturerData/Contacts/Contact* child element shall identify the contact's email address, name, and telephone number.

B.1.2.120.1 Attributes

ManufacturerData/Contacts/Contact contains the following attributes:

Name	Type	Description	Use
email	*c:NonBlankString*	The email address for the contact.	Optional
name	*c:NonBlankString*	The contact's given name.	Required
phoneNumber	*c:NonBlankString*	The contact's telephone number.	Optional

B.1.2.120.2 Child elements

ManufacturerData/Contacts/Contact contains no child elements.

B.1.2.121 ManufacturerData/FaxNumber

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *ManufacturerData/FaxNumber* child element shall contain a textual representation of the facsimile telephone number of the manufacturer of the item.

B.1.2.121.1 Attributes

ManufacturerData/FaxNumber contains no attributes.

B.1.2.121.2 Child elements

ManufacturerData/FaxNumber contains no child elements.

B.1.2.122 ManufacturerData/MailingAddress

Base type: *c:MailingAddress*

Properties: isRef 0, content complex

The *ManufacturerData/MailingAddress* child element shall contain a textual representation of the postal mailing address of the manufacturer of the item.

B.1.2.122.1 Attributes

ManufacturerData/MailingAddress contains no attributes.

B.1.2.122.2 Child elements

ManufacturerData/MailingAddress inherits the child elements of *c:MailingAddress* (*Address1*, *Address2*, *City*, *Country*, *PostalCode*, and *State*).

B.1.2.123 ManufacturerData/URL

Base type: *c:NonBlankURI*

Properties: isRef 0, content simple

Facets: minLength 1

The *ManufacturerData/URL* child element shall contain the URL of the Web site for the manufacturer of the items.

B.1.2.123.1 Attributes

ManufacturerData/URL contains no attributes.

B.1.2.123.2 Child elements

ManufacturerData/URL contains no child elements.

B.1.2.124 ManufacturerIdentificationNumber

Base type: Extension of *c:IdentificationNumber*

Properties: base *c:IdentificationNumber*

The *ManufacturerIdentificationNumber* complex type shall be the base type for any XML schema element that will identify the manufacturer of an item.

B.1.2.124.1 Attributes

ManufacturerIdentificationNumber contains the following attributes:

Name	Type	Description	Use
manufacturerName	*c:NonBlankString*	A descriptive or common name for the manufacturer.	Required
number	*c:NonBlankString*	The part number of the entity.	Required
type	—	An indication of whether this is a part number, model number, or other.	Required

B.1.2.124.2 Child elements

ManufacturerIdentificationNumber contains no child elements.

B.1.2.125 NamedValue

Base type: Extension of *c:Value*

Properties: base *c:Value*

The *NamedValue* complex type shall be the base type for any XML schema element that will contain a data value with which a textual name must be associated.

B.1.2.125.1 Attributes

NamedValue contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the subject data value.	Required

B.1.2.125.2 Child elements

NamedValue inherits the child elements of *c:Value* (*Collection*, *Datum*, and *IndexedArray*).

B.1.2.126 octal

Base type: Extension of *c:DatumType*

Properties: base *c:DatumType*

The *octal* complex type shall be the base type for any XML schema elements of type *c:DatumType* that contain an octal-encoded binary value.

B.1.2.126.1 Attributes

octal contains the following attribute, in addition to those inherited from *c:DatumType* (*nonStandardUnit*, *standardUnit*, and *unitQualifier*):

Name	Type	Description	Use
value	xs:string	The octal representation of the numeric value. The attribute shall contain the character 0 followed by a finite-length sequence of characters 0–7.	Required

B.1.2.126.2 Child elements

octal inherits the child elements of *c:DatumType* (the group *c:DatumQuality*).

B.1.2.127 octalArray

Base type: Extension of *c:IndexedArrayType*

Properties: base *c:IndexedArrayType*

The *octalArray* complex type shall be the "`xsi:type`" of any element of type *c:IndexedArrayType* that contains an array of octal-encoded binary values.

B.1.2.127.1 Attributes

octalArray inherits the attributes of *c:IndexedArrayType* (*dimensions*, *nonStandardUnit*, *standardUnit*, and *unitQualifier*).

B.1.2.127.2 Child elements

octalArray contains the following child elements, in addition to those inherited from *c:IndexedArrayType* (the group *c:DatumQuality*):

Name	Subclause	Type	Use
DefaultElementValue	B.1.2.128	*c:octal*	Optional
Element	B.1.2.129	*c:octal*	Optional

B.1.2.128 octalArray/DefaultElementValue

Base type: *c:octal*

Properties: isRef 0, content complex

The *octalArray/DefaultElementValue* child element shall contain the default octal value of the array element.

B.1.2.128.1 Attributes

octalArray/DefaultElementValue inherits the attributes of *c:octal* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.1.2.128.2 Child elements

octalArray/DefaultElementValue inherits the child elements of *c:octal* (the group *c:DatumQuality*).

B.1.2.129 octalArray/Element

Base type: Extension of *c:octal*

Properties: isRef 0, content complex

The *octalArray/Element* child element shall contain the octal value of the array element.

B.1.2.129.1 Attributes

octalArray/Element contains the following attribute, in addition to those inherited from *c:octal* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*):

Name	Type	Description	Use
position	*c:ArrayIndexor*	The element value's index within the array.	Required

B.1.2.129.2 Child elements

octalArray/Element inherits the child elements of *c:octal* (the group *c:DatumQuality*).

B.1.2.130 Operator

The *Operator* complex type shall be the base type for any XML schema element that contains identifying information for the human operator of an ATE or other test equipment.

B.1.2.130.1 Attributes

Operator contains the following attributes:

Name	Type	Description	Use
ID	*c:NonBlankString*	A user-defined string uniquely identifying the subject *Operator*.	Required
name	*c:NonBlankString*	A descriptive or common name for the subject *Operator*.	Optional

B.1.2.130.2 Child elements

Operator contains the following child element:

Name	Subclause	Type	Use
OtherData	B.1.2.131	*c:NamedValue*	0 … ∞

B.1.2.131 Operator/OtherData

Base type: *c:NamedValue*

Properties: isRef 0, content complex

The *Operator/OtherData* child element shall contain information associated with the subject operator beyond that provided for in the parent element attributes.

B.1.2.131.1 Attributes

Operator/OtherData inherits the attribute of *c:NamedValue* (*name*).

B.1.2.131.2 Child elements

Operator/OtherData inherits the child elements of *c:NamedValue* (*Collection*, *Datum*, and *IndexedArray*).

B.1.2.132 Organization

The *Organization* complex type shall be the base type for any XML schema element that contains identifying information for an organization or entity.

B.1.2.132.1 Attributes

Organization contains the following attributes:

Name	Type	Description	Use
cageCode	*c:NonBlankString*	The CAGE code for the company indicated by the `name` attribute.	Optional
name	*c:NonBlankString*	A descriptive or common name for the manufacturer.	Required

B.1.2.132.2 Child elements

Organization contains the following child elements:

Name	Subclause	Type	Use
Address	B.1.2.133	*c:MailingAddress*	Optional
Contacts	B.1.2.134	—	Optional
FaxNumber	B.1.2.136	*c:NonBlankString*	Optional
URL	B.1.2.137	*c:NonBlankURI*	Optional
WorkCenter	B.1.2.138	—	Optional

B.1.2.133 Organization/Address

Base type: *c:MailingAddress*

Properties: isRef 0, content complex

The *Organization/Address* child element shall contain the mailing address of the manufacturer.

B.1.2.133.1 Attributes

Organization/Address contains no attributes.

B.1.2.133.2 Child elements

Organization/Address inherits the child elements of *c:MailingAddress* (*Address1*, *Address2*, *City*, *Country*, *PostalCode*, and *State*).

B.1.2.134 Organization/Contacts

Properties: isRef 0, content complex

The *Organization/Contacts* child element shall contain the contact information for the manufacturer of the item. This includes e-mail addresses and phone numbers.

B.1.2.134.1 Attributes

Organization/Contacts contains no attributes.

B.1.2.134.2 Child elements

Organization/Contacts contains the following child element:

Name	Subclause	Type	Use
Contact	B.1.2.135	*c:Person*	1 … ∞

B.1.2.135 Organization/Contacts/Contact

Base type: *c:Person*

Properties: isRef 0, content complex

The *Organization/Contacts/Contact* child element shall be a container of contact information.

B.1.2.135.1 Attributes

Organization/Contacts/Contact contains the following attributes:

Name	Type	Description	Use
affiliation	*c:NonBlankString*	The organization the contact represents.	Optional
email	*c:NonBlankString*	The contacts e-mail address.	Optional
ID	*c:NonBlankString*	A user-defined string uniquely identifying the contact.	Required
name	*c:NonBlankString*	A descriptive or common name for the operator.	Optional
phoneNumber	*c:NonBlankString*	The contacts telephone number.	Optional

B.1.2.135.2 Child elements

Organization/Contacts/Contact inherits the child elements of *c:Person* (*Address* and *OtherData*).

B.1.2.136 Organization/FaxNumber

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Organization/FaxNumber* child element shall contain the facsimile number of the manufacturer.

B.1.2.136.1 Attributes

Organization/FaxNumber contains no attributes.

B.1.2.136.2 Child elements

Organization/FaxNumber contains no child elements.

B.1.2.137 Organization/URL

Base type: *c:NonBlankURI*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Organization/URL* child element shall contain the uniform resource locator of the manufacturer.

B.1.2.137.1 Attributes

Organization/URL contains no attributes.

B.1.2.137.2 Child elements

Organization/URL contains no child elements.

B.1.2.138 Organization/WorkCenter

Properties: isRef 0, content complex

The *Organization/WorkCenter* child element shall identify the shop in which information was collected.

B.1.2.138.1 Attributes

Organization/WorkCenter contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the work center.	Required

B.1.2.138.2 Child elements

Organization/WorkCenter contains no child elements.

B.1.2.139 Person

Base type: Extension of *c:Operator*

Properties: base *c:Operator*

The *Person* complex type shall be the base type for any XML schema element that contains identifying information for a person.

B.1.2.139.1 Attributes

Person contains the following attributes, in addition to those inherited from *c:Operator* (*ID* and *name*):

Name	Type	Description	Use
affiliation	*c:NonBlankString*	The organization the person represents.	Optional
email	*c:NonBlankString*	The persons e-mail address.	Optional
phoneNumber	*c:NonBlankString*	The persons telephone number.	Optional

B.1.2.139.2 Child elements

Person contains the following child element:

Name	Subclause	Type	Use
Address	B.1.2.140	*c:MailingAddress*	Optional

B.1.2.140 Person/Address

Base type: *c:MailingAddress*

Properties: isRef 0, content complex

The *Person/Address* child element shall identify the mailing address for the person.

B.1.2.140.1 Attributes

Person/Address contains no attributes.

B.1.2.140.2 Child elements

Person/Address inherits the child elements of *c:MailingAddress* (*Address1*, *Address2*, *City*, *Country*, *PostalCode*, and *State*).

B.1.2.141 PhysicalInterface

The *PhysicalInterface* complex type shall be the base type for any XML schema element that contains identifying information for the physical interface of an ATE or other test equipment.

B.1.2.141.1 Attributes

PhysicalInterface contains no attributes.

B.1.2.141.2 Child elements

PhysicalInterface contains the following child elements:

Name	Subclause	Type	Use
Connectors	B.1.2.142	—	1 … ∞
Ports	B.1.2.144	—	1 … ∞

B.1.2.142 PhysicalInterface/Connectors

Properties: isRef 0, content complex

The *PhysicalInterface/Connectors* child element shall be a collector for an unbounded set of child *PhysicalInterface/Connectors/Connector* elements.

B.1.2.142.1 Attributes

PhysicalInterface/Connectors contains no attributes.

B.1.2.142.2 Child elements

PhysicalInterface/Connectors contains the following child element:

Name	Subclause	Type	Use
Connector	B.1.2.143	*c:Connector*	1 … ∞

B.1.2.143 PhysicalInterface/Connectors/Connector

Base type: *c:Connector*

Properties: isRef 0, content complex

The *PhysicalInterface/Connectors/Connector* child element shall identify a physical connector of a hardware item.

B.1.2.143.1 Attributes

PhysicalInterface/Connectors/Connector inherits the attributes of *c:Connector* (*ID*, *location*, *matingConnectorType*, *name*, *type*, and *version*).

B.1.2.143.2 Child elements

PhysicalInterface/Connectors/Connector inherits the child elements of *c:Connector* (*Description*, *Extension*, and *Identification*).

B.1.2.144 PhysicalInterface/Ports

Properties: isRef 0, content complex

The *PhysicalInterface/Ports* child element shall be a collector for an unbounded set of child *c:Port* elements.

B.1.2.144.1 Attributes

PhysicalInterface/Ports contains no attributes.

B.1.2.144.2 Child elements

PhysicalInterface/Ports contains the following child element:

Name	Subclause	Type	Use
Port	B.1.2.145	*c:Port*	1 … ∞

B.1.2.145 PhysicalInterface/Ports/Port

Base type: Extension of *c:Port*

Properties: isRef 0, content complex

The *PhysicalInterface/Ports/Port* child element shall identify a physical port of a hardware item.

B.1.2.145.1 Attributes

PhysicalInterface/Ports/Port inherits the attributes of *c:Port* (*direction*, *name*, and *type*).

B.1.2.145.2 Child elements

PhysicalInterface/Ports/Port contains the following child element, in addition to those inherited from *c:Port* (*Extension*):

Name	Subclause	Type	Use
ConnectorPins	B.1.2.146	—	Optional

B.1.2.146 PhysicalInterface/Ports/Port/ConnectorPins

Properties: isRef 0, content complex

The *PhysicalInterface/Ports/ConnectorPins* child element shall be a collector for an unbounded set of *c:ConnectorPin* child elements.

B.1.2.146.1 Attributes

PhysicalInterface/Ports/ConnectorPins contains no attributes.

B.1.2.146.2 Child elements

PhysicalInterface/Ports/ConnectorPins contains the following child element:

Name	Subclause	Type	Use
ConnectorPin	B.1.2.147	*c:ConnectorLocation*	1 … ∞

B.1.2.147 PhysicalInterface/Ports/Port/ConnectorPins/ConnectorPin

Base type: *c:ConnectorLocation*

Properties: isRef 0, content complex

The *PhysicalInterface/Ports/ConnectorPins/ConnectorPin* child element shall identify a physical pin of a connector.

B.1.2.147.1 Attributes

PhysicalInterface/Ports/ConnectorPins/ConnectorPin inherits the attributes of *c:ConnectorLocation* (*connectorID* and *pinID*).

B.1.2.147.2 Child elements

PhysicalInterface/Ports/ConnectorPins/ConnectorPin contains no child elements.

B.1.2.148 Port

The *Port* complex type shall be the base type for any XML schema element that contains identifying information for the port of an ATE or other test equipment.

B.1.2.148.1 Attributes

Port contains the following attributes:

Name	Type	Description	Use
direction	*c:PortDirection*	An enumeration providing for the specification of the direction in which data moves on the described port. Enumeration values are Input, Output, and Bi-Directional.	Optional
name	*c:NonBlankString*	A descriptive or common name for the port.	Required
type	*c:PortType*	An enumeration providing for the specification of the type of the described port. Enumeration values are Ground, Analog, Digital, Power, Optical, or Software.	Optional

B.1.2.148.2 Child elements

Port contains the following child element:

Name	Subclause	Type	Use
Extension	B.1.2.149	*c:Extension*	Optional

B.1.2.149 Port/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *Port/Extension* child element shall contain a specific extension point for use cases that require elements not provided in the basic structure.

B.1.2.149.1 Attributes

Port/Extension contains no attributes.

B.1.2.149.2 Child elements

Port/Extension inherits the child element of *c:Extension* (*##other*).

B.1.2.150 SingleLimit

Base type: Extension of *c:Value*

Properties: base *c:Value*

The *SingleLimit* complex type shall be the base type of any element that will contain a single limit value used in a comparison.

B.1.2.150.1 Attributes

SingleLimit contains the following attribute:

Name	Type	Description	Use
comparator	*c:ComparisonOperator*	A limit describes a boundary. There may be uses cases where comparisons are made with multiple values. In such cases, these multiple value comparisons may be combined with logical AND or OR operators.	Required

B.1.2.150.2 Child elements

SingleLimit inherits the child elements of *c:Value* (*Collection*, *Datum*, and *IndexedArray*).

B.1.2.151 SoftwareInstance

Base type: Extension of *c:ItemInstance*

Properties: base *c:ItemInstance*

The *SoftwareInstance* complex type shall be the base type for any XML schema element that is intended to capture identification information specifying a single instance of a software item.

B.1.2.151.1 Attributes

SoftwareInstance contains no attributes.

B.1.2.151.2 Child elements

SoftwareInstance contains the following child element, in addition to those inherited from *c:ItemInstance* (*Definition*, *DescriptionDocumentReference*, and *SerialNumber*):

Name	Subclause	Type	Use
ReleaseDate	B.1.2.152	xs:date	Optional

B.1.2.152 SoftwareInstance/ReleaseDate

Base type: xs:date

Properties: isRef 0, content simple

The *SoftwareInstance/ReleaseDate* child element shall contain the actual release date of the referenced software item.

B.1.2.152.1 Attributes

SoftwareInstance/ReleaseDate contains no attributes.

B.1.2.152.2 Child elements

SoftwareInstance/ReleaseDate contains no child elements.

B.1.2.153 string

Base type: Extension of *c:DatumType*

Properties: base *c:DatumType*

The *string* complex type shall be the "`xsi:type`" of any attribute or an element of type *c:DatumType* that contains a string value.

B.1.2.153.1 Attributes

string inherits the attributes of *c:DatumType* (*nonStandardUnit*, *standardUnit*, and *unitQualifier*).

B.1.2.153.2 Child elements

string contains the following child element, in addition to those inherited from *c:DatumType* (*Confidence*, *ErrorLimits*, *Range*, and *Resolution*):

Name	Subclause	Type	Use
Value	B.1.2.154	xs:string	Required

B.1.2.154 string/Value

Base type: xs:string

Properties: isRef 0, content simple

The *string/Value* child element shall contain the value of a string.

B.1.2.154.1 Attributes

string/Value contains no attributes.

B.1.2.154.2 Child elements

string/Value contains no child elements.

B.1.2.155 stringArray

Base type: Extension of *c:IndexedArrayType*

Properties: base *c:IndexedArrayType*

The *stringArray* complex type shall be the "`xsi:type`" of any element of type *c:IndexedArrayType* that contains an array of string values.

B.1.2.155.1 Attributes

stringArray inherits the attributes of *c:IndexedArrayType* (*dimensions*, *nonStandardUnit*, *standardUnit*, and *unitQualifier*)

B.1.2.155.2 Child elements

stringArray contains the following child elements, in addition to those inherited from *c:IndexedArrayType* (the group *c:DatumQuality*):

Name	Subclause	Type	Use
DefaultElementValue	B.1.2.156	*c:string*	Optional
Element	B.1.2.157	*c:string*	0 … ∞

B.1.2.156 stringArray/DefaultElementValue

Base type: *c:string*

Properties: isRef 0, content complex

The *stringArray/DefaultElementValue* child element shall contain the default string value of the array element.

B.1.2.156.1 Attributes

stringArray/DefaultElementValue inherits the attributes of *c:string* (*nonStandardUnit*, *standardUnit*, and *unitQualifier*).

B.1.2.156.2 Child elements

stringArray/DefaultElementValue inherits the child elements of *c:string* (*Confidence*, *ErrorLimits*, *Range*, *Resolution*, and *Value*).

B.1.2.157 stringArray/Element

Base type: Extension of *c:string*

Properties: isRef 0, content complex

The *stringArray/Element* child element shall contain the string value of the array element.

B.1.2.157.1 Attributes

stringArray/Element contains the following attribute, in addition to those inherited from *c:string* (*nonStandardUnit*, *standardUnit*, and *unitQualifier*).

Name	Type	Description	Use
position	*c:ArrayIndexor*	The element value's index within the array.	Required

B.1.2.157.2 Child elements

stringArray/Element inherits the child elements of *c:string* (*Confidence*, *ErrorLimits*, *Range*, *Resolution*, and *Value*).

B.1.2.158 unsignedInteger

Base type: Extension of *c:DatumType*

Properties: base *c:DatumType*

The *unsignedInteger* complex type shall be the "`xsi:type`" for elements of type *c:DatumType* that contain a 32-bit unsigned integer value.

B.1.2.158.1 Attributes

unsignedInteger contains the following attribute, in addition to those inherited from *c:DatumType* (*nonStandardUnit*, *standardUnit*, and *unitQualifier*):

Name	Type	Description	Use
value	xs:unsignedInt	The numeric value, between 0 and 4 294 967 295 (inclusive).	Required

B.1.2.158.2 Child elements

unsignedInteger inherits the child elements of *c:DatumType* (*Confidence*, *ErrorLimits*, *Range*, *Resolution*, and *Value*).

B.1.2.159 unsignedIntegerArray

Base type: Extension of *c:IndexedArrayType*

Properties: base *c:IndexedArrayType*

The *unsignedIntegerArray* complex type shall be the "`xsi:type`" of any element of type *c:IndexedArrayType* that contains an array of unsigned integer values.

B.1.2.159.1 Attributes

unsignedIntegerArray inherits the attributes of *c:IndexedArrayType* (*dimensions*, *nonStandardUnit*, *standardUnit*, and *unitQualifier*).

B.1.2.159.2 Child elements

unsignedIntegerArray contains the following child elements, in addition to those inherited from *c:IndexedArrayType* (the group *c:DatumQuality*):

Name	Subclause	Type	Use
DefaultElementValue	B.1.2.160	*c:unsignedInteger*	Optional
Element	B.1.2.161	*c:unsignedInteger*	Optional

B.1.2.160 unsignedIntegerArray/DefaultElementValue

Base type: *c:unsignedInteger*

Properties: isRef 0, content complex

The *unsignedIntegerArray/DefaultElementValue* child element shall contain the default unsigned integer value of the array element.

B.1.2.160.1 Attributes

unsignedIntegerArray/DefaultElementValue inherits the attributes of *c:unsignedInteger* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.1.2.160.2 Child elements

unsignedIntegerArray/DefaultElementValue inherits the child elements of *c:DatumType* (the group *c:DatumQuality*).

B.1.2.161 unsignedIntegerArray/Element

Base type: Extension of *c:unsignedInteger*

Properties: isRef 0, content complex

The *unsignedIntegerArray/Element* child element shall contain the unsigned integer value of the array element.

B.1.2.161.1 Attributes

unsignedIntegerArray/Element contains the following attribute, in addition to those inherited from *c:unsignedInteger* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*):

Name	Type	Description	Use
position	*c:ArrayIndexor*	The element value's index within the array.	Required

B.1.2.161.2 Child elements

unsignedIntegerArray/Element inherits the child elements of *c:unsignedInteger* (the group *c:DatumQuality*).

B.1.2.162 unsignedLong

Base type: Extension of *c:DatumType*

Properties: base *c:DatumType*

The *unsignedLong* complex type shall be the "`xsi:type`" for elements of type *c:DatumType* that contain a 64-bit unsigned integer value.

B.1.2.162.1 Attributes

unsignedLong contains the following attribute, in addition to those inherited from c:DatumType (*standardUnit*, *nonStandardUnit*, and *unitQualifier*).

Name	Type	Description	Use
value	xs:long	The numeric value, between 0 and 18 466 744 073 709 551 615 (inclusive).	Required

B.1.2.162.2 Child elements

unsignedLong inherits the child elements of *c:DatumType* (the group *c:DatumQuality*).

B.1.2.163 unsignedLongArray

Base type: Extension of *c:IndexedArrayType*

Properties: base *c:IndexedArrayType*

The *unsignedLongArray* complex type shall be the "`xsi:type`" for elements of type *c:IndexedArrayType* that contain an array of 32-bit unsigned integer values.

B.1.2.163.1 Attributes

unsignedLongArray inherits the attributes from *c:IndexedArrayType* (*dimensions*, *standardUnit*, *nonStandardUnit*, and *unitQualifier*).

B.1.2.163.2 Child elements

unsignedLongArray contains the following child elements, in addition to those inherited from *c:IndexedArrayType* (the group *c:DatumQuality*).

Name	Subclause	Type	Use
DefaultElementValue	B.1.2.164	*c:unsignedLong*	Optional
Element	B.1.2.165	*c:unsignedLong*	0 … ∞

B.1.2.164 unsignedLongArray/DefaultElementValue

Base type: *c:unsignedLong*

Properties: isRef 0, content complex

The *unsignedLongArray/DefaultElementValue* child element shall contain the default integer value of the array element.

B.1.2.164.1 Attributes

unsignedLongArray/DefaultElementValue inherits the attributes of *c:unsignedLong* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.1.2.164.2 Child elements

unsignedLongArray/DefaultElementValue inherits the child elements of *c:unsignedLong* (the group *c:DatumQuality*).

B.1.2.165 unsignedLongArray/Element

Base type: Extension of *c:unsignedLong*

Properties: isRef 0, content complex

The *unsignedLongArray/Element* child element shall contain the integer value of the array element.

B.1.2.165.1 Attributes

unsignedLongArray/Element contains the following attribute, in addition to those inherited from *c:unsignedLong* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*):

Name	Type	Description	Use
position	*c:ArrayIndexor*	The element value's index within the array.	Required

B.1.2.165.2 Child elements

unsignedLong/Element inherits the child elements of *c:unsignedLong* (the group *c:DatumQuality*).

B.1.2.166 UserDefinedIdentificationNumber

Base type: Extension of *c:IdentificationNumber*

Properties: base *c:IdentificationNumber*

The *UserDefinedIdentificationNumber* complex type shall be the base type for any XML schema element that will identify an item.

B.1.2.166.1 Attributes

UserDefinedIdentificationNumber contains the following attributes:

Name	Type	Description	Use
number	*c:NonBlankString*	The part number of the entity.	Required
qualifier	*c:NonBlankString*	An adjective providing additional descriptive data that further specify or identify the 'number' attribute. Example: the identification number specified by the	Required

		user.	
type	—	An indication of whether this is a part number, model number, or other.	Required

B.1.2.166.2 Child elements

UserDefinedIdentificationNumber contains no child elements.

B.1.2.167 Value

The *Value* complex type shall be utilized for XML schema elements that contain values (e.g., boolean, numeric, date-time, string, collections, arrays). Different child elements shall be used to represent a single data value, a collection of data values, or an array of data values.

B.1.2.167.1 Attributes

Value contains no attributes.

B.1.2.167.2 Child elements

Value contains one of the following child elements:

	Name	Subclause	Type	Use
Choice	Collection	B.1.2.168	*c:Collection*	Required
	Datum	B.1.2.169	*c:DatumType*	
	IndexedArray	B.1.2.170	*c:IndexedArrayType*	
NOTE—Choice indicates that only one of these elements may be specified.				

B.1.2.168 Value/Collection

Base type: *c:Collection*

Properties: isRef 0, content complex

The *Value/Collection* child element shall contain a group of data values that constitute a single entity or set.

B.1.2.168.1 Attributes

Value/Collection inherits the attributes of *c:Collection* (*defaultNonStandardUnit*, *defaultStandardUnit*, and *defaultUnitQualifier*).

B.1.2.168.2 Child elements

Value/Collection inherits the child elements of *c:Collection* (the group *c:DatumQuality*).

B.1.2.169 Value/Datum

Base type: *c:DatumType*

Properties: isRef 0, content complex

The *Value/Datum* child element shall contain a single data value (boolean, numeric, date-time, or string).

B.1.2.169.1 Attributes

Value/Datum inherits the attributes of *c:DatumType* (*nonStandardUnit*, *standardUnit*, and *unitQualifier*).

B.1.2.169.2 Child elements

Value/Datum inherits the child elements of *c:Collection* (the group *c:DatumQuality* and element *c:Item*).

B.1.2.170 Value/IndexedArray

Base type: *c:IndexedArrayType*

Properties: isRef 0, content complex

The *Value/IndexedArray* child element shall contain multidimensional arrays of data. This information includes simple name/value pairs as well as more complex matrices.

B.1.2.170.1 Attributes

Value/IndexedArray inherits the attributes of *c:IndexedArrayType* (*dimensions*, *nonStandardUnit*, *standardUnit*, and *unitQualifier*).

B.1.2.170.2 Child elements

Value/IndexedArray inherits the child elements of *c:Collection* (the group *c:DatumQuality*).

B.1.2.171 WorkOrder

The *WorkOrder* complex type shall be utilized for the identification of a work order related to, or authorizing, the testing of the UUT.

B.1.2.171.1 Attributes

WorkOrder contains no attributes.

B.1.2.171.2 Child elements

WorkOrder contains the following child elements:

Name	Subclause	Type	Use
Description	B.1.2.172	*c:NonBlankString*	Optional
Extension	B.1.2.173	*c:Extension*	Optional
MaintenanceLevel	B.1.2.174	—	Optional
WorkItemNumber	B.1.2.175	*c:NonBlankString*	Optional
WorkOrderNumber	B.1.2.176	*c:NonBlankString*	Required

B.1.2.172 WorkOrder/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *WorkOrder/Description* child element shall contain the narrative for the work order.

B.1.2.172.1 Attributes

WorkOrder/Description contains no attributes.

B.1.2.172.2 Child elements

WorkOrder/Description contains no child elements.

B.1.2.173 WorkOrder/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *WorkOrder/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.1.2.173.1 Attributes

WorkOrder/Extension contains no attributes.

B.1.2.173.2 Child elements

WorkOrder/Extension inherits the child element of *c:Extension* (*##other*).

B.1.2.174 WorkOrder/MaintenanceLevel

Properties: isRef 0, content complex

The *WorkOrder/MaintenanceLevel* child element shall identify the level of maintenance.

B.1.2.174.1 Attributes

WorkOrder/MaintenanceLevel contains the following attributes:

Name	Type	Description	Use
abbreviation	*c:NonBlankString*	The abbreviation for the level of maintenance. Examples: I and D.	Required
name	*c:NonBlankString*	A descriptive or common name for the level of maintenance. Examples: Intermediate and Depot.	Optional

B.1.2.174.2 Child elements

WorkOrder/MaintenanceLevel contains no child elements.

B.1.2.175 WorkOrder/WorkItemNumber

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *WorkOrder/WorkItemNumber* child element shall identify the instance document or collection of information at a particular level of maintenance.

B.1.2.175.1 Attributes

WorkOrder/WorkItemNumber contains no attributes.

B.1.2.175.2 Child elements

WorkOrder/WorkItemNumber contains no child elements.

B.1.2.176 WorkOrder/WorkOrderNumber

Base type: c:NonBlankString

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *WorkOrder/WorkOrderNumber* child element shall identify the instance document or collection of information across multiple levels of maintenance.

B.1.2.176.1 Attributes

WorkOrder/WorkOrderNumber contains no attributes.

B.1.2.176.2 Child elements

WorkOrder/WorkOrderNumber contains no child elements.

B.1.3 Simple types

B.1.3.1 ArrayIndexor

Base type: restriction of xs:string

Properties: final restriction

Regular expression: \[(([0-9]+)((,([0-9]+))*)\] (Restricts contents to a comma-delimited set of decimal numbers.)

This type shall be used as the base type of any attribute or element that specifies the size of an array or the index of an element within an array. In use, attributes derived from this element shall contain a string designating an *n*-dimensional array index or array dimension, with the format [*a*,*b*,*c*,…,*n*], where *a*,*b*,*c*,…*n* are numeric indices. When a derived attribute specifies the size of an array, the attribute shall indicate the maximum size of each dimension of the array. When a derived attribute indicates a specific element of an array, the index value(s) shall be zero-based ordinal numbers. Examples: (element index: [0] or [0,1] or [2,2,0]; maximum array index: [2,3] or [3,3,3]). Indexes shall be only positive; in other words, no negative indexing is permitted.

B.1.3.2 ComparisonOperator

Base type: restriction of xs:string

Enumerations: GT | GE | LT | LE

This type shall be used as the base type for any XML schema attribute or element that specifies a comparison operator between two elements.

B.1.3.3 EqualityComparisonOperator

Base type: restriction of xs:string

Enumerations: EQ | NE | CIEQ | CINE

This type shall be used as the base type for any XML schema attribute or element that specifies a comparison operator between two elements.

EQ = Case-sensitive equal for strings, equal for all other datatypes.

NE = Case-sensitive not-equal for strings, not equal for all other datatypes.

CIEQ = Case-insensitive equal for strings, equal for all other datatypes.

CINE = Case-insensitive not-equal for strings, not equal for all other datatypes.

B.1.3.4 HexValue

Base type: restriction of xs:string

Regular expression: (0[x|X])?([0-9]|[a-f]|[A-F])* (Restricts contents to a hexadecimal number.)

This type shall be used as the base type for any XML schema attribute or element that contains a hex-encoded binary value. This binary value contains the optional string **0x** followed by a finite-length sequence of characters 0–9 and a–f.

B.1.3.5 LogicalOperator

Base type: restriction of xs:string

Enumerations: AND | OR

This type shall be used as the base type for any XML schema attribute or element that specifies a boolean logic combination of two elements.

B.1.3.6 MaskOperator

Base type: restriction of xs:string

Enumerations: AND | OR | XOR

This type shall be used as the base type for any XML schema attribute or element that specifies a boolean logic combination of two mask values.

B.1.3.7 NonBlankString

Base type: restriction of xs:string

This type shall be used as the base type of any XML schema attribute or element that is required to be nonblank. This type uses the XML `<xs:minLength value="1"/>` specification to create a non-nullable string, i.e., a string that must contain at least one character. Also, white space will be collapsed (i.e., multiple space characters will be replaced with a single space).

B.1.3.8 NonBlankURI

Base type: restriction of xs:anyURI

This type shall be used as the base type of any XML schema attribute or element that is intended to contain a nonblank uniform resource identifier (URI). This type uses the XML `<xs:minLength value="1"/>` specification to create a non-nullable string, i.e., a string that must contain at least one character. Also, white space will be collapsed (i.e., multiple space characters will be replaced with a single space).

B.1.3.9 PortDirection

Base type: restriction of xs:string

Enumerations: Input | Output | Bi-Directional

This type shall be used as the base type for any XML schema attribute or element that describes a port and requires specification of data movement direction on that port.

B.1.3.10 PortType

Base type: restriction of xs:string

Enumerations: Ground | Analog | Digital | Power | Optical | Software

This type shall be used as the base type for any XML schema attribute or element that describes a port and requires specification of the type of the port.

B.1.3.11 StandardUnit

Base type: restriction of xs:string

This type is defined only as a convenience. The StandardUnit type shall be used by any XML schema attribute or element that contains the unit of measure for a numerical value (e.g., volts, ohms, MHz). The contents of this attribute shall be compliant with IEEE Std 260.1 [B11].

B.1.3.12 Uuid

Base type: restriction of xs:string

Pattern: [A-Fa-f0-9]{32}|(\{|\()?[A-Fa-f0-9]{8}-([A-Fa-f0-9]{4}-){3}[A-Fa-f0-9]{12}(\}|\))?

This type is used by other XML schema attributes or elements that will hold a universal unique identifier (UUID), commonly known as either a globally unique identifier (GUID) or UUID. The regular expression defined limits the contents of an attribute to either a single 32-digit hexadecimal string or a 32-digit hex string patterned as [8]-[4]-[4]-[4]-[12] digits.

B.1.4 Attribute groups

B.1.4.1 ClassifiedAttributes

This attribute group shall be used by all XML schemas that require security classification identification.

Name	Type	Description	Use
classified	xs:Boolean	An indication that the element is or is not classified.	Optional
securityClassification	*c:NonBlankString*	A use-case-determined string declaring the security classification level of the element containing this attribute and the subordinate branch of the XML document.	Optional

B.1.4.2 DocumentRootAttributes

In accordance with Annex A, this attribute group shall be used as the root element for all XML schemas.

This attribute group includes the following attribute, in addition to those inherited from the *c:ClassifiedAttributes* attribute group (*classified* and *securityClassification*).

Name	Type	Description	Use
uuid	*c:Uuid*	As defined in c:Uuid, uuid is a universal unique identifier for the element containing this attribute.	Required

B.1.4.3 RepeatedItemAttributes

This attribute group shall be used as the root element of all XML schemas that provide for the duplication of an item.

Name	Type	Description	Use
baseIndex	xs:int	Starting index for the items.	Optional
count	xs:int	Number of identical items.	Optional
incrementedBy	xs:int	Specifies the value by which the items are to be incremented.	Optional
replacementCharacter	*c:NonBlankString*	Specifies the character replacement in association with the calculated index.	Optional

B.1.4.4 UnitAttributes

In nearly all ATS use cases, strictly limiting units of measure to SI or English units is restrictive. In numerous cases, it is desirable to qualify a unit with an additional text string, e.g., Peak-to-Peak or RMS for voltage measurements. This attribute group allows for the inclusion of a standard SI unit of measure (as defined in IEEE Std 260.1 [B11]), a nonstandard unit of measure, and a qualifier thereto.

Name	Type	Description	Use
nonStandardUnit	*c:NonBlankString*	Any nonstandard unit not already defined in IEEE Std 260.1	Optional
standardUnit	*c:StandardUnit*	When used, this attribute shall contain only a unit of measure defined in IEEE Std 260.1	Optional
unitQualifier	*c:NonBlankString*	A textual qualifier that is to be applied to the attribute of either the standardUnit or nonStandardUnit. Examples: RMS or Peak-to-Peak for a standardUnit of volts.	Optional
NOTE—If one is not sure if a particular unit being utilized is standard or nonstandard, assume it is nonstandard, and represent it as a nonStandardUnit.			

B.1.5 Groups

B.1.5.1 DatumQuality

The *DatumQuality* group shall be used by any element that requires the specification of any of the group's child elements.

B.1.5.1.1 Attributes

DatumQuality contains no attributes.

B.1.5.1.2 Child elements

DatumQuality contains the following child elements:

Name	Subclause	Type	Use
Confidence	B.1.5.2	xs:double	Optional
ErrorLimits	B.1.5.3	*c:Limit*	Optional
Range	B.1.5.4	*c:Limit*	Optional
Resolution	B.1.5.5	xs:double	Optional

B.1.5.2 DatumQuality/Confidence

Base type: xs:double

Properties: isRef 0, content simple

The *DatumQuality/Confidence* child element shall contain the required confidence.

B.1.5.2.1 Attributes

DatumQuality/Confidence contains no attributes.

B.1.5.2.2 Child elements

DatumQuality/Confidence contains no child elements.

B.1.5.3 DatumQuality/ErrorLimits

Base type: *c:Limit*

Properties: isRef 0, content complex

The *DatumQuality/ErrorLimits* child element shall contain the error limits.

B.1.5.3.1 Attributes

DatumQuality/ErrorLimits inherits the attributes of *c:Limit* (*name* and *operator*).

B.1.5.3.2 Child elements

DatumQuality/ErrorLimits inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.1.5.4 DatumQuality/Range

Base type: *c:Limit*
Properties: isRef 0, content complex

The *DatumQuality/Range* child element shall contain the range.

B.1.5.4.1 Attributes

DatumQuality/Range inherits the attributes of *c:Limit* (*name* and *operator*).

B.1.5.4.2 Child elements

DatumQuality/Range inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.1.5.5 DatumQuality/Resolution

Base type: xs:double

Properties: isRef 0, content simple

The *DatumQuality/Resolution* child element shall contain the required resolution.

B.1.5.5.1 Attributes

DatumQuality/Resolution contains no attributes.

B.1.5.5.2 Child elements

DatumQuality/Resolution contains no child elements.

B.2 Common element schema—HardwareCommon.xsd

target namespace	urn:IEEE-1671:2010:HardwareCommon
version	3.12
imported schema	urn:IEEE-1671:2010:Common

A standard XSD that is intended as the source of an XML instance document shall contain a single root element. The **HardwareCommon XML schema is a reference XML schema containing only type definitions** that may be used in other XML schemas. **It has no root element, and there will be no instance documents directly validated against the HardwareCommon XML schema**.

ATML HardwareCommon imports ATML Common (see B.1); only the ATML HardwareCommon unique elements are defined within this clause.

B.2.1 Elements

None

B.2.2 Complex types

B.2.2.1 AnalogTriggerPropertyGroup

Base type: Extension of *hc:TriggerPropertyGroup*

Properties: base *hc:TriggerPropertyGroup*

The *AnalogTriggerPropertyGroup* complex type shall be the base type for XML schema elements intended to document properties of an analog signal-based trigger.

B.2.2.1.1 Attributes

AnalogTriggerPropertyGroup inherits the attribute of *hc:TriggerPropertyGroup* (*name*).

B.2.2.1.2 Child elements

AnalogTriggerPropertyGroup contains the following child element, in addition to those inherited from *hc:TriggerPropertyGroup* (*Description* and *Extension*):

Name	Subclause	Type	Use
Level	B.2.2.2	*hc:LevelType*	Required

B.2.2.2 AnalogTriggerPropertyGroup/Level

Base type: *hc:LevelType*

Properties: isRef 0, content complex

The *AnalogTriggerPropertyGroup/Level* child element shall identify an analog trigger level (value, dimensions, and resolution).

B.2.2.2.1 Attributes

AnalogTriggerPropertyGroup/Level inherits the attributes of *hc:LevelType* (*numberOfBits*, *units*, and *value*).

B.2.2.2.2 Child elements

AnalogTriggerPropertyGroup/Level contains no child elements.

B.2.2.3 Capabilities

The *Capabilities* complex type shall be used as the base type for XML schema elements intended to document capabilities and interconnections of a hardware item.

B.2.2.3.1 Attributes

Capabilities contains no attributes.

B.2.2.3.2 Child elements

Capabilities contains the following child elements:

	Name	Subclause	Type	Use
Choice	CapabilitiesReference	B.2.2.4	*c:DocumentReference*	1 … ∞
	Capability	B.2.2.5	*hc:Capability*	
	CapabilityMap	B.2.2.6	*hc:CapabilityMap*	Required
NOTE—Choice indicates that only one of these elements may be specified.				

B.2.2.4 Capabilities/CapabilitiesReference

Base type: *c:DocumentReference*

Properties: isRef 0, content complex

The *Capabilities/CapabilitiesReference* child element shall identify an external document containing a description of the capabilities.

B.2.2.4.1 Attributes

Capabilities/CapabilitiesReference inherits the attributes of *c:DocumentReference* (*ID* and *uuid*).

B.2.2.4.2 Child elements

Capabilities/CapabilitiesReference contains no child elements.

B.2.2.5 Capabilities/Capability

Base type: *hc:Capability*

Properties: isRef 0, content complex

The *Capabilities/Capability* child element shall identify the capability and interface of a hardware item.

B.2.2.5.1 Attributes

Capabilities/Capability inherits the attribute of *hc:Capability* (*name*).

B.2.2.5.2 Child elements

Capabilities/Capability inherits the child elements of *hc:Capability* (*Description*, *Extension*, *Interface*, and *SignalDescription*).

B.2.2.6 Capabilities/CapabilityMap

Base type: *hc:CapabilityMap*

Properties: isRef 0, content complex

The *Capabilities/CapabilityMap* child element shall identify how the hardware item is connected.

B.2.2.6.1 Attributes

Capabilities/CapabilityMap contains no attributes.

B.2.2.6.2 Child elements

Capabilities/CapabilityMap inherits the child element of *hc:CapabilityMap* (*Mapping*).

B.2.2.7 Capability

Base type: Extension of *hc:Item*

Properties: base *hc:Item*

The *Capability* complex type shall identify a specific capability and interface of a hardware item.

B.2.2.7.1 Attributes

Capability inherits the attribute of *hc:Item* (*name*).

B.2.2.7.2 Child elements

Capability contains the following child elements, in addition to those inherited from *hc:Item* (*Description* and *Extension*):

Name	Subclause	Type	Use
Interface	B.2.2.8	*c:Interface*	Required
SignalDescription	B.2.2.9	*c:Extension*	Optional

B.2.2.8 Capability/Interface

Base type: *c:Interface*

Properties: isRef 0, content complex

The *Capability/Interface* child element shall identify the interface (as ports and optionally connectors) to the *c:Capability/SignalDescription* of the hardware item.

B.2.2.8.1 Attributes

Capability/Interface contains no attributes.

B.2.2.8.2 Child elements

Capability/Interface inherits the child element of *c:Interface* (*Ports*).

B.2.2.9 Capability/SignalDescription

Base type: *c:Extension*

Properties: isRef 0, content complex

The *Capability/SignalDescription* child element shall identify the signal capability at the *c:Capability/Interface* interface of the hardware item.

B.2.2.9.1 Attributes

Capability/SignalDescription contains no attributes.

B.2.2.9.2 Child elements

Capability/SignalDescription inherits the child element of *c:Extension* (*##other*).

B.2.2.10 CapabilityMap

The *CapabilityMap* complex type shall document the mapping of capabilities to interfaces.

B.2.2.10.1 Attributes

CapabilityMap contains no attributes.

B.2.2.10.2 Child elements

CapabilityMap contains the following child element:

Name	Subclause	Type	Use
Mapping	B.2.2.11	*hc:Mapping*	1 … ∞

B.2.2.11 CapabilityMap/Mapping

Base type: *hc:Mapping*

Properties: isRef 0, content complex

The *CapabiliyMap/Mapping* child element shall identify the capability to interface mapping.

B.2.2.11.1 Attributes

CapabiliyMap/Mapping inherits the attributes of *hc:Mapping* (*baseIndex*, *count*, *incrementedBy*, and *replacementCharacter*).

B.2.2.11.2 Child elements

CapabiliyMap/Mapping inherits the child element of *hc:Mapping* (*Map*).

B.2.2.12 Characteristic

Base type: Extension of *hc:Specification*

Properties: base *hc:Specification*

The *Characteristic* complex type shall describe the performance that may be expected from the instrument.

B.2.2.12.1 Attributes

Characteristic inherits the attribute of *hc:Specification* (*name*).

B.2.2.12.2 Child elements

Characteristic inherits the child elements of *hc:Specification* (*Conditions*, *Definition*, *Description*, *ExclusiveOptions*, *Graph*, *Limits*, and *SupplementalInformation*).

B.2.2.13 ControlLanguage

Properties: abstract true

The *ControlLanguage* complex type shall be the base type for XML schema elements intended to document control languages. Derived types include standard commands for programmable instrumentation (SCPI), message based, and register based.

B.2.2.13.1 Attributes

ControlLanguage contains no attributes.

B.2.2.13.2 Child elements

ControlLanguage contains the following child element:

Name	Subclause	Type	Use
Documentation	B.2.2.14	*c:Document*	Optional

B.2.2.14 ControlLanguage/Documentation

Base type: *c:Document*

Properties: isRef 0, content complex

The *ControlLanguage/Documentation* child element shall document control languages.

B.2.2.14.1 Attributes

ControlLanguage/Documentaiton inherits the attributes of *c:Document* (*name* and *uuid*).

B.2.2.14.2 Child elements

ControlLanguage/Documentation inherits the child elements of *c:Document* (*Extension*, *Text*, and *URL*).

B.2.2.15 CrossPointSwitch

Base type: Extension of *hc:Item*

Properties: base *hc:Item*

The *CrossPointSwitch* complex type shall be the base type for XML schema elements intended to document properties of a cross point switch.

B.2.2.15.1 Attributes

CrossPointSwitch contains the following attribute, in addition to those inherited from *hc:Item* (*name*):

Name	Type	Description	Use
lineCount	xs:int	The number of matrix lines available to connect the rows and columns.	Required

B.2.2.15.2 Child elements

CrossPointSwitch contains the following child elements, in addition to those inherited from *hc:Item* (*Description* and *Extension*):

Name	Subclause	Type	Use
Columns	B.2.2.16	—	Required
Rows	B.2.2.18	—	Required

B.2.2.16 CrossPointSwitch/Columns

Properties: isRef 0, content complex

The *CrossPointSwitch/Columns* child element shall document properties of the columns of a cross point switch.

B.2.2.16.1 Attributes

CrossPointSwitch/Columns contains no attributes.

B.2.2.16.2 Child elements

CrossPointSwitch/Columns contains the following child element:

Name	Subclause	Type	Use
Port	B.2.2.17	*hc:SwitchPort*	1 … ∞

B.2.2.17 CrossPointSwitch/Columns/Port

Base type: *hc:SwitchPort*

Properties: isRef 0, content complex

The *CrossPointSwitch/Columns/Port* child element shall document properties of the port in the columns of the cross point switch.

B.2.2.17.1 Attributes

CrossPointSwitch/Columns/Port inherits the attributes of *hc:SwitchPort* (*baseIndex*, *count*, *incrementedBy*, *name*, and *replacementCharacter*).

B.2.2.17.2 Child elements

CrossPointSwitch/Columns/Port inherits the child elements of *hc:SwitchPort* (*Description*, *Extension*, and *Pin*).

B.2.2.18 CrossPointSwitch/Rows

Properties: isRef 0, content complex

The *CrossPointSwitch/Rows* child element shall document properties of the rows of a cross point switch.

B.2.2.18.1 Attributes

CrossPointSwitch/Rows contains no attributes.

B.2.2.18.2 Child elements

CrossPointSwitch/Rows contains the following child element:

Name	Subclause	Type	Use
Port	B.2.2.19	*hc:SwitchPort*	1 … ∞

B.2.2.19 CrossPointSwitch/Rows/Port

Base type: *hc:SwitchPort*

Properties: isRef 0, content complex

The *CrossPointSwitch/Rows/Port* child element shall document properties of the port in the rows of the cross point switch.

B.2.2.19.1 Attributes

CrossPointSwitch/Rows/Port inherits the attributes of *hc:SwitchPort* (*baseIndex*, *count*, *incrementedBy*, *name*, and *replacementCharacter*).

B.2.2.19.2 Child elements

CrossPointSwitch/Rows/Port inherits the child elements of *hc:SwitchPort* (*Description*, *Extension*, and *Pin*).

B.2.2.20 DetectionType

The *DetectionType* complex type shall be the base type for XML schema elements intended to document properties of a digital trigger. The properties shall be either edge detection or level detection.

B.2.2.20.1 Attributes

DetectionType contains the following attributes:

Name	Type	Description	Use
edgeDetection	*hc:DigitalEdge*	An identification of the digital edge that shall be present for the trigger to occur. The edge shall be Rising, Falling, or Selectable.	Optional
levelDetection	*hc:DigitalLevel*	An identification of the digital level that shall be present for the trigger to occur. The level shall be High, Low, or Selectable.	Optional

B.2.2.20.2 Child elements

DetectionType contains no child elements.

B.2.2.21 DigitalTriggerPropertyGroup

Base type: Extension of *hc:TriggerPropertyGroup*

Properties: base *hc:TriggerPropertyGroup*

The *DigitalTriggerPropertyGroup* complex type shall be the base type for XML schema elements intended to document properties of a digital signal-based trigger.

B.2.2.21.1 Attributes

DigitalTriggerPropertyGroup inherits the attribute of *hc:TriggerPropertyGroup* (*name*).

B.2.2.21.2 Child elements

DigitalTriggerPropertyGroup contains the following child elements, in addition to those inherited from *hc:TriggerPropertyGroup* (*Description* and *Extension*):

Name	Subclause	Type	Use
Detection	B.2.2.22	*hc:DetectionType*	Required
MinPulseWidth	B.2.2.23	*hc:MinPulseWidthType*	Required

B.2.2.22 DigitalTriggerPropertyGroup/Detection

Base type: *hc:DetectionType*

Properties: isRef 0, content complex

The *DigitalTriggerPropertyGroup/Detection* child element shall identify the properties of a digital trigger.

B.2.2.22.1 Attributes

DigitalTriggerPropertyGroup/Detection inherits the attributes of *hc:DetectionType* (*edgeDetection* and *levelDetection*).

B.2.2.22.2 Child elements

DigitalTriggerPropertyGroup/Detection contains no child elements.

B.2.2.23 DigitalTriggerPropertyGroup/MinPulseWidth

Base type: *hc:MinPulseWidthType*

Properties: isRef 0, content complex

The *DigitalTriggerPropertyGroup/MinPulseWidth* child element shall identify the minimum pulse-width of the digital trigger.

B.2.2.23.1 Attributes

DigitalTriggerPropertyGroup/MinPulseWidth inherits the attributes of *hc:MinPulseWidthType* (*units* and *value*).

B.2.2.23.2 Child elements

DigitalTriggerPropertyGroup/MinPulseWidth contains no child elements.

B.2.2.24 Driver

Properties: abstract true

The *Driver* complex type shall be the base type for XML schema elements intended to document instrument drivers.

B.2.2.24.1 Attributes

Driver contains no attributes.

B.2.2.24.2 Child elements

Driver contains one of the following child elements:

	Name	Subclause	Type	Use
Choice	Bit16	B.2.2.25	*hc:DriverModule*	Required
	Bit32	B.2.2.26	*hc:DriverModule*	
	Bit64	B.2.2.27	*hc:DriverModule*	
	Unified	B.2.2.28	—	
NOTE—Choice indicates that only one of these elements may be specified.				

B.2.2.25 Driver/Bit16

Base type: *hc:DriverModule*

Properties: isRef 0, content complex

The *Driver/Bit16* child element shall identify the 16-bit instrument driver.

B.2.2.25.1 Attributes

Driver/Bit16 inherits the attributes of *hc:DriverModule* (*filename* and *installationDirectory*).

B.2.2.25.2 Child elements

Driver/Bit16 contains no child elements.

B.2.2.26 Driver/Bit32

Base type: *hc:DriverModule*

Properties: isRef 0, content complex

The *Driver/Bit32* child element shall identify the 32-bit instrument driver.

B.2.2.26.1 Attributes

Driver/Bit32 inherits the attributes of *hc:DriverModule* (*filename* and *installationDirectory*).

B.2.2.26.2 Child elements

Driver/Bit32 contains no child elements.

B.2.2.27 Driver/Bit64

Base type: *hc:DriverModule*

Properties: isRef 0, content complex

The *Driver/Bit64* child element shall identify the 64-bit instrument driver.

B.2.2.27.1 Attributes

Driver/Bit64 inherits the attributes of *hc:DriverModule* (*filename* and *installationDirectory*).

B.2.2.27.2 Child elements

Driver/Bit64 contains no child elements.

B.2.2.28 Driver/Unified

Properties: isRef 0, content complex

The *Driver/Unified* child element shall identify the unified instrument driver (e.g., a driver that can function as either a 32-bit or a 64-bit driver).

B.2.2.28.1 Attributes

Driver/Unified contains no attributes.

B.2.2.28.2 Child elements

Driver/Unified contains the following child elements:

Name	Subclause	Type	Use
Bit32	B.2.2.29	*hc:DriverModule*	Required
Bit64	B.2.2.30	*hc:DriverModule*	Required

B.2.2.29 Driver/Unified/Bit32

Base type: *hc:DriverModule*

Properties: isRef 0, content complex

The *Driver/Unified/Bit32* child element shall identify the unified 32-bit instrument driver.

B.2.2.29.1 Attributes

Driver/Unified/Bit32 inherits the attributes of *hc:DriverModule* (*filename* and *installationDirectory*).

B.2.2.29.2 Child elements

Driver/Unified/Bit32 contains no child elements.

B.2.2.30 Driver/Unified/Bit64

Base type: *hc:DriverModule*

Properties: isRef 0, content complex

The *Driver/Unified/Bit64* child element shall identify the unified 64-bit instrument driver.

B.2.2.30.1 Attributes

Driver/Unified/Bit64 inherits the attributes of *hc:DriverModule* (*filename* and *installationDirectory*).

B.2.2.30.2 Child elements

Driver/Unified/Bit64 contains no child elements.

B.2.2.31 DriverModule

The *DriverModule* complex type shall be the base type for XML schema elements intended to identify instrument driver executables. For example, all forms of interchangeable virtual instrumentation (IVI) drivers are software executables.

B.2.2.31.1 Attributes

DriverModule contains the following attributes:

Name	Type	Description	Use
fileName	*c:NonBlankString*	A descriptive or common computer-based name for the driver.	Required
installationDirectory	*c:NonBlankString*	A descriptive or common computer-based path to the directory where the driver is installed.	Optional

B.2.2.31.2 Child elements

DriverModule contains no child elements.

B.2.2.32 FacilitiesRequirements

The *FacilityRequirements* complex type shall be the base type for XML schema elements intended to document properties of the facility required to perform testing.

B.2.2.32.1 Attributes

FacilitiesRequirements contains no attributes.

B.2.2.32.2 Child elements

FacilityRequirements contains the following child elements:

Name	Subclause	Type	Use
Cooling	B.2.2.33	xs:string	Optional
Extension	B.2.2.34	*c:Extension*	Optional
FacilitiesInterface	B.2.2.35	*c:Interface*	Optional
FacilityRequirementsDocuments	B.2.2.36	—	Optional
Hydraulic	B.2.2.38	xs:string	Optional
Pneumatic	B.2.2.39	xs:string	Optional

B.2.2.33 FacilitiesRequirements/Cooling

Base type: xs:string

Properties: isRef 0, content simple

The *FacilitiesRequirements/Cooling* child element shall identify any cooling requirements of the hardware item.

B.2.2.33.1 Attributes

FacilitiesRequirements/Cooling contains no attributes.

B.2.2.33.2 Child elements

FacilitiesRequirements/Cooling contains no child elements.

B.2.2.34 FacilitiesRequirements/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *FacilitiesRequirements/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.2.2.34.1 Attributes

FacilitiesRequirements/Extension contains no attributes.

B.2.2.34.2 Child elements

FacilitiesRequirements/Extension inherits the child element of *c:Extension* (*##other*).

B.2.2.35 FacilitiesRequirements/FacilitiesInterface

Base type: *c:Interface*

Properties: isRef 0, content complex

The *FacilitiesRequirements/FacilitiesInterface* child element shall identify any nonpower interfaces (in the form of a *c:Interface*) of the hardware item.

B.2.2.35.1 Attributes

FacilitiesRequirements/FacilitiesInterface contains no attributes.

B.2.2.35.2 Child elements

FacilitiesRequirements/FacilitiesInterface inherits the child element of *c:Interface* (*Ports*).

B.2.2.36 FacilitiesRequirements/FacilityRequirementsDocuments

Properties: isRef 0, content complex

The *FacilitiesRequirements/FacilityRequirementsDocuments* child element shall identify all of the facility's requirements documents for the hardware item.

B.2.2.36.1 Attributes

FacilitiesRequirements/FacilityRequirementsDocuments contains no attributes.

B.2.2.36.2 Child elements

FacilitiesRequirements/FacilityRequirementsDocuments contains the following child element:

Name	Subclause	Type	Use
FacilitiesRequirementsDocument	B.2.2.37	*c:Document*	1 …∞

B.2.2.37 FacilitiesRequirements/FacilityRequirementsDocuments/FacilitiesRequirementsDocument

Base type: *c:Document*

Properties: isRef 0, content complex

The *FacilitiesRequirements/FacilityRequirementsDocuments/FacilitiesRequirementsDocument* element shall identify the facility's requirements document for the hardware item.

B.2.2.37.1 Attributes

FacilitiesRequirements/FacilityRequirementsDocuments/FacilitiesRequirementsDocument inherits the attributes of *c:Document* (*name* and *uuid*).

B.2.2.37.2 Child elements

FacilitiesRequirements/FacilityRequirementsDocuments/FacilitiesRequirementsDocument inherits the child elements of *c:Document* (*Extension*, *Text*, and *URL*).

B.2.2.38 FacilitiesRequirements/Hydraulic

Base type: xs:string

Properties: isRef 0, content simple

The *FacilitiesRequirements/Hydraulic* child element shall identify hydraulic requirements of the hardware item.

B.2.2.38.1 Attributes

FacilitiesRequirements/Hydraulic contains no attributes.

B.2.2.38.2 Child elements

FacilitiesRequirements/Hydraulic contains no child elements.

B.2.2.39 FacilitiesRequirements/Pneumatic

Base type: xs:string

Properties: isRef 0, content simple

The *FacilitiesRequirements/Pneumatic* child element shall identify pneumatic requirements of the hardware item.

B.2.2.39.1 Attributes

FacilitiesRequirements/Pneumatic contains no attributes.

B.2.2.39.2 Child elements

FacilitiesRequirements/Pneumatic contains no child elements.

B.2.2.40 Feature

Base type: Extension of *hc:Specification*

Properties: base *hc:Specification*

The *Feature* complex type shall be the base type for XML schema elements intended to document the features of the instrument not described within a performance description (see *hc:Characteristic*).

B.2.2.40.1 Attributes

Feature inherits the attribute of *hc:Specification* (*name*).

B.2.2.40.2 Child elements

Feature inherits the child elements of *hc:Specification* (*Conditions*, *Definition*, *Description*, *ExclusiveOptions*, *Graph*, *Limits*, *RequiredOptions*, and *SupplementalInformation*).

B.2.2.41 Generic

Base type: Extension of *hc:ControlLanguage*

Properties: base *hc:ControlLanguage*

The *Generic* complex type shall be the base type for XML schema elements intended to identify the document that contains the instrument's generic control language specification.

B.2.2.41.1 Attributes

Generic contains no attributes.

B.2.2.41.2 Child elements

Generic inherits the child element of *hc:ControlLanguage* (*Documentation)*.

B.2.2.42 Guaranteed

Base type: Extension of *hc:Specification*

Properties: base *hc:Specification*

The *Guaranteed* complex type shall be the base type for XML schema elements intended to document the specifications of the hardware item that are the basis for determining whether the hardware item is in need of repair. Should the hardware item not meet the specifications provided by *Guaranteed*, the hardware item should be classified as either unhealthy or not functioning.

B.2.2.42.1 Attributes

Guaranteed inherits the attribute of *hc:Specification* (*name*).

B.2.2.42.2 Child elements

Guaranteed inherits the child elements of *hc:Specification* (*Conditions*, *Definition*, *Description*, *ExclusiveOptions*, *Graph*, *Limits*, *RequiredOptions*, and *SupplementalInformation*).

B.2.2.43 HardwareItemDescription

Base type: Extension of *c:ItemDescription*

Properties: base *c:ItemDescription*, abstract true

The *HardwareItemDescription* complex type shall be the base type for XML schema elements intended to describe hardware entities. Derived types include InstrumentDescription.xsd, UUTDescription.xsd, TestStation.xsd, and TestAdapter.xsd.

B.2.2.43.1 Attributes

HardwareItemDescription inherits the attributes of *c:ItemDescription* (*name* and *version*).

B.2.2.43.2 Child elements

HardwareItemDescription contains the following child elements, in addition to those inherited from *c:ItemDescription* (*Description*, *Extension*, and *Identification*):

Name	Subclause	Type	Use
CalibrationRequirements	B.2.2.44	—	Optional
ConfigurationOptions	B.2.2.48	—	Optional
Control	B.2.2.50	—	Optional
Components	B.2.2.74	—	Optional
Documentation	B.2.2.76	—	Optional
EnvironmentalRequirements	B.2.2.78	*c:EnvironmentalRequirements*	Optional
Errors	B.2.2.79	—	Optional
FactoryDefaults	B.2.2.82	—	Optional
Interface	B.2.2.84	*c:PhysicalInterface*	Required
LegalDocuments	B.2.2.85	—	Optional
NetworkList	B.2.2.91	—	Optional
OperationalRequirements	B.2.2.93	*hc:OperationalRequirements*	Optional
ParentComponents	B.2.2.94	—	Optional
PhysicalCharacteristics	B.2.2.96	*hc:PhysicalCharacteristics*	Optional
PowerRequirements	B.2.2.97	*hc:PowerSpecifications*	Optional

B.2.2.44 HardwareItemDescription/CalibrationRequirements

Properties: isRef 0, content complex

The *HardwareItemDescription/CalibrationRequirements* child element shall identify the calibration requirements of the hardware item.

B.2.2.44.1 Attributes

HardwareItemDescription/CalibrationRequirements contains no attributes.

B.2.2.44.2 Child elements

HardwareItemDescription/CalibrationRequirements contains the following child element:

Name	Subclause	Type	Use
CalibrationRequirement	B.2.2.45	—	1 … ∞

B.2.2.45 HardwareItemDescription/CalibrationRequirements/CalibrationRequirement

Properties: isRef 0, content complex

The *HardwareItemDescription/CalibrationRequirements/CalibrationRequirement* child element shall identify both the support equipment needed to run calibration and the calibration procedure.

B.2.2.45.1 Attributes

HardwareItemDescription/CalibrationRequirements/CalibrationRequirement contains the following attribute:

Name	Type	Description	Use
frequency	xs:duration	An indication of how often calibration shall be performed.	Required

B.2.2.45.2 Child elements

HardwareItemDescription/CalibrationRequirements/CalibrationRequirement contains the following child elements:

Name	Subclause	Type	Use
Procedure	B.2.2.46	*c:Document*	1 … ∞
SupportEquipment	B.2.2.47	*c:NonBlankString*	0 … ∞

B.2.2.46 HardwareItemDescription/CalibrationRequirements/CalibrationRequirement/Procedure

Base type: *c:Document*

Properties: isRef 0, content complex

The *HardwareItemDescription/CalibrationRequirements/CalibrationRequirement/Procedure* child element shall identify the calibration procedure.

B.2.2.46.1 Attributes

HardwareItemDescription/CalibrationRequirements/CalibrationRequirement/Procedure inherits the attributes of *c:Document* (*name* and *uuid*).

B.2.2.46.2 Child elements

HardwareItemDescription/CalibrationRequirements/CalibrationRequirement/Procedure inherits the child elements of *c:Document* (*Extension*, *Text*, and *URL*).

B.2.2.47 HardwareItemDescription/CalibrationRequirements/CalibrationRequirement/SupportEquipment

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *HardwareItemDescription/CalibrationRequirements/CalibrationRequirement/SuppoertEquipment* child element shall identify the support equipment needed to run calibration.

B.2.2.47.1 Attributes

HardwareItemDescription/CalibrationRequirements/CalibrationRequirement/SupportEquipment contains no attributes.

B.2.2.47.2 Child elements

HardwareItemDescription/CalibrationRequirements/CalibrationRequirement/SupportEquipment contains no child elements.

B.2.2.48 HardwareItemDescription/ConfigurationOptions

Properties: isRef 0, content complex

The *HardwareItemDescription/ConfigurationOptions* child element shall identify the configuration option(s) of the hardware item. These options are values the user can modify, which will persist after a power cycle of the hardware item.

B.2.2.48.1 Attributes

HardwareItemDescription/ConfigurationOptions contains no attributes.

B.2.2.48.2 Child elements

HardwareItemDescription/ConfigurationOptions contains the following child element:

Name	Subclause	Type	Use
Option	B.2.2.49	—	1 … ∞

B.2.2.49 HardwareItemDescription/ConfigurationOptions/Option

Properties: isRef 0, content complex

The *HardwareItemDescription/ConfigurationOptions/Option* child element shall identify the name of the configuration item.

B.2.2.49.1 Attributes

HardwareItemDescription/ConfigurationOptions/Option contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the hardware item value the user can modify, which will persist after a power cycle.	Required

B.2.2.49.2 Child elements

HardwareItemDescription/ConfigurationOptions/Option contains no child elements.

B.2.2.50 HardwareItemDescription/Control

Properties: isRef 0, content complex

The *HardwareItemDescription/Control* child element shall be a collector element of control languages, drivers, extension, firmwares, and tools for the hardware item.

B.2.2.50.1 Attributes

HardwareItemDescription/Control contains no attributes.

B.2.2.50.2 Child elements

HardwareItemDescription/Control contains the following child elements:

Name	Subclause	Type	Use
ControlLanguages	B.2.2.51	—	Optional
Drivers	B.2.2.53	—	Optional
Extension	B.2.2.66	*c:Extension*	Optional
Firmwares	B.2.2.67	—	Optional
Tools	B.2.2.69	—	Optional

B.2.2.51 HardwareItemDescription/Control/ControlLanguages

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/ControlLanguages* child element shall identify the control language(s) of the hardware item.

B.2.2.51.1 Attributes

HardwareItemDescription/ControlLanguages contains no attributes.

B.2.2.51.2 Child elements

HardwareItemDescription/ControlLanguages contains the following child element:

Name	Subclause	Type	Use
ControlLanguage	B.2.2.52	*hc:ControlLanguage*	1 … ∞

B.2.2.52 HardwareItemDescription/Control/ControlLanguages/ControlLanguage

Base type: *hc:ControlLanguage*

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/ControlLanguages/ControlLanguage* child element shall identify the control language for the hardware item.

B.2.2.52.1 Attributes

HardwareItemDescription/ControlLanguages/ControlLanguage contains no attributes.

B.2.2.52.2 Child elements

HardwareItemDescription/Control/ControlLanguages/ControlLanguage inherits the child element of *hc:ControlLanguage* (*Documentation*).

B.2.2.53 HardwareItemDescription/Control/Drivers

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Drivers* child element shall identify the software interface driver(s) of the hardware item.

B.2.2.53.1 Attributes

HardwareItemDescription/Control/Drivers contains no attributes.

B.2.2.53.2 Child elements

HardwareItemDescription/Control/Drivers contains the following child element:

Name	Subclause	Type	Use
Driver	B.2.2.54	*hc:VersionIdentifier*	1 … ∞

B.2.2.54 HardwareItemDescription/Control/Drivers/Driver

Base type: Extension of *hc:VersionIdentifier*

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Drivers/Driver* child element shall identify the software interface driver for the hardware item.

B.2.2.54.1 Attributes

HardwareItemDescription/Control/Drivers/Driver inherits the attributes of *hc:VersionIdentifier* (*name*, *qualifier*, and *version*).

B.2.2.54.2 Child elements

HardwareItemDescription/Control/Drivers/Driver contains the following child elements:

Name	Subclause	Type	Use
Dependencies	B.2.2.55	—	Optional
Extension	B.2.2.58	*c:Extension*	Optional
Manufacturer	B.2.2.59	*c:ManufacturerData*	Optional
Platform	B.2.2.60	—	Required
Type	B.2.2.65	*hc:Driver*	Required

B.2.2.55 HardwareItemDescription/Control/Drivers/Driver/Dependencies

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Drivers/Driver/Dependencies* child element shall identify software and/or firmware dependencies.

B.2.2.55.1 Attributes

HardwareItemDescription/Control/Drivers/Driver/Dependencies contains no attributes.

B.2.2.55.2 Child elements

HardwareItemDescription/Control/Drivers/Driver/Dependencies contains the following child elements:

Name	Subclause	Type	Use
Firmware	B.2.2.56	*hc:VersionIdentifier*	0 … ∞
Software	B.2.2.57	*hc:VersionIdentifier*	0 … ∞

B.2.2.56 HardwareItemDescription/Control/Drivers/Driver/Dependencies/Firmware

Base type: *hc:VersionIdentifier*

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Drivers/Driver/Dependencies/Firmware* child element shall identify the firmware dependency.

B.2.2.56.1 Attributes

HardwareItemDescription/Control/Drivers/Driver/Dependencies/Firmware inherits the attributes of *hc:VersionIdentifier* (*name*, *qualifier*, and *version*).

B.2.2.56.2 Child elements

HardwareItemDescription/Control/Drivers/Driver/Dependencies/Firmware contains no child elements.

B.2.2.57 HardwareItemDescription/Control/Drivers/Driver/Dependencies/Software

Base type: *hc:VersionIdentifier*

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Drivers/Driver/Dependencies/Software* child element shall identify the software dependency.

B.2.2.57.1 Attributes

HardwareItemDescription/Control/Drivers/Driver/Dependencies/Software inherits the attributes of *hc:VersionIdentifier* (*name*, *qualifier*, and *version*).

B.2.2.57.2 Child elements

HardwareItemDescription/Control/Drivers/Driver/Dependencies/Software contains no child elements.

B.2.2.58 HardwareItemDescription/Control/Drivers/Driver/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Drivers/Driver/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.2.2.58.1 Attributes

HardwareItemDescription/Control/Drivers/Driver/Extension contains no attributes.

B.2.2.58.2 Child elements

HardwareItemDescription/Control/Drivers/Driver/Extension inherits the child element of *c:Extension* (*##other*).

B.2.2.59 HardwareItemDescription/Control/Drivers/Driver/Manufacturer

Base type: *c:ManufacturerData*

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Drivers/Driver/Manufacturer* child element shall identify the developer of the driver.

B.2.2.59.1 Attributes

HardwareItemDescription/Control/Drivers/Driver/Manufacturer inherits the attributes of *c:ManufacturerData* (*cageCode* and *name*).

B.2.2.59.2 Child elements

HardwareItemDescription/Control/Drivers/Driver/Manufacturer inherits the child elements of *c:ManufacturerData* (*Contacts*, *FaxNumber*, *MailingAddress*, and *URL*).

B.2.2.60 HardwareItemDescription/Control/Drivers/Driver/Platform

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Drivers/Driver/Platform* child element shall identify computing hardware requirements needed in order for the driver to execute.

B.2.2.60.1 Attributes

HardwareItemDescription/Control/Drivers/Driver/Platform contains no attributes.

B.2.2.60.2 Child elements

HardwareItemDescription/Control/Drivers/Driver/Platform contains the following child elements:

Name	Subclause	Type	Use
HardDisk	B.2.2.61	—	Optional
OperatingSystem	B.2.2.62	*hc:VersionIdentifier*	1 … ∞
PhysicalMemory	B.2.2.63	—	Optional
Processor	B.2.2.64	—	Optional

B.2.2.61 HardwareItemDescription/Control/Drivers/Driver/Platform/HardDisk

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Drivers/Driver/Platform/HardDisk* child element shall identify computer hard disk requirements needed in order for the driver to be stored and to execute.

B.2.2.61.1 Attributes

HardwareItemDescription/Control/Drivers/Driver/Platform/HardDisk contains the following attribute:

Name	Type	Description	Use
minimum	*c:NonBlankString*	The minimum storage capacity needed to store the software driver and permit its execution. Example: 6 GB.	Optional

B.2.2.61.2 Child elements

HardwareItemDescription/Control/Drivers/Driver/Platform/HardDisk contains no child elements.

B.2.2.62 HardwareItemDescription/Control/Drivers/Driver/Platform/OperatingSystem

Base type: Extension of *hc:VersionIdentifier*

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Drivers/Driver/Platform/OperatingSystem* child element shall identify computer operating system requirements needed in order for the driver to execute.

B.2.2.62.1 Attributes

HardwareItemDescription/Control/Drivers/Driver/Platform/OperatingSystem contains the following child element, in addition to those inherited from *hc:VersionIdentifier* (*name*, *qualifier*, and *version*):

Name	Type	Description	Use
servicePack	*c:NonBlankString*	The operating system's service pack identification. Example: Service Pack 1.1.	Optional

B.2.2.62.2 Child elements

HardwareItemDescription/Control/Drivers/Driver/Platform/OperatingSystem contains no child elements.

B.2.2.63 HardwareItemDescription/Control/Drivers/Driver/Platform/PhysicalMemory

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Drivers/Driver/Platform/PhysicalMemory* child element shall identify computer physical memory requirements needed in order for the driver to execute.

B.2.2.63.1 Attributes

HardwareItemDescription/Control/Drivers/Driver/Platform/PhysicalMemory contains the following attribute:

Name	Type	Description	Use
minimum	*c:NonBlankString*	The minimum physical memory capacity needed for the software driver to execute. Example: 512 MB.	Optional

B.2.2.63.2 Child elements

HardwareItemDescription/Control/Drivers/Driver/Platform/PhysicalMemory contains no child elements.

B.2.2.64 HardwareItemDescription/Control/Drivers/Driver/Platform/Processor

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Drivers/Driver/Platform/Processor* child element shall identify computer processor speed requirements needed in order for the driver to execute.

B.2.2.64.1 Attributes

HardwareItemDescription/Control/Drivers/Driver/Platform/Processor contains the following attribute:

Name	Type	Description	Use
speed	*c:NonBlankString*	The minimum clock speed of the processor required for the software driver to execute. Example: 10 GHz or greater Acme-3 processor.	Optional

B.2.2.64.2 Child elements

HardwareItemDescription/Control/Drivers/Driver/Platform/Processor contains no child elements.

B.2.2.65 HardwareItemDescription/Control/Drivers/Driver/Type

Base type: *hc:Driver*

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Drivers/Driver/Type* child element shall identify the name and location of the driver.

B.2.2.65.1 Attributes

HardwareItemDescription/Control/Drivers/Driver/Type inherits the attributes of *hc:Driver* (*Bit16*, *Bit32*, *Bit64*, and *Unified*).

Child elements

HardwareItemDescription/Control/Drivers/Driver/Type contains no child elements.

B.2.2.66 HardwareItemDescription/Control/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.2.2.66.1 Attributes

HardwareItemDescription/Control/Extension contains no attributes.

B.2.2.66.2 Child elements

HardwareItemDescription/Control/Extension inherits the child element of *c:Extension* (*##other*).

B.2.2.67 HardwareItemDescription/Control/Firmwares

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Firmwares* child element shall identify the firmware(s) of the hardware item.

B.2.2.67.1 Attributes

HardwareItemDescription/Control/Firmwares contains no attributes.

B.2.2.67.2 Child elements

HardwareItemDescription/Control/Firmwares contains the following child element:

Name	Subclause	Type	Use
Firmware	B.2.2.68	*hc:VersionIdentifier*	1 … ∞

B.2.2.68 HardwareItemDescription/Control/Firmwares/Firmware

Base type: *hc:VersionIdentifier*

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Firmwares/Firmware* child element shall identify firmware for the hardware item.

B.2.2.68.1 Attributes

HardwareItemDescription/Control/Firmwares/Firmware inherits the attributes of *hc:VersionIdentifier* (*name*, *qualifier*, and *version*).

B.2.2.68.2 Child elements

HardwareItemDescription/Control/Firmwares/Firmware contains no child elements.

B.2.2.69 HardwareItemDescription/Control/Tools

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Tools* child element shall identify all software tools associated with the hardware item.

B.2.2.69.1 Attributes

HardwareItemDescription/Control/Tools contains no attributes.

B.2.2.69.2 Child elements

HardwareItemDescription/Control/Tools contains the following child element:

Name	Subclause	Type	Use
Tool	B.2.2.70	*hc:VersionIdentifier*	1 … ∞

B.2.2.70 HardwareItemDescription/Control/Tools/Tool

Base type: Extension of *hc:VersionIdentifier*

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Tools/Tool* child element shall identify a software tool associated with the hardware item.

B.2.2.70.1 Attributes

HardwareItemDescription/Control/Tools/Tool contains the following attribute, in addition to those inherited from *hc:VersionIdentifier* (*name*, *qualifier*, and *version*):

Name	Type	Description	Use
filePath	*c:NonBlankString*	The location of the software tool, within the operating system structure, on the hard disk.	Optional

B.2.2.70.2 Child elements

HardwareItemDescription/Control/Tools/Tool contains the following child element:

Name	Subclause	Type	Use
Dependencies	B.2.2.71	—	Optional

B.2.2.71 HardwareItemDescription/Control/Tools/Tool/Dependencies

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Tools/Tool/Dependencies* child element shall identify a tool's software and/or driver dependencies.

B.2.2.71.1 Attributes

HardwareItemDescription/Control/Tools/Tool/Dependencies contains no attributes.

B.2.2.71.2 Child elements

HardwareItemDescription/Control/Tools/Tool/Dependencies contains one of the following child elements:

	Name	Subclause	Type	Use
Choice	Driver	B.2.2.72	*hc:VersionIdentifier*	1 …∞
	Software	B.2.2.73	*hc:VersionIdentifier*	
NOTE—Choice indicates that only one of these elements may be specified.				

B.2.2.72 HardwareItemDescription/Control/Tools/Tool/Dependencies/Driver

Base type: Extension of *hc:VersionIdentifier*

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Tools/Tool/Dependencies/Driver* child element shall identify a tool's driver dependency.

B.2.2.72.1 Attributes

HardwareItemDescription/Control/Tools/Tool/Dependencies/Driver inherits the attributes of *hc:VersionIdentifier* (*name*, *qualifier*, and *version*).

B.2.2.72.2 Child elements

HardwareItemDescription/Control/Tools/Tool/Dependencies/Driver contains no child elements.

B.2.2.73 HardwareItemDescription/Control/Tools/Tool/Dependencies/Software

Base type: Extension of *hc:VersionIdentifier*

Properties: isRef 0, content complex

The *HardwareItemDescription/Control/Tools/Tool/Dependencies/Software* child element shall identify a tool's software dependency.

B.2.2.73.1 Attributes

HardwareItemDescription/Control/Tools/Tool/Dependencies/Software inherits the attributes of *hc:VersionIdentifier* (*name*, *qualifier*, and *version*).

B.2.2.73.2 Child elements

HardwareItemDescription/Control/Tools/Tool/Dependencies/Software contains no child elements.

B.2.2.74 HardwareItemDescription/Components

Properties: isRef 0, content complex

The *HardwareItemDescription/Components* child element shall be a collector element of the identification of the subassemblies to the subject hardware item.

B.2.2.74.1 Attributes

HardwareItemDescription/Components contains no attributes.

B.2.2.74.2 Child elements

HardwareItemDescription/Components contains the following child element:

Name	Subclause	Type	Use
Component	B.2.2.75	*c:ItemDescriptionReference*	1 … ∞

B.2.2.75 HardwareItemDescription/Components/Component

Base type: Extension of *c:ItemDescriptionReference*

Properties: isRef 0, content complex

The *HardwareItemDescription/Components/Component* child element shall identify a subassembly of the subject hardware item.

B.2.2.75.1 Attributes

HardwareItemDescription/Components/Component contains the following attribute:

Name	Type	Description	Use
ID	*c:NonBlankString*	A user-defined string uniquely identifying the subassembly. Example: Pre-Amp A1.	Required

B.2.2.75.2 Child elements

HardwareItemDescription/Components/Component inherits the child elements of *c:ItemDescriptionReference* (*Definition* and *DescriptionDocumentReference*).

B.2.2.76 HardwareItemDescription/Documentation

Properties: isRef 0, content complex

The *HardwareItemDescription/Documentation* child element shall be a collector element of the documentation of the subject hardware item to be assembled.

B.2.2.76.1 Attributes

HardwareItemDescription/Documentation contains no attributes.

B.2.2.76.2 Child elements

HardwareItemDescription/Documentation contains the following child element:

Name	Subclause	Type	Use
Document	B.2.2.77	*c:Document*	1 … ∞

B.2.2.77 HardwareItemDescription/Documentation/Document

Base type: *c:Document*

Properties: isRef 0, content complex

The *HardwareItemDescription/Documentation/Document* child element shall identify a document for the subject hardware item.

B.2.2.77.1 Attributes

HardwareItemDescription/Documentation/Document inherits the attributes of *c:Document* (*name* and *uuid*).

B.2.2.77.2 Child elements

HardwareItemDescription/Documentation/Document inherits the child elements of *c:Document* (*Extension*, *Text*, and *URL*).

B.2.2.78 HardwareItemDescription/EnvironmentalRequirements

Base type: *c:EnvironmentalRequirements*

Properties: isRef 0, content complex

The *HardwareItemDescription/EnvironmentalRequirements* child element shall identify the operational and/or storage and transport requirements for the hardware item.

B.2.2.78.1 Attributes

HardwareItemDescription/EnvironmentalRequirements contains no attributes.

B.2.2.78.2 Child elements

HardwareItemDescription/EnvironmentalRequirements inherits the child elements of *c:EnvironmentalRequirements* (*Operation* and *StorageRequirements*).

B.2.2.79 HardwareItemDescription/Errors

Properties: isRef 0, content complex

The *HardwareItemDescription/Errors* child element shall identify the type, source, and identification of all errors associated with the hardware item.

B.2.2.79.1 Attributes

HardwareItemDescription/Errors contains no attributes.

B.2.2.79.2 Child elements

HardwareItemDescription/Errors contains the following child element:

Name	Subclause	Type	Use
Error	B.2.2.80	—	1 … ∞

B.2.2.80 HardwareItemDescription/Errors/Error

Properties: isRef 0, content complex

The *HardwareItemDescription/Errors/Error* child element shall identify an error associated with the hardware item.

B.2.2.80.1 Attributes

HardwareItemDescription/Errors/Error contains the following attributes:

Name	Type	Description	Use
ID	*c:NonBlankString*	A user-defined string uniquely identifying the error. Example: built-in test (BIT) #5 failure.	Required
source	*c:NonBlankString*	A descriptive or common name for the source of the error. Example: Power-up BIT.	Optional
type	*hc:ErrorType*	The severity of the error. Example: Fatal.	Optional

B.2.2.80.2 Child elements

HardwareItemDescription/Errors/Error contains the following child element:

Name	Subclause	Type	Use
Description	B.2.2.81	*c:NonBlankString*	Required

B.2.2.81 HardwareItemDescription/Errors/Error/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *HardwareItemDescription/Errors/Error/Description* child element shall provide a description of the error associated with the hardware item.

B.2.2.81.1 Attributes

HardwareItemDescription/Errors/Error/Description contains no attributes.

B.2.2.81.2 Child elements

HardwareItemDescription/Errors/Error/Description contains no child elements.

B.2.2.82 HardwareItemDescription/FactoryDefaults

Properties: isRef 0, content complex

The *HardwareItemDescription/FactoryDefaults* child element shall identify the default factory settings of the hardware item.

B.2.2.82.1 Attributes

HardwareItemDescription/FactoryDefaults contains no attributes.

B.2.2.82.2 Child elements

HardwareItemDescription/FactoryDefaults contains the following child element:

Name	Subclause	Type	Use
Default	B.2.2.83	*c:NamedValue*	1 … ∞

B.2.2.83 HardwareItemDescription/FactoryDefaults/Default

Base type: *c:NamedValue*

Properties: isRef 0, content complex

The *HardwareItemDescription/FactoryDefaults/Default* child element shall identify a default factory setting of the hardware item.

B.2.2.83.1 Attributes

HardwareItemDescription/FactoryDefaults/Default inherits the attribute of *c:NamedValue* (*name*).

B.2.2.83.2 Child elements

HardwareItemDescription/FactoryDefaults/Default inherits the child elements of *c:NamedValue* (*Collection*, *Datum*, and *IndexedArray*).

B.2.2.84 HardwareItemDescription/Interface

Base type: *c:PhysicalInterface*

Properties: isRef 0, content complex

The *HardwareItemDescription/Interface* child element shall identify the electrical interfaces to the hardware item.

B.2.2.84.1 Attributes

HardwareItemDescription/Interface contains no attributes.

B.2.2.84.2 Child elements

HardwareItemDescription/Interface inherits the child elements of *c:PhysicalInterface* (*Connectors* and *Ports*).

B.2.2.85 HardwareItemDescription/LegalDocuments

Properties: isRef 0, content complex

The *HardwareItemDescription/LegalDocuments* child element shall be a collector element of legal documents for the subject hardware item to be assembled.

B.2.2.85.1 Attributes

HardwareItemDescription/LegalDocuments contains no attributes.

B.2.2.85.2 Child elements

HardwareItemDescription/LegalDocuments contains one of the following child elements:

	Name	Subclause	Type	Use
Choice	Conformance	B.2.2.86	*c:Document*	1 … ∞
	Exportability	B.2.2.87	*c:Document*	
	License	B.2.2.88	*c:Document*	
	Safety	B.2.2.89	*c:Document*	

	Warranty	B.2.2.90	*c:Document*	
NOTE—Choice indicates that only one of these elements may be specified.				

B.2.2.86 HardwareItemDescription/LegalDocuments/Conformance

Base type: *c:Document*

Properties: isRef 0, content complex

The *HardwareItemDescription/LegalDocuments/Conformance* child element shall identify conformance documentation for the subject hardware item.

B.2.2.86.1 Attributes

HardwareItemDescription/LegalDocuments/Conformance inherits the attributes of *c:Document* (*name* and *uuid*).

B.2.2.86.2 Child elements

HardwareItemDescription/LegalDocuments/Conformance inherits the child elements of *c:Document* (*Extension*, *Text*, and *URL*).

B.2.2.87 HardwareItemDescription/LegalDocuments/Exportability

Base type: *c:Document*

Properties: isRef 0, content complex

The *HardwareItemDescription/LegalDocuments/Exportability* child element shall identify exportability documentation for the subject hardware item.

B.2.2.87.1 Attributes

HardwareItemDescription/LegalDocuments/Exportability inherits the attributes of *c:Document* (*name* and *uuid*).

B.2.2.87.2 Child elements

HardwareItemDescription/LegalDocuments/Exportability inherits the child elements of *c:Document* (*Extension*, *Text*, and *URL*).

B.2.2.88 HardwareItemDescription/LegalDocuments/License

Base type: *c:Document*

Properties: isRef 0, content complex

The *HardwareItemDescription/LegalDocuments/License* child element shall identify licensing documentation for the subject hardware item.

B.2.2.88.1 Attributes

HardwareItemDescription/LegalDocuments/License inherits the attributes of *c:Document* (*name* and *uuid*).

B.2.2.88.2 Child elements

HardwareItemDescription/LegalDocuments/License inherits the child elements of *c:Document* (*Extension*, *Text*, and *URL*).

B.2.2.89 HardwareItemDescription/LegalDocuments/Safety

Base type: *c:Document*

Properties: isRef 0, content complex

The *HardwareItemDescription/LegalDocuments/Safety* child element shall identify safety documentation for the subject hardware item.

B.2.2.89.1 Attributes

HardwareItemDescription/LegalDocuments/Safety inherits the attributes of *c:Document* (*name* and *uuid*).

B.2.2.89.2 Child elements

HardwareItemDescription/LegalDocuments/Safety inherits the child elements of *c:Document* (*Extension*, *Text*, and *URL*).

B.2.2.90 HardwareItemDescription/LegalDocuments/Warranty

Base type: *c:Document*

Properties: isRef 0, content complex

The *HardwareItemDescription/LegalDocuments/Warranty* child element shall identify warranty documentation for the subject hardware item.

B.2.2.90.1 Attributes

HardwareItemDescription/LegalDocuments/Warranty inherits the attributes of *c:Document* (*name* and *uuid*).

B.2.2.90.2 Child elements

HardwareItemDescription/LegalDocuments/Warranty inherits the child elements of *c:Document* (*Extension*, *Text*, and *URL*).

B.2.2.91 HardwareItemDescription/NetworkList

Properties: isRef 0, content complex

The *HardwareItemDescription/NetworkList* child element shall identify how the each port on the hardware item is connected.

B.2.2.91.1 Attributes

HardwareItemDescription/NetworkList contains no attributes.

B.2.2.91.2 Child elements

HardwareItemDescription/NetworkList contains the following child element:

Name	Subclause	Type	Use
Network	B.2.2.92	*hc:Network*	1 … ∞

B.2.2.92 HardwareItemDescription/NetworkList/Network

Base type: *hc:Network*

Properties: isRef 0, content complex

The *HardwareItemDescription/NetworkList/Network* child element shall identify the port connection.

B.2.2.92.1 Attributes

HardwareItemDescription/NetworkList/Network inherits the attributes of *hc:Network* (*baseIndex*, *count*, *incrementBy*, and *replacementCharacter*)

B.2.2.92.2 Child elements

HardwareItemDescription/NetworkList/Network inherits the child elements of *hc:Network* (*Description*, *Extension*, and *Node*).

B.2.2.93 HardwareItemDescription/OperationalRequirements

Base type: *hc:OperationalRequirements*

Properties: isRef 0, content complex

The *HardwareItemDescription/OperationalRequirements* child element shall identify the operational requirements of the hardware item.

B.2.2.93.1 Attributes

HardwareItemDescription/OperationalRequirements inherits the attributes of *hc:OperationalRequirements* (*name* and *uuid*).

B.2.2.93.2 Child elements

HardwareItemDescription/OperationalRequirements inherits the child element of *hc:OperationalRequirements* (*OperationalRequirement*).

B.2.2.94 HardwareItemDescription/ParentComponents

Properties: isRef 0, content complex

The *HardwareItemDescription/ParentComponents* child element shall be a collector element of the identification of the next-higher assembly of the subject hardware item.

B.2.2.94.1 Attributes

HardwareItemDescription/ParentComponents contains no attributes.

B.2.2.94.2 Child elements

HardwareItemDescription/ParentComponents contains the following child element:

Name	Subclause	Type	Use
Component	B.2.2.95	*c:ItemDescriptionReference*	1 … ∞

B.2.2.95 HardwareItemDescription/ParentComponents/Component

Base type: Extension of *c:ItemDescriptionReference*

Properties: isRef 0, content complex

The *HardwareItemDescription/ParentComponents/Component* child element shall identify the next-higher assembly of the subject hardware item.

B.2.2.95.1 Attributes

HardwareItemDescription/ParentComponents/Component contains the following attribute:

Name	Type	Description	Use
ID	*c:NonBlankString*	A user-defined string uniquely identifying the next-higher assembly. Example: Acme Tank.	Required

B.2.2.95.2 Child elements

HardwareItemDescription/ParentComponents/Component inherits the child elements of *c:ItemDescriptionReference* (*Definition* and *DescriptionDocumentReference*).

B.2.2.96 HardwareItemDescription/PhysicalCharacteristics

Base type: *hc:PhysicalCharacteristics*

Properties: isRef 0, content complex

The *HardwareItemDescription/PhysicalCharacteristics* child element shall be a collector element of the identification of the mass, volume, and measurements for the subject hardware item.

B.2.2.96.1 Attributes

HardwareItemDescription/PhysicalCharacteristics contains no attributes.

B.2.2.96.2 Child elements

HardwareItemDescription/PhysicalCharacteristics inherits the child elements of *hc:PhysicalCharacteristics* (*LinearMeasurements*, *Mass*, *Other*, and *Volume*).

B.2.2.97 HardwareItemDescription/PowerRequirements

Base type: *hc:PowerSpecifications*

Properties: isRef 0, content complex

The *HardwareItemDescription/PowerRequirements* child element shall be a collector element of the identification of ac or dc power requirements for the subject hardware item.

B.2.2.97.1 Attributes

HardwareItemDescription/PowerRequirements contains no attributes.

B.2.2.97.2 Child elements

HardwareItemDescription/PowerRequirements inherits the child elements of *hc:PowerSpecifications* (*DC* and *AC*).

B.2.2.98 Item

The *Item* complex type shall be the base type for hardware entities.

B.2.2.98.1 Attributes

Item contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the hardware item.	Required

B.2.2.98.2 Child elements

Item contains the following child elements:

Name	Subclause	Type	Use
Description	B.2.2.99	*c:NonBlankString*	Optional
Extension	B.2.2.100	*c:Extension*	Optional

B.2.2.99 Item/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Item/Description* child element shall be a description of the hardware item.

B.2.2.99.1 Attributes

Item/Description contains no attributes.

B.2.2.99.2 Child elements

Item/Description contains no child elements.

B.2.2.100 Item/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *Item/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.2.2.100.1 Attributes

Item/Extension contains no attributes.

B.2.2.100.2 Child elements

Item/Extension inherits the child element of *c:Extension* (*##other*).

B.2.2.101 IVI

Base type: Extension of *hc:Driver*

Properties: base *hc:Driver*, abstract true

The *IVI* complex type shall be the base type for XML schema elements intended to document properties of an interchangeable virtual instrumentation (IVI)[12] driver.

B.2.2.101.1 Attributes

IVI contains no attributes.

B.2.2.101.2 Child elements

IVI contains the following child elements, in addition to those inherited from *hc:Driver* (*Bit16*, *Bit32*, *Bit64*, and *Unified*):

Name	Subclause	Type	Use
Class	B.2.2.102	*c:NonBlankString*	0 … ∞
ComplianceDocument	B.2.2.103	*c:Document*	Optional

B.2.2.102 IVI/Class

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *IVI/Class* child element shall identify the IVI class (or classes) provided by the IVI driver.

B.2.2.102.1 Attributes

IVI/Class contains no attributes.

B.2.2.102.2 Child elements

IVI/Class contains no child elements.

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

B.2.2.103 IVI/ComplianceDocument

Base type: *c:Document*

Properties: isRef 0, content complex

The *IVI/ComplianceDocument* child element shall identify the IVI compliance document.

B.2.2.103.1 Attributes

IVI/ComplianceDocument inherits the attributes of *c:Document* (*name* and *uuid*).

B.2.2.103.2 Child elements

IVI/ComplianceDocument inherits the child elements of *c:Document* (*Extension*, *Text*, and *URL*).

B.2.2.104 IVI-C

Base type: Extension of *hc:IVI*

Properties: base *hc:IVI*

The *IVI-C* complex type shall be the base type for XML schema elements intended to document properties of an IVI C language (IVI-C) driver.

B.2.2.104.1 Attributes

IVI-C contains the following attribute:

Name	Type	Description	Use
prefix	*c:NonBlankString*	A descriptive or common name for the prefix to be used for all application programming interface (API) functions in the IVI-C driver.	Required

B.2.2.104.2 Child elements

IVI-C inherits the child elements of *hc:IVI* (*Bit16*, *Bit32*, *Bit64*, *Class*, *ComplianceDocument*, and *Unified*).

B.2.2.105 IVI-COM

Base type: Extension of *hc:IVI*

Properties: base *hc:IVI*

The *IVI-COM* complex type shall be the base type for XML schema elements intended to document properties of an IVI component object module (IVI-COM) driver.

B.2.2.105.1 Attributes

IVI-COM contains the following attribute:

Name	Type	Description	Use
progID	*c:NonBlankString*	A user-defined string uniquely identifying the driver class.	Required

B.2.2.105.2 Child elements

IVI-COM inherits the child elements of *hc:IVI* (*Bit16*, *Bit32*, *Bit64*, *Class*, *ComplianceDocument*, and *Unified*).

B.2.2.106 IVI.NET

Base type: Extension of *hc:IVI*

Properties: base *hc:IVI*

The *IVI.NET* complex type shall be the base type for XML schema elements intended to document properties of an IVI.NET driver.

B.2.2.106.1 Attributes

IVI.NET contains the following attribute:

Name	Type	Description	Use
assemblyQualifiedClassName	*c:NonBlankString*	A instantiation of the driver class.	Required

B.2.2.106.2 Child elements

IVI.COM inherits the child elements of *hc:IVI* (*Bit16*, *Bit32*, *Bit64*, *Class*, *ComplianceDocument*, and *Unified*).

B.2.2.107 LANTriggerPropertyGroup

Base type: Extension of *hc:TriggerPropertyGroup*

Properties: base *hc:TriggerPropertyGroup*

The *LANTriggerPropertyGroup* complex type shall be the base type for XML schema elements intended to document properties of a local area network (LAN) trigger.

B.2.2.107.1 Attributes

LANTriggerPropertyGroup inherits the attribute of *hc:TriggerPropertyGroup* (*name*).

B.2.2.107.2 Child elements

LANTriggerPropertyGroup inherits the child elements of *hc:TriggerPropertyGroup* (*Description* and *Extension*).

B.2.2.108 LevelType

The *LevelType* complex type shall be the base type for XML schema elements intended to document properties of an analog voltage in order for a trigger to occur.

B.2.2.108.1 Attributes

LevelType contains the following attributes:

Name	Type	Description	Use
numberOfBits	xs:int	The resolution of the trigger signal amplitude reading. An integer number shall be specified.	Required
units	*hc:LevelUnits*	The units associated with the trigger signal amplitude. Either %FullScale or +/-V shall be specified.	Required
value	xs:double	The amplitude of the trigger signal.	Required

B.2.2.108.2 Child elements

LevelType contains no child elements.

B.2.2.109 Mapping

The *Mapping* complex type shall be the base type for XML schema elements intended to document the mapping of capabilities to ports of the hardware item.

B.2.2.109.1 Attributes

Mapping inherits the attributes of the *c:RepeatedItemAttributes* attribute group (*baseIndex*, *count*, *incrementedBy*, and *replacementCharacter*).

B.2.2.109.2 Child elements

Mapping contains the following child element:

Name	Subclause	Type	Use
Map	B.2.2.110	*hc:Network*	1 … ∞

B.2.2.110 Mapping/Map

Base type: *hc:Network*

Properties: isRef 0, content complex

The *Mapping/Map* child element shall identify a specific capability to a specific port.

B.2.2.110.1 Attributes

Mapping/Map inherits the attributes of *hc:Network* (*baseIndex*, *count*, *incrementedBy*, and *replacementCharacter*).

B.2.2.110.2 Child elements

Mapping/Map inherits the child elements of *hc:Network* (*Description*, *Extension*, and *Node*).

B.2.2.111 MatrixPort

Base type: Extension of *hc:RepeatedItem*

Properties: base *hc:RepeatedItem*

The *MatrixPort* complex type shall be the base type for XML schema elements intended to document properties of matrix switch port(s).

B.2.2.111.1 Attributes

MatrixPort inherits the attributes of *hc:RepeatedItem* (*baseIndex*, *count*, *incrementedBy*, *name*, and *replacementCharacter*).

B.2.2.111.2 Child elements

MatrixPort inherits the child elements of *hc:RepeatedItem* (*Description*, and *Extension*).

B.2.2.112 MatrixSwitch

Base type: Extension of *hc:Item*

Properties: base *hc:Item*

The *MatrixSwitch* complex type shall be the base type for XML schema elements intended to document the name of a matrix switch.

B.2.2.112.1 Attributes

MatrixSwitch inherits the attribute of *hc:Item* (*name*).

B.2.2.112.2 Child elements

MatrixSwitch contains the following child elements, in addition to those inherited from *hc:Item* (*Description*, and *Extension*):

Name	Subclause	Type	Use
Columns	B.2.2.113	—	Required
Rows	B.2.2.115	—	Required

B.2.2.113 MatrixSwitch/Columns

Properties: isRef 0, content complex

The *MatrixSwitch/Columns* child element shall document properties of the columns of a matrix switch.

B.2.2.113.1 Attributes

MatrixSwitch/Columns contains no attributes.

B.2.2.113.2 Child elements

MatrixSwitch/Columns contains the following child element:

Name	Subclause	Type	Use
Pin	B.2.2.114	*hc:MatrixPort*	1 … ∞

B.2.2.114 MatrixSwitch/Columns/Pin

Base type: *hc:MatrixPort*

Properties: isRef 0, content complex

The *MatrixSwitch/Columns/Pin* child element shall document properties of the pin in the columns of a matrix switch.

B.2.2.114.1 Attributes

MatrixSwitch/Columns/Pin inherits the attributes of *hc:MatrixPort* (*baseIndex*, *count*, *incrementedBy*, *name*, and *replacementCharacter*).

B.2.2.114.2 Child elements

MatrixSwitch/Columns/Pin inherits the child elements of *hc:MatrixPort* (*Description* and *Extension*).

B.2.2.115 MatrixSwitch/Rows

Properties: isRef 0, content complex

The *MatrixSwitch/Rows* child element shall document properties of the rows of a matrix switch.

B.2.2.115.1 Attributes

MatrixSwitch/Rows contains no attributes.

B.2.2.115.2 Child elements

MatrixSwitch/Rows contains the following child element:

Name	Subclause	Type	Use
Pin	B.2.2.116	*hc:MatrixPort*	1 … ∞

B.2.2.116 MatrixSwitch/Rows/Pin

Base type: *hc:MatrixPort*

Properties: isRef 0, content complex

The *MatrixSwitch/Rows/Pin* child element shall document properties of the pin in the rows of a matrix switch.

B.2.2.116.1 Attributes

MatrixSwitch/Rows/Pin inherits the attributes of *hc:MatrixPort* (*baseIndex*, *count*, *incrementedBy*, *name*, and *replacementCharacter*).

B.2.2.116.2 Child elements

MatrixSwitch/Rows/Pin inherits the child elements of *hc:MatrixPort* (*Description* and *Extension*).

B.2.2.117 MinPulseWidthType

The *MinPulseWidthType* complex type shall be the base type for XML schema elements intended to document the minimum pulse width of a digital-signal-based trigger.

B.2.2.117.1 Attributes

MinPulseWidthType contains the following attributes:

Name	Type	Description	Use
units	*hc:PulseUnits*	The dimension associated with the pulse width's value. Allowable dimensions shall be S, mS, uS, nS, pS, or fS.	Required
value	xs:double	The numeric value of the pulse width.	Required

B.2.2.117.2 Child elements

MinPulseWidthType contains no child elements.

B.2.2.118 Network

The *Network* complex type shall be the base type for XML schema elements intended to document properties of how various hardware entities are connected.

B.2.2.118.1 Attributes

Network inherits the attributes of the *c:RepeatedItemAttributes* attribute group (*baseIndex*, *count*, *incrementedBy*, and *replacementCharacter*).

B.2.2.118.2 Child elements

Network contains the following child elements:

Name	Subclause	Type	Use
Description	B.2.2.119	*c:NonBlankString*	Optional
Extension	B.2.2.120	*c:Extension*	Optional
Node	B.2.2.121	*hc:NetworkNode*	1 … ∞

B.2.2.119 Network/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Network/Description* child element shall provide a description of the network connection.

B.2.2.119.1 Attributes

Network/Description contains no attributes.

B.2.2.119.2 Child elements

Network/Description contains no child elements.

B.2.2.120 Network/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *Network/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.2.2.120.1 Attributes

Network/Extension contains no attributes.

B.2.2.120.2 Child elements

Network/Extension inherits the child element of *c:Extension* (*##other*).

B.2.2.121 Network/Node

Base type: *hc:NetworkNode*

Properties: isRef 0, content complex

The *Network/Node* child element shall identify the properties of the network node to which the hardware item is connected.

B.2.2.121.1 Attributes

Network/Node contains no attributes.

B.2.2.121.2 Child elements

Network/Node inherits the child elements of *hc:NetworkNode* (*Description*, *Extension*, and *Path*).

B.2.2.122 NetworkNode

The *NetworkNode* complex type shall be the base type for XML schema elements intended to document properties of the network node to which the hardware item is connected.

B.2.2.122.1 Attributes

NetworkNode contains no attributes.

B.2.2.122.2 Child elements

NetworkNode contains the following child elements:

Name	Subclause	Type	Use
Description	B.2.2.123	*c:NonBlankString*	Optional
Extension	B.2.2.124	*c:Extension*	Optional
Path	B.2.2.125	*c:NonBlankString*	Required

B.2.2.123 NetworkNode/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whitespace replace

The *NetworkNode/Description* child element shall describe the network node to which the hardware item is connected.

B.2.2.123.1 Attributes

NetworkNode/Description contains no attributes.

B.2.2.123.2 Child elements

NetworkNode/Description contains no child elements.

B.2.2.124 NetworkNode/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *NetworkNode/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.2.2.124.1 Attributes

NetworkNode/Extension contains no attributes.

B.2.2.124.2 Child elements

NetworkNode/Extension inherits the child element of *c:Extension* (*##other*).

B.2.2.125 NetworkNode/Path

Base type: Extension of *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whitespace replace

The *NetworkNode/Path* child element describes the XPath expression that shall evaluate to a single node. This single node is part of the path.

B.2.2.125.1 Attributes

NetworkNode/Path contains the following attributes:

Name	Type	Description	Use
documentId	*c:NonBlankString*	The UUID for the document referenced by the element.	Optional

B.2.2.125.2 Child elements

NetworkNode/Path contains no child elements.

B.2.2.126 Nominal

Base type: Extension of *hc:Specification*

Properties: base *hc:Specification*

The *Nominal* complex type shall describe specifications of the instrument that are true by design (however, not tested or measured).

B.2.2.126.1 Attributes

Nominal inherits the attribute of *hc:Specification* (*name*).

B.2.2.126.2 Child elements

Nominal inherits the child elements of *hc:Specification* (*Conditions*, *Definition*, *Description*, *ExclusiveOptions*, *Graph*, *Limits*, *RequiredOptions*, and *SupplementalInformation*).

B.2.2.127 OperationalRequirements

The *OperationalRequirements* complex type shall be the base type for XML schema elements intended to document the operational requirements that must be satisfied in order for proper operation of the hardware item.

B.2.2.127.1 Attributes

OperationalRequirements contains the following attributes:

Name	Type	Description	Use
warmUpTime	xs:duration	The warm-up time of the hardware item.	Required

B.2.2.127.2 Child elements

OperationalRequirements contains the following child element:

Name	Subclause	Type	Use
OperationalRequirement	B.2.2.128	*c:NamedValue*	1 … ∞

B.2.2.128 OperationalRequirements/OperationalRequirement

Base type: *c:NamedValue*

Properties: isRef 0, content complex

The *OperationalRequirements/OperationalRequirement* child element shall textually describe an operational requirement of the hardware item.

B.2.2.128.1 Attributes

OperationalRequirements/OperationalRequirement inherits the attribute of *c:NamedValue* (*name*).

B.2.2.128.2 Child elements

OperationalRequirements/OperationalRequirement inherits the child elements of *c:NamedValue* (*Collection*, *Datum*, and *IndexedArray*).

B.2.2.129 PhysicalCharacteristics

The *PhysicalCharacteristics* complex type shall be the base type for XML schema elements intended to document the physical characteristics of a hardware item.

B.2.2.129.1 Attributes

PhysicalCharacteristics contains no attributes.

B.2.2.129.2 Child elements

PhysicalCharacteristics contains the following child elements:

Name	Subclause	Type	Use
LinearMeasurements	B.2.2.130	—	Optional
Mass	B.2.2.135	*c:double*	Optional
Other	B.2.2.136	—	Optional
Volume	B.2.2.138	*c:double*	Optional

B.2.2.130 PhysicalCharacteristics/LinearMeasurements

Properties: isRef 0, content complex

The *PhysicalCharacteristics/LinearMeasurements* child element shall be a collector element of the identification of the lineal measurements of the subject hardware item.

B.2.2.130.1 Attributes

PhysicalCharacteristics/LinearMeasurements contains no attributes.

B.2.2.130.2 Child elements

PhysicalCharacteristics/LinearMeasurements contains the following child elements:

Name	Subclause	Type	Use
Depth	B.2.2.131	*c:double*	Optional
Height	B.2.2.132	*c:double*	Optional
RackUSize	B.2.2.133	—	Optional
Width	B.2.2.134	*c:double*	Optional

B.2.2.131 PhysicalCharacteristics/LinearMeasurements/Depth

Base type: *c:double*

Properties: isRef 0, content complex

The *PhysicalCharacteristics/LinearMeasurements/Depth* child element shall identify the lineal depth measurement of the subject hardware item.

B.2.2.131.1 Attributes

PhysicalCharacteristics/LinearMeasurements/Depth inherits the attributes of *c:double* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.2.2.131.2 Child elements

PhysicalCharacteristics/LinearMeasurements/Depth inherits the child elements of *c:double* (*Confidence, ErrorLimits*, *Range*, and *Resolution*).

B.2.2.132 PhysicalCharacteristics/LinearMeasurements/Height

Base type: *c:double*

Properties: isRef 0, content complex

The *PhysicalCharacteristics/LinearMeasurements/Height* child element shall identify the lineal height measurement of the subject hardware item.

B.2.2.132.1 Attributes

PhysicalCharacteristics/LinearMeasurements/Height inherits the attributes of *c:double* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.2.2.132.2 Child elements

PhysicalCharacteristics/LinearMeasurements/Height inherits the child elements of *c:double* (*Confidence, ErrorLimits*, *Range*, and *Resolution*).

B.2.2.133 PhysicalCharacteristics/LinearMeasurements/RackUSize

Properties: isRef 0, content complex

The *PhysicalCharacteristics/LinearMeasurements/RackUSize* child element shall identify the rack unit size of the subject hardware item.

B.2.2.133.1 Attributes

PhysicalCharacteristics/LinearMeasurements/RackUSize contains the following attribute:

Name	Type	Description	Use
value	xs:double	The dimensionless rack unit height of the hardware item. Example: 3. Note that 1 rack unit is 1.75 in (44.45 mm).	Required

B.2.2.133.2 Child elements

PhysicalCharacteristics/LinearMeasurements/RackUSize contains no child elements.

B.2.2.134 PhysicalCharacteristics/LinearMeasurements/Width

Base type: *c:double*

Properties: isRef 0, content complex

The *PhysicalCharacteristics/LinearMeasurements/Width* child element shall identify the lineal width measurement of the subject hardware item.

B.2.2.134.1 Attributes

PhysicalCharacteristics/LinearMeasurements/Width inherits the attributes of *c:double* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.2.2.134.2 Child elements

PhysicalCharacteristics/LinearMeasurements/Width inherits the child elements of *c:double* (*Confidence, ErrorLimits*, *Range*, and *Resolution*).

B.2.2.135 PhysicalCharacteristics/Mass

Base type: *c:double*

Properties: isRef 0, content complex

The *PhysicalCharacteristics/Mass* child element shall identify the mass of the subject hardware item.

B.2.2.135.1 Attributes

PhysicalCharacteristics/Mass inherits the attributes of *c:double* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.2.2.135.2 Child elements

PhysicalCharacteristics/Mass inherits the child elements of *c:double* (*Confidence, ErrorLimits*, *Range*, and *Resolution*).

B.2.2.136 PhysicalCharacteristics/Other

Properties: isRef 0, content complex

The *PhysicalCharacteristics/Other* child element shall identify other physical characteristics of the subject hardware item not delineated as a child element of *PhysicalCharacteristics*.

B.2.2.136.1 Attributes

PhysicalCharacteristics/Other contains no attributes.

B.2.2.136.2 Child elements

PhysicalCharacteristics/Other contains the following child element:

Name	Subclause	Type	Use
Value	B.2.2.137	*c:NamedValue*	1 … ∞

B.2.2.137 PhysicalCharacteristics/Other/Value

Base type: *c:NamedValue*

Properties: isRef 0, content complex

The *PhysicalCharacteristics/Other/Value* child element shall identify any other physical characteristics of the subject hardware item not specifically contained within the *PhysicalCharacteristics* complex type.

B.2.2.137.1 Attributes

PhysicalCharacteristics/Other/Value inherits the attribute of *c:NamedValue* (*name*).

B.2.2.137.2 Child elements

PhysicalCharacteristics/Other/Value inherits the child elements of *c:NamedValue* (*Collection*, *Datum*, and *IndexedArray*).

B.2.2.138 PhysicalCharacteristics/Volume

Base type: *c:double*

Properties: isRef 0, content complex

The *PhysicalCharacteristics/Volume* child element shall identify the physical volume of the subject hardware item.

B.2.2.138.1 Attributes

PhysicalCharacteristics/Volume inherits the attributes of *c:double* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.2.2.138.2 Child elements

PhysicalCharacteristics/Volume inherits the child elements of *c:double* (*Confidence, ErrorLimits*, *Range*, and *Resolution*).

B.2.2.139 PowerSpecifications

The *PowerSpecifications* complex type shall be the base type for XML schema elements intended to document the input power requirements of a hardware item.

B.2.2.139.1 Attributes

PowerSpecifications contains no attributes.

B.2.2.139.2 Child elements

PowerSpecifications contains one of the following child elements:

	Name	Subclause	Type	Use
Choice	AC	B.2.2.140	—	1 … ∞
	DC	B.2.2.148	—	
NOTE—Choice indicates that only one of these elements may be specified.				

B.2.2.140 PowerSpecifications/AC

Properties: isRef 0, content complex

The *PowerSpecifications/AC* child element shall be a collector element of the identification of ac power characteristics for the subject hardware item.

B.2.2.140.1 Attributes

PowerSpecifications/AC contains the following attribute:

Name	Type	Description	Use
phase	xs:double	The dimensionless number of phases to the ac power form. The default shall be 1. Example: 3 (indicating a three-phase ac requirement that is either a delta or a wye configuration).	Optional

B.2.2.140.2 Child elements

PowerSpecifications/AC contains the following child elements:

	Name	Subclause	Type	Use
Choice	Amperage	B.2.2.141	*c:Limit*	Required
	PowerDraw	B.2.2.146	*c:Limit*	
	ConnectorPins	B.2.2.142	—	Optional
	Description	B.2.2.144	*c:NonBlankString*	Optional
	Frequency	B.2.2.145	*c:Limit*	Required
	Voltage	B.2.2.147	*c:Limit*	Required
NOTE—Choice indicates that only one of these elements may be specified.				

B.2.2.141 PowerSpecifications/AC/Amperage

Base type: *c:Limit*

Properties: isRef 0, content complex

The *PowerSpecifications/AC/Amperage* child element shall identify the amperage of the identified phase.

B.2.2.141.1 Attributes

PowerSpecifications/AC/Amperage inherits the attributes of *c:Limit* (*name* and *operator*).

B.2.2.141.2 Child elements

PowerSpecifications/AC/Amperage inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.2.2.142 PowerSpecifications/AC/ConnectorPins

Properties: isRef 0, content complex

The *PowerSpecifications/AC/ConnectorPins* child element shall identify the ac power connector pins.

B.2.2.142.1 Attributes

PowerSpecifications/AC/ConnectorPins contains no attributes.

B.2.2.142.2 Child elements

PowerSpecifications/AC/ConnectorPins contains the following child element:

Name	Subclause	Type	Use
ConnectorPin	B.2.2.143	*c:ConnectorLocation*	1 … ∞

B.2.2.143 PowerSpecifications/AC/ConnectorPins/ConnectorPin

Properties: isRef 0, content complex

The *PowerSpecifications/AC/ConnectorPins/ConnectorPin* child element shall identify a particular ac power connector pin.

B.2.2.143.1 Attributes

PowerSpecifications/AC/ConnectorPins/ConnectorPin contains the following attributes:

Name	Type	Description	Use
connectorID	c:NonBlankString	A user-defined string uniquely identifying the connector.	Required
pinID	c:NonBlankString	A user-defined string uniquely identifying the pin within the connector.	Optional

B.2.2.143.2 Child elements

PowerSpecifications/AC/ConnectorPins/ConnectorPin contains no child elements.

B.2.2.144 PowerSpecifications/AC/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *PowerSpecifications/AC/Description* child element shall describe the ac power. This description may include such items as three-phase configurations (delta or wye), electromagnetic interference and electromagnetic compatibility (EMI/EMC) characteristics, etc.

B.2.2.144.1 Attributes

PowerSpecifications/AC/Description contains no attributes.

B.2.2.144.2 Child elements

PowerSpecifications/AC/Description contains no child elements.

B.2.2.145 PowerSpecifications/AC/Frequency

Base type: *c:Limit*

Properties: isRef 0, content complex

The *PowerSpecifications/AC/Frequency* child element shall identify the frequency of the identified phase.

B.2.2.145.1 Attributes

PowerSpecifications/AC/Frequency inherits the attributes of *c:Limit* (*name* and *operator*).

B.2.2.145.2 Child elements

PowerSpecifications/AC/Frequency inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.2.2.146 PowerSpecifications/AC/PowerDraw

Base type: *c:Limit*

Properties: isRef 0, content complex

The *PowerSpecifications/AC/PowerDraw* child element shall indicate the amount of current (in amperes), as upper and lower limits, demanded of a supply circuit by the parent entity inheriting this data type.

B.2.2.146.1 Attributes

PowerSpecifications/AC/PowerDraw inherits the attributes of *c:Limit* (*name* and *operator*).

B.2.2.146.2 Child elements

PowerSpecifications/AC/PowerDraw inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.2.2.147 PowerSpecifications/AC/Voltage

Base type: *c:Limit*

Properties: isRef 0, content complex

The *PowerSpecifications/AC/Voltage* child element shall identify the voltage of the identified phase.

B.2.2.147.1 Attributes

PowerSpecifications/AC/Voltage inherits the attributes of *c:Limit* (*name* and *operator*).

B.2.2.147.2 Child elements

PowerSpecifications/AC/Voltage inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.2.2.148 PowerSpecifications/DC

Properties: isRef 0, content complex

The *PowerSpecifications/DC* child element shall be a collector of a hardware item's description and permits the identification of dc power characteristics for the subject hardware item.

B.2.2.148.1 Attributes

PowerSpecifications/DC contains the following attributes:

Name	Type	Description	Use
polarity	xs:double	An indication of the polarity of the dc voltage with respect to ground. Examples: positive and negative.	Optional
ripple	xs:double	The ac component of the dc voltage.	Optional

B.2.2.148.2 Child elements

PowerSpecifications/DC contains the following child elements:

	Name	Subclause	Type	Use
Choice	Amperage	B.2.2.149	*c:Limit*	Required
	PowerDraw	B.2.2.153	*c:Limit*	
	ConnectorPins	B.2.2.150	—	Optional
	Description	B.2.2.152	*c:NonBlankString*	Optional
	Voltage	B.2.2.154	*c:Limit*	Required
NOTE—Choice indicates that only one of these elements may be specified.				

B.2.2.149 PowerSpecifications/DC/Amperage

Base type: *c:Limit*

Properties: isRef 0, content complex

The *PowerSpecifications/DC/Amperage* child element shall identify the amperage of the dc power.

B.2.2.149.1 Attributes

PowerSpecifications/DC/Amperage inherits the attributes of *c:Limit* (*name* and *operator*).

B.2.2.149.2 Child elements

PowerSpecifications/DC/Amperage inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.2.2.150 PowerSpecifications/DC/ConnectorPins

Properties: isRef 0, content complex

The *PowerSpecifications/DC/ConnectorPins/ConnectorPin* child element shall identify the dc power connector pins.

B.2.2.150.1 Attributes

PowerSpecifications/DC/ConnectorPins contains no attributes.

B.2.2.150.2 Child elements

PowerSpecifications/DC/ConnectorPins contains the following child element:

Name	Subclause	Type	Use
ConnectorPin	B.2.2.151	*c:ConnectorLocation*	1 … ∞

B.2.2.151 PowerSpecifications/DC/ConnectorPins/ConnectorPin

Properties: isRef 0, content complex

The *PowerSpecifications/DC/ConnectorPins/ConnectorPin* child element shall identify a particular dc power connector pin.

B.2.2.151.1 Attributes

PowerSpecifications/DC/ConnectorPins/ConnectorPin contains the following attributes:

Name	Type	Description	Use
connectorID	c:NonBlankString	A user-defined string uniquely identifying the connector.	Required
pinID	c:NonBlankString	A user-defined string uniquely identifying the pin within the connector.	Optional

B.2.2.151.2 Child elements

PowerSpecifications/DC/ConnectorPins/ConnectorPin contains no child elements.

B.2.2.152 PowerSpecifications/DC/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *PowerSpecifications/DC/Description* child element shall describe the dc power.

B.2.2.152.1 Attributes

PowerSpecifications/DC/Description contains no attributes.

B.2.2.152.2 Child elements

PowerSpecifications/DC/Description contains no child elements.

B.2.2.153 PowerSpecifications/DC/PowerDraw

Base type: *c:Limit*

Properties: isRef 0, content complex

The *PowerSpecifications/DC/PowerDraw* child element shall indicate the amount of current (in amperes), as upper and lower limits, demanded of a supply circuit by the parent entity inheriting this data type.

B.2.2.153.1 Attributes

PowerSpecifications/DC/PowerDraw inherits the attributes of *c:Limit* (*name* and *operator*).

B.2.2.153.2 Child elements

PowerSpecifications/DC/PowerDraw inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.2.2.154 PowerSpecifications/DC/Voltage

Base type: *c:Limit*

Properties: isRef 0, content complex

The *PowerSpecifications/DC/Voltage* child element shall identify the voltage of the dc power.

B.2.2.154.1 Attributes

PowerSpecifications/DC/Voltage inherits the attributes of *c:Limit* (*name* and *operator*).

B.2.2.154.2 Child elements

PowerSpecifications/DC/Voltage inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.2.2.155 Register

Base type: Extension of *hc:ControlLanguage*

Properties: base *hc:ControlLanguage*

The *Register* complex type shall be the base type for XML schema child elements intended to identify the document that contains the instrument's register commands.

B.2.2.155.1 Attributes

Register contains no attributes.

B.2.2.155.2 Child elements

Register inherits the child element of *hc:ControlLanguage* (*Documentation*).

B.2.2.156 RepeatedItem

Base type: Extension of *hc:Item*

Properties: base *hc:Item*

The *RepeatedItem* complex type shall be the base type for XML schema elements intended to document multiple identical items with a single element within an instance document.

B.2.2.156.1 Attributes

RepeatedItem inherits the attributes from hc:Item (*name*) as well as those from the *c:RepeatedItemAttributes* attribute group (*baseIndex*, *count*, *incrementedBy*, and *replacementCharacter*):

B.2.2.156.2 Child elements

RepeatedItem inherits the child elements of *hc:Item* (*Description* and *Extension*).

B.2.2.157 Resource

Base type: Extension of *hc:RepeatedItem*

Properties: base *hc:RepeatedItem*

The *Resource* complex type shall be the base type for XML schema elements intended to document a resource and define its interface(s).

B.2.2.157.1 Attributes

Resource contains the following attribute, in addition to those inherited from *hc:RepeatedItem* (*baseIndex*, *count*, *incrementedBy*, *name*, and *replacementCharacter*):

Name	Type	Description	Use
index	xs:init	The index of the element within an *hc:RepeatedItem* array.	Optional

B.2.2.157.2 Child elements

Resource contains the following child elements, in addition to those inherited from *hc:RepeatedItem* (*Description* and *Extension*):

Name	Subclause	Type	Use
Interface	B.2.2.158	*c:Interface*	Required
Triggers	B.2.2.159	*hc:Triggers*	Optional

B.2.2.158 Resource/Interface

Base type: *c:Interface*

Properties: isRef 0, content complex

The *Resource/Interface* child element shall identify the electrical interface(s) to the hardware item.

B.2.2.158.1 Attributes

Resource/Interface contains no attributes.

B.2.2.158.2 Child elements

Resource/Interface inherits the child element of *c:Interface* (*Ports*).

B.2.2.159 Resource/Triggers

Base type: *hc:Triggers*

Properties: isRef 0, content complex

The *Resource/Triggers* child element shall identify the triggering associated with the hardware item.

B.2.2.159.1 Attributes

Resource/Triggers contains no attributes.

B.2.2.159.2 Child elements

Resource/Triggers inherits the child element of *hc:Triggers* (*Trigger*).

B.2.2.160 Resources

The *Resources* complex type shall be the base type for XML schema elements intended to document resources and define their interfaces.

B.2.2.160.1 Attributes

Resources contains no attributes.

B.2.2.160.2 Child elements

Resources contains the following child element:

Name	Subclause	Type	Use
Resource	B.2.2.161	*hc:Resource*	1 … ∞

B.2.2.161 Resources/Resource

Base type: *hc:Resource*

Properties: isRef 0, content complex

The *Resources/Resource* child element shall identify a resource and to define the resources interface.

B.2.2.161.1 Attributes

Resources/Resource inherits the attributes of *hc:Resource* (*baseIndex*, *count*, *incrementedBy*, *index*, *name*, and *replacementCharacter*):

B.2.2.161.2 Child elements

Resources/Resource inherits the child elements of *hc:Resource* (*Description*, *Extension*, *Interface*, and *Triggers*).

B.2.2.162 SCPI

Base type: Extension of *hc:ControlLanguage*

Properties: base *hc:ControlLanguage*

The *SCPI* complex type shall be the base type for XML schema elements intended to identify the document that contains the instrument's SCPI commands.

B.2.2.162.1 Attributes

SCPI contains no attributes.

B.2.2.162.2 Child elements

SCPI inherits the child element of *hc:ControlLanguage* (*Documentation*).

B.2.2.163 SoftwareTriggerPropertyGroup

Base type: Extension of *hc:TriggerPropertyGroup*

Properties: base *hc:TriggerPropertyGroup*

The *SoftwareTriggerPropertyGroup* complex type shall be the base type for XML schema elements intended to document properties of a trigger initiated by software.

B.2.2.163.1 Attributes

SoftwareTriggerPropertyGroup inherits the attribute of *hc:TriggerPropertyGroup* (*name*).

B.2.2.163.2 Child elements

SoftwareTriggerPropertyGroup inherits the child elements of *hc:TriggerPropertyGroup* (*Description* and *Extension*).

B.2.2.164 Specification

Properties: abstract true

The *Specification* complex type shall be the base type for XML schema elements intended to document each of the actual specifications used to develop the instruments: *hc:Characteristic*, *hc:Feature*, *hc:Guaranteed*, *hc:Nominal*, *hc:Typical*, or *hc:Specifications* collections.

B.2.2.164.1 Attributes

Specification contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the specification. Example: Acme ABCD DMM Product Specifications.	Required

B.2.2.164.2 Child elements

Specification contains the following child elements:

Name	Subclause	Type	Use
Conditions	B.2.2.165	*hc:SpecificationConditions*	Optional
Definition	B.2.2.166	—	Optional
Description	B.2.2.169	*c:NonBlankString*	Required
ExclusiveOptions	B.2.2.170	—	Optional
Graph	B.2.2.172	—	Optional
Limits	B.2.2.175	—	Optional
RequiredOptions	B.2.2.176	—	Optional
SupplementalInformation	B.2.2.179	*c:NonBlankString*	0 …∞

B.2.2.165 Specification/Conditions

Base type: *hc:SpecificationConditions*

Properties: isRef 0, content complex

The *Specification/Conditions* child element shall identify the conditions under which the specification is measured.

B.2.2.165.1 Attributes

Specification/Conditions contains no attributes.

B.2.2.165.2 Child elements

Specification/Conditions inherits the child element of *hc:SpecificationConditions* (*Condition*).

B.2.2.166 Specification/Definition

Properties: isRef 0, content complex

The *Specification/Definition* child element shall provide the mathematical description of how the specification is defined and verified, or it shall identify the document where the definition can be found.

B.2.2.166.1 Attributes

Specification/Definition contains no attributes.

B.2.2.166.2 Child elements

Specification/Definition contains one of the following child elements:

	Name	Subclause	Type	Use
Choice	Document	B.2.2.167	*c:Document*	Required
	Text	B.2.2.168	*c:NonBlankString*	
NOTE—Choice indicates that only one of these elements may be specified.				

B.2.2.167 Specification/Definition/Document

Base type: *c:Document*

Properties: isRef 0, content complex

The *Specification/Definition/Document* child element shall identify the document where the specification definitions can be located.

B.2.2.167.1 Attributes

Specification/Definition/Document inherits the attributes of *c:Document* (*name* and *uuid*).

B.2.2.167.2 Child elements

Specification/Definition/Document inherits the child elements of *c:Document* (*Extension*, *Text*, and *URL*).

B.2.2.168 Specification/Definition/Text

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Specification/Definition/Text* child element shall provide a description of the specification and provide a description of how the specification is verified.

B.2.2.168.1 Attributes

Specification/Definition/Text contains no attributes.

B.2.2.168.2 Child elements

Specification/Definition/Text contains no child elements.

B.2.2.169 Specification/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Specification/Description* child element shall provide a short description in English of the specification.

B.2.2.169.1 Attributes

Specification/Description contains no attributes.

B.2.2.169.2 Child elements

Specification/Description contains no child elements.

B.2.2.170 Specification/ExclusiveOptions

Properties: isRef 0, content complex

The *Specification/ExclusiveOptions* child element shall identify any instrumentation options that, if installed in the instrument, would invalidate the specification.

B.2.2.170.1 Attributes

Specification/ExclusiveOptions contains no attributes.

B.2.2.170.2 Child elements

Specification/ExclusiveOptions contains the following child element:

Name	Subclause	Type	Use
Option	B.2.2.171	*c:NonBlankString*	1 … ∞

B.2.2.171 Specification/ExclusiveOptions/Option

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Specification/ExclusiveOptions/Option* child element shall identify an instrument option that, if installed in the instrument, would invalidate the specification.

B.2.2.171.1 Attributes

Specification/ExclusiveOptions/Option contains no attributes.

B.2.2.171.2 Child elements

Specification/ExclusiveOptions/Option contains no child elements.

B.2.2.172 Specification/Graph

Properties: isRef 0, content complex

The *Specification/Graph* child element shall identify specification(s) that can be represented and conveyed to humans only graphically. This identification shall be either via extension or by specifying the URL where the graphical data can be located.

B.2.2.172.1 Attributes

Specification/Graph contains no attributes.

B.2.2.172.2 Child elements

Specification/Graph contains one of the following child elements:

	Name	Subclause	Type	Use
Choice	Extension	B.2.2.173	*c:Extension*	Required
	URL	B.2.2.174	*c:NonBlankURI*	
NOTE—Choice indicates that only one of these elements may be specified.				

B.2.2.173 Specification/Graph/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *Specification/Graph/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.2.2.173.1 Attributes

Specification/Graph/Extension contains no attributes.

B.2.2.173.2 Child elements

Specification/Graph/Extension inherits the child element of *c:Extension* (*##other*).

B.2.2.174 Specification/Graph/URL

Base type: *c:NonBlankURI*

Properties: isRef 0, content simple

Facets: minLength 1

The *Specification/Graph/URL* child element shall identify the URL where the graphical data can be located.

B.2.2.174.1 Attributes

Specification/Graph/URL contains no attributes.

B.2.2.174.2 Child elements

Specification/Graph/URL contains no child elements.

B.2.2.175 Specification/Limits

Properties: isRef 0, content complex

The *Specification/Limits* child element shall identify limits for the specification.

B.2.2.175.1 Attributes

Specification/Limits contains no attributes.

B.2.2.175.2 Child elements

Specification/Limits contains the following child element:

Name	Subclause	Type	Use
Limit	B.2.2.176	*c:Limit*	1 … ∞

B.2.2.176 Specification/Limits/Limit

Base type: *c:Limit*

Properties: isRef 0, content complex

The *Specification/Limits/Limit* child element shall identify the specification limit.

B.2.2.176.1 Attributes

Specification/Limits/Limit inherits the attributes of *c:Limit* (*name* and *operator*).

B.2.2.176.2 Child elements

Specification/Limits/Limit inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.2.2.177 Specification/RequiredOptions

Properties: isRef 0, content complex

The *Specification/RequiredOptions* child element shall identify any instrumentation options that are required to be installed in the instrument in order for the specification to be valid.

B.2.2.177.1 Attributes

Specification/RequiredOptions contains no attributes.

B.2.2.177.2 Child elements

Specification/RequiredOptions contains the following child element:

Name	Subclause	Type	Use
Option	B.2.2.178	*c:NonBlankString*	1 … ∞

B.2.2.178 Specification/RequiredOptions/Option

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Specification/RequiredOptions/Option* child element shall identify an installed instrument option.

B.2.2.178.1 Attributes

Specification/RequiredOptions/Option contains no attributes.

B.2.2.178.2 Child elements

Specification/RequiredOptions/Option contains no child elements.

B.2.2.179 Specification/SupplementalInformation

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Specification/SupplementalInformation* child element shall identify any additional information that may be required in order to clarify the specification (such as information typically found in instrumentation datasheet footnotes).

B.2.2.179.1 Attributes

Specification/SupplementalInformation contains no attributes.

B.2.2.179.2 Child elements

Specification/SupplementalInformation contains no child elements.

B.2.2.180 SpecificationConditions

The *SpecificationConditions* complex type shall identify the conditions under which the specification is valid.

B.2.2.180.1 Attributes

SpecificationConditions contains no attributes.

B.2.2.180.2 Child elements

SpecificationConditions contains the following child element:

Name	Subclause	Type	Use
Condition	B.2.2.181	*c:NonBlankString*	1 … ∞

B.2.2.181 SpecificationConditions/Condition

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *SpecificationConditions/Condition* child element shall identify a specification condition (e.g., the instrument specification shall be considered valid only after a 30 min warm-up period).

B.2.2.181.1 Attributes

SpecificationConditions/Condition contains no attributes.

B.2.2.181.2 Child elements

SpecificationConditions/Condition contains no child elements.

B.2.2.182 SpecificationGroup

The *SpecificationGroup* complex type shall define the groupings of specifications that share a common set of conditions.

B.2.2.182.1 Attributes

SpecificationGroup contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the specification group. Example: AC Characteristics.	Optional

B.2.2.182.2 Child elements

SpecificationGroup contains the following child elements:

	Name	Subclause	Type	Use
	Conditions	B.2.2.183	*hc:SpecificationConditions*	Optional
	Description	B.2.2.184	*c:NonBlankString*	Optional
Choice	Group	B.2.2.185	*hc:SpecificationGroup*	1 … ∞
	Specification	B.2.2.186	*hc:Specification*	
NOTE—Choice indicates that only one of these elements may be specified.				

B.2.2.183 SpecificationGroup/Conditions

Base type: *hc:SpecificationConditions*

Properties: isRef 0, content complex

The *SpecificationGroup/Conditions* child element shall identify the conditions under which the grouped specifications are measured.

B.2.2.183.1 Attributes

SpecificationGroup/Conditions contains no attributes.

B.2.2.183.2 Child elements

SpecificationGroup/Conditions inherits the child element of *hc:SpecificationConditions* (*Condition*).

B.2.2.184 SpecificationGroup/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *SpecificationGroup/Description* child element shall textually describe the specification group.

B.2.2.184.1 Attributes

SpecificationGroup/Description contains no attributes.

B.2.2.184.2 Child elements

SpecificationGroup/Description contains no child elements.

B.2.2.185 SpecificationGroup/Group

Base type: *hc:SpecificationGroup*

Properties: isRef 0, content complex

The *SpecificationGroup/Group* child element shall uniquely name a group of specifications that are sharing a common set of conditions.

B.2.2.185.1 Attributes

SpecificationGroup/Group inherits the attribute of *hc:SpecificationGroup* (*name*).

B.2.2.185.2 Child elements

SpecificationGroup/Group inherits the child elements of *hc:SpecificationGroup* (*Conditions*, *Description*, *Group*, and *Specification*).

B.2.2.186 SpecificationGroup/Specification

Base type: *hc:Specification*

Properties: isRef 0, content complex

The *SpecificationGroup/Specification* child element shall identify the *hc:Specification* (*name*).

B.2.2.186.1 Attributes

SpecificationGroup/Specification inherits the attribute of *hc:Specification* (*name*).

B.2.2.186.2 Child elements

SpecificationGroup/Specification inherits the child elements of *hc:Specification* (*Conditions*, *Definition*, *Description*, *ExclusiveOptions*, *Graph*, *Limits*, *RequiredOptions*, and *SupplementalInformation*).

B.2.2.187 Specifications

The *Specifications* complex type shall be the specification, and groupings of specifications, that share a common set of conditions. *Specifications* may be used to define specification traceability (e.g., the certification of the specification) and define the conditions under which the specification is measured.

B.2.2.187.1 Attributes

Specifications contains no attributes.

B.2.2.187.2 Child elements

Specifications contains the following child elements:

	Name	Subclause	Type	Use
	Certifications	B.2.2.188	—	Optional
	Conditions	B.2.2.190	*hc:SpecificationConditions*	Optional
Choice	Group	B.2.2.191	*hc:SpecificationGroup*	1 …∞
	Specification	B.2.2.192	*hc:Specification*	
NOTE—Choice indicates that only one of these elements may be specified.				

B.2.2.188 Specifications/Certifications

Properties: isRef 0, content complex

The *Specifications/Certifications* child element shall identify traceability information for each specification.

B.2.2.188.1 Attributes

Specifications/Certifications contains no attributes.

B.2.2.188.2 Child elements

Specifications/Certifications contains the following child element:

Name	Subclause	Type	Use
Certification	B.2.2.189	*c:NonBlankString*	1 … ∞

B.2.2.189 Specifications/Certifications/Certification

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Specifications/Certifications/Certification* child element shall identify the certification of the specification.

B.2.2.189.1 Attributes

Specifications/Certifications/Certification contains no attributes.

B.2.2.189.2 Child elements

Specifications/Certifications/Certification contains no child elements.

B.2.2.190 Specifications/Conditions

Base type: *hc:SpecificationConditions*

Properties: isRef 0, content complex

The *Specifications/Conditions* child element shall identify the conditions under which a specification is measured.

B.2.2.190.1 Attributes

Specifications/Conditions contains no attributes.

B.2.2.190.2 Child elements

Specifications/Conditions inherits the child element of *hc:SpecificationConditions* (*Condition*).

B.2.2.191 Specifications/Group

Base type: *hc:SpecificationGroup*

Properties: isRef 0, content complex

The *Specifications/Group* child element shall uniquely name a group of specifications that are sharing a common set of conditions.

B.2.2.191.1 Attributes

Specifications/Group inherits the attribute of *hc:SpecificationGroup* (*name*).

B.2.2.191.2 Child elements

Specifications/Group inherits the child elements of *hc:SpecificationGroup* (*Conditions*, *Description*, *Group*, and *Specification*).

B.2.2.192 Specifications/Specification

Base type: *hc:Specification*

Properties: isRef 0, content complex

The *Specifications/Specification* child element shall identify the specification.

B.2.2.192.1 Attributes

Specifications/Specification inherits the attribute of *hc:Specification* (*name*).

B.2.2.192.2 Child elements

Specifications/Specification inherits the child elements of *hc:Specification* (*Conditions*, *Definition*, *Description*, *Graph*, *ExclusiveOptions*, *Limits*, *RequiredOptions*, and *SupplementalInformation*).

B.2.2.193 Switch

Base type: Extension of *hc:RepeatedItem*

Properties: base *hc:RepeatedItem*

The *Switch* complex type shall be the base type for XML schema elements intended to document properties of a switch.

B.2.2.193.1 Attributes

Switch inherits the attributes of *hc:RepeatedItem* (*baseIndex*, *count*, *incrementBy*, *name*, and *replacementCharacter*).

B.2.2.193.2 Child elements

Switch contains the following child elements, in addition to those inherited from *hc:RepeatedItem* (*Description* and *Extension*):

Name	Subclause	Type	Use
Connections	B.2.2.194	—	Required
Interface	B.2.2.197	*c:Interface*	Required

B.2.2.194 Switch/Connections

Properties: isRef 0, content complex

The *Switch/Connections* child element shall identify relay settings.

B.2.2.194.1 Attributes

Switch/Connections contains no attributes.

B.2.2.194.2 Child elements

Switch/Connections contains the following child element:

Name	Subclause	Type	Use
RelaySetting	B.2.2.195	—	1 … ∞

B.2.2.195 Switch/Connections/RelaySetting

Properties: isRef 0, content complex

The *Switch/Connections/RelaySetting* child element shall identify a relay setting.

B.2.2.195.1 Attributes

Switch/Connections/RelaySetting contains the following attribute:

Name	Type	Description	Use
name	xs:string	A descriptive or common name for the relay's position. Example: Open.	Required

B.2.2.195.2 Child elements

Switch/Connections/RelaySetting contains the following child element:

Name	Subclause	Type	Use
RelayConnection	B.2.2.196	—	0 …∞

B.2.2.196 Switch/Connections/RelaySetting/RelayConnection

Properties: isRef 0, content complex

The *Switch/Connections/RelaySetting/RelayConnection* child element shall identify a path established by the relay setting.

B.2.2.196.1 Attributes

Switch/Connections/RelaySetting/RelayConnection contains the following attributes:

Name	Type	Description	Use
from	*c:NonBlankString*	A descriptive or common name for the beginning point to which the path is associated. Example: J1-34.	Required
to	*c:NonBlankString*	A descriptive or common name for the end point to which the path is associated. Example: J1-243.	Required

B.2.2.196.2 Child elements

Switch/Connections/RelaySetting/RelayConnection contains no child elements.

B.2.2.197 Switch/Interface

Base type: *c:Interface*

Properties: isRef 0, content complex

The *Switch/Interface* child element shall identify the hardware interface to the switch.

B.2.2.197.1 Attributes

Switch/Interface contains no attributes.

B.2.2.197.2 Child elements

Switch/Interface inherits the child element of *c:Interface* (*Ports*).

B.2.2.198 Switching

The *Switching* complex type shall be the base type for XML schema elements intended to document properties of a switching subsystem.

B.2.2.198.1 Attributes

Switching contains no attributes.

B.2.2.198.2 Child elements

Switching contains one of the following child elements:

	Name	Subclause	Type	Use
Choice	CrossPointSwitch	B.2.2.199	*hc:CrossPointSwitch*	1 …∞
	MatrixSwitch	B.2.2.200	*hc:MatrixSwitch*	
	Switch	B.2.2.201	*hc:Switch*	
NOTE—Choice indicates that only one of these elements may be specified.				

B.2.2.199 Switching/CrossPointSwitch

Base type: *hc:CrossPointSwitch*

Properties: isRef 0, content complex

The *Switching/CrossPointSwitch* child element shall document the properties of a cross point switch.

B.2.2.199.1 Attributes

Switching/CrossPointSwitch inherits the attributes of *hc:CrossPointSwitch* (*name* and *lineCount*).

B.2.2.199.2 Child elements

Switching/CrossPointSwitch inherits the child elements of *hc:CrossPointSwitch* (*Columns*, *Description*, *Extension*, and *Rows*).

B.2.2.200 Switching/MatrixSwitch

Base type: *hc:MatrixSwitch*

Properties: isRef 0, content complex

The *Switching/MatrixSwitch* child element shall document the properties of a matrix switch.

B.2.2.200.1 Attributes

Switching/MatrixSwitch inherits the attribute of *hc:MatrixSwitch* (*name*).

B.2.2.200.2 Child elements

Switching/MatrixSwitch inherits the child elements of *hc:MatrixSwitch* (*Columns*, *Description*, *Extension*, and *Rows*).

B.2.2.201 Switching/Switch

Base type: *hc:Switch*

Properties: isRef 0, content complex

The *Switching/Switch* child element shall document the properties of a switch.

B.2.2.201.1 Attributes

Switching/Switch inherits the attributes of *hc:Switch* (*baseIndex*, *count*, *incrementBy*, *name*, and *replacementCharacter*).

B.2.2.201.2 Child elements

Switching/Switch inherits the child elements of *hc:Switch* (*Connections*, *Description*, *Extension*, and *Interface*).

B.2.2.202 SwitchPort

Base type: Extension of *hc:RepeatedItem*

Properties: base *hc:RepeatedItem*

The *SwitchPort* complex type shall be the base type for XML schema elements intended to document properties of the switch port.

B.2.2.202.1 Attributes

SwitchPort inherits the attributes of *hc:RepeatedItem* (*baseIndex*, *count*, *incrementBy*, *name*, and *replacementCharacter*).

B.2.2.202.2 Child elements

SwitchPort contains the following child element, in addition to those inherited from *hc:RepeatedItem* (*Description* and *Extension*):

Name	Subclause	Type	Use
Pin	B.2.2.203	—	1 … ∞

B.2.2.203 SwitchPort/Pin

Properties: isRef 0, content complex

The *SwitchPort/Pin* child element shall identify a physical pin of a switch.

B.2.2.203.1 Attributes

SwitchPort/Pin contains the following attributes:

Name	Type	Description	Use
line	xs:int	The number of lines available to connect the rows or columns.	Required
name	*c:NonBlankString*	A descriptive or common name for the switch pin.	Required

B.2.2.203.2 Child elements

SwitchPort/Pin contains no child elements.

B.2.2.204 Trigger

The *Trigger* complex type shall be the base type for XML schema elements intended to document properties of a trigger signal.

B.2.2.204.1 Attributes

Trigger contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the trigger.	Required

B.2.2.204.2 Child elements

Trigger contains the following child elements:

Name	Subclause	Type	Use
Description	B.2.2.205	*c:NonBlankString*	Optional
TriggerPorts	B.2.2.206	—	Required
TriggerProperties	B.2.2.208	—	Required

B.2.2.205 Trigger/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Trigger/Description* child element shall provide an accurate description of what the trigger signal is (e.g., electrically, in time, what the trigger is based upon).

B.2.2.205.1 Attributes

Trigger/Description contains no attributes.

B.2.2.205.2 Child elements

Trigger/Description contains no child elements.

B.2.2.206 Trigger/TriggerPorts

Properties: isRef 0, content complex

The *Trigger/TriggerPorts* child element shall identify the ports on which the trigger may occur.

B.2.2.206.1 Attributes

Trigger/TriggerPorts contains no attributes.

B.2.2.206.2 Child elements

Trigger/TriggerPorts contains the following child element:

Name	Subclause	Type	Use
TriggerPort	B.2.2.207	*hc:TriggerPort*	1 … ∞

B.2.2.207 Trigger/TriggerPorts/TriggerPort

Base type: *hc:TriggerPort*

Properties: isRef 0, content complex

The *Trigger/TriggerPorts/TriggerPort* child element shall identify the port on which the trigger will occur.

B.2.2.207.1 Attributes

Trigger/TriggerPorts/TriggerPort inherit the attributes of *hc:TriggerPort* (*direction*, *name*, and *type*).

B.2.2.207.2 Child elements

Trigger/TriggerPorts/TriggerPort inherits the child element of *hc:TriggerPort* (*Description*).

B.2.2.208 Trigger/TriggerProperties

Properties: isRef 0, content complex

The *Trigger/TriggerProperties* child element shall identify the signal that will generate the trigger.

B.2.2.208.1 Attributes

Trigger/TriggerProperties contains no attributes.

B.2.2.208.2 Child elements

Trigger/TriggerProperties contains the following child element:

Name	Subclause	Type	Use
TriggerPropertyGroup	B.2.2.209	*hc:TriggerPropertyGroup*	1 … ∞

B.2.2.209 Trigger/TriggerProperties/TriggerPropertyGroup

Base type: *hc:TriggerPropertyGroup*

Properties: isRef 0, content complex

The *Trigger/TriggerProperties/TriggerPropertyGroup* child element shall identify the properties of the trigger signal.

B.2.2.209.1 Attributes

Trigger/TriggerProperties/TriggerPropertyGroup inherits the attribute of *hc:TriggerPropertyGroup* (*name*).

B.2.2.209.2 Child elements

Trigger/TriggerProperties/TriggerPropertyGroup inherits the child elements of *hc:TriggerPropertyGroup* (*Description* and *Extension*).

B.2.2.210 TriggerPort

The *TriggerPort* complex type shall be the base type for XML schema elements intended to document properties of a trigger port.

B.2.2.210.1 Attributes

TriggerPort contains the following attributes:

Name	Type	Description	Use
direction	*c:PortDirection*	An enumeration providing for the specification of the direction in which data move on the described port. Enumeration values are Input, Output, and Bi-Directional.	Required
name	*c:NonBlankString*	A descriptive or common name for the port.	Required
type	*hc:TriggerPortType*	An identification of the type of signal that will be present at the port (i.e., Digital, Analog, Software, or LAN).	Required

B.2.2.210.2 Child elements

TriggerPort contains the following child element:

Name	Subclause	Type	Use
Description	B.2.2.211	*c:NonBlankString*	Optional

B.2.2.211 TriggerPort/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *TriggerPort/Description* child element shall identify the interfaces that this trigger may be routed either **to** or **from**.

B.2.2.211.1 Attributes

TriggerPort/Description contains no attributes.

B.2.2.211.2 Child elements

TriggerPort/Description contains no child elements.

B.2.2.212 TriggerPropertyGroup

Properties: abstract true

The *TriggerPropertyGroup* complex type shall be the base type for XML schema elements intended to document properties of a trigger signal.

B.2.2.212.1 Attributes

TriggerPropertyGroup contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the signal that will generate the trigger.	Required

B.2.2.212.2 Child elements

TriggerPropertyGroup contains the following child elements:

Name	Subclause	Type	Use
Description	B.2.2.213	*c:NonBlankString*	Optional
Extension	B.2.2.214	*c:Extension*	Optional

B.2.2.213 TriggerPropertyGroup/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *TriggerPropertyGroup/Description* child element shall describe the trigger signal.

B.2.2.213.1 Attributes

TriggerPropertyGroup/Description contains no attributes.

B.2.2.213.2 Child elements

TriggerPropertyGroup/Description contains no child elements.

B.2.2.214 TriggerPropertyGroup/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *TriggerPropertyGroup/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.2.2.214.1 Attributes

TriggerPropertyGroup/Extension contains no attributes.

B.2.2.214.2 Child elements

TriggerPropertyGroup/Extension inherits the child element of *c:Extension* (*##other*).

B.2.2.215 Triggers

The *Triggers* complex type shall be the base type for XML schema elements intended to document properties of one or more trigger signals.

B.2.2.215.1 Attributes

Triggers contains no attributes.

B.2.2.215.2 Child elements

Triggers contains the following child elements:

Name	Subclause	Type	Use
Trigger	B.2.2.216	*hc:Trigger*	1 … ∞

B.2.2.216 Triggers/Trigger

Base type: *hc:Trigger*

Properties: isRef 0, content complex

The *Triggers/Trigger* child element shall document the properties of a trigger signal.

B.2.2.216.1 Attributes

Triggers/Trigger inherit the attribute of *hc:Trigger* (*name*).

B.2.2.216.2 Child elements

Triggers/Trigger inherits the child elements of *hc:Trigger* (*Description*, *TriggerPorts*, and *TriggerProperties*).

B.2.2.217 Typical

Base type: Extension of *hc:Specification*

Properties: base *hc:Specification*

The *Typical* complex type shall define specification(s) that the instrument is expected to meet.

B.2.2.217.1 Attributes

Typical contains the following attribute, in addition to those inherited from *hc:Specification* (*name*):

Name	Type	Description	Use
expectedSuccessRange	xs:double	The percentage of actual instruments that would be expected to actually meet the specification (expressed as a percentage). Example: 95.	Optional

B.2.2.217.2 Child elements

Typical inherits the child elements of *hc:Specification* (*Conditions*, *Definition*, *Description*, *Graph*, *ExclusiveOptions*, *Limits*, *RequiredOptions*, and *SupplementalInformation*).

B.2.2.218 VersionIdentifier

The *VersionIdentifier* complex type shall be the base type for XML schema elements intended to document versions of software, firmware, or operating system supported by the entity. This information shall be either the minimum or maximum version number.

B.2.2.218.1 Attributes

VersionIdentifier contains the following child elements:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the version.	Optional
qualifier	xs:NMTOKENS	An indication of whether the version specified is the minimum or maximum.	Required
version	*c:NonBlankString*	An identification of the version number.	Required

B.2.2.218.2 Child elements

VersionIdentifier contains no child elements.

B.2.2.219 VPP

Base type: Extension of *hc:Driver*

Properties: base *hc:Driver*

The *VPP* complex type shall be the base type for XML schema elements intended to document properties of a VMEbus extensions for instrumentation (VXI) plug and play (VPP) driver.

B.2.2.219.1 Attributes

VPP contains the following attribute, in addition to those inherited from *hc:Driver* *Bit16*, *Bit32*, *Bit64*, and *Unified*):

Name	Type	Description	Use
prefix	*c:NonBlankString*	The prefix to be used for all API functions in the VPP driver.	Required

B.2.2.219.2 Child elements

VPP contains no child elements.

B.2.3 Simple types

B.2.3.1 DigitalEdge

Base type: restriction of xs:string

Enumerations: Rising | Falling | Selectable

This type shall be used as the base type for the *DetectionType* XML schema attribute for specifying the edge of a digital trigger signal.

B.2.3.2 DigitalLevel

Base type: restriction of xs:string

Enumerations: High | Low | Selectable

This type shall be used as the base type for the *DetectionType* XML schema attribute for specifying the logic level of a digital trigger signal.

B.2.3.3 ErrorType

Base type: xs:string

This type shall be used as the base type for the *HardwareItemDescription/Errors/Error* XML schema attribute for specifying the severity of an error. Examples: Warning, Error, and Fatal.

B.2.3.4 LevelUnits

Base type: restriction of xs:string

Enumerations: %FullScale | +/-V

This type shall be used as the base type for the *LevelType* XML schema attribute to specify the dimension of this attribute.

B.2.3.5 PulseUnits

Base type: restriction of xs:string

Enumerations: S | mS | uS | nS | pS | fS

This type shall be used as the base type for the *MinPulseWidthType* XML schema attribute for specifying the dimensions of the units.

B.2.3.6 TriggerPortType

Base type: restriction of xs:string

Enumerations: Digital | Analog | Software | LAN

This type shall be used as the base type for the *TriggerPort* XML schema attribute for specifying what type of trigger will be on a particular port.

B.2.4 Attribute groups

None.

B.3 Common element schema—TestEquipment.xsd

target namespace	urn:IEEE-1671:2010:TestEquipment
version	1.12
imported schemas	urn:IEEE-1671:2010:Common urn:IEEE-1671:2010:HardwareCommon

A standard XSD intended as the source of an instance XML document shall contain a single root element. **The TestEquipment XML schema is a reference schema containing only type definitions** that may be used in other XML schemas. **It has no root element, and there will be no XML instance documents directly validated against the TestEquipment XML schema**.

ATML TestEquipment imports ATML Common (see B.1) and ATML HardwareCommon (see B.2); only the ATML TestEquipment unique XML elements are defined within this clause.

B.3.1 Elements

None

B.3.2 Complex types

B.3.2.1 Controller

Base type: Extension of *c:ItemDescription*

Properties: base *c:ItemDescription*

The *Controller* complex type shall be the base type for XML schema elements intended to document the properties of a controller item.

B.3.2.1.1 Attributes

Controller inherits the attributes of *c:ItemDescription* (*name* and *version*).

B.3.2.1.2 Child elements

Controller contains the following child elements, in addition to those inherited from *c:ItemDescription* (*Description*, *Extension*, and *Identification*):

Name	Subclause	Type	Use
AudioCapabilities	B.3.2.2	—	Optional
InstalledSoftware	B.3.2.4	—	Optional
OperatingSystems	B.3.2.6	—	Required
Peripherals	B.3.2.10	—	Optional
PhysicalMemory	B.3.2.12	*c:double*	Required
Processor	B.3.2.13	—	Required

Storage	B.3.2.18	—	Required
VideoCapabilities	B.3.2.21	—	Optional

B.3.2.2 Controller/AudioCapabilities

Properties: isRef 0, content complex

The *Controller/AudioCapabilities* child element shall identify audio capabilities of the controller.

B.3.2.2.1 Attributes

Controller/AudioCapabilities contains no attributes.

B.3.2.2.2 Child elements

Controller/AudioCapabilities contains the following child element:

Name	Subclause	Type	Use
Audio	B.3.2.3	*c:NonBlankString*	1 … ∞

B.3.2.3 Controller/AudioCapabilities/Audio

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Controller/AudioCapabilities/Audio* child element shall identify an audio capability.

B.3.2.3.1 Attributes

Controller/AudioCapabilities/Audio contains no attributes.

B.3.2.3.2 Child elements

Controller/AudioCapabilities/Audio contains no child elements.

B.3.2.4 Controller/InstalledSoftware

Properties: isRef 0, content complex

The *Controller/InstalledSoftware* child element shall identify all software installed on the controller.

B.3.2.4.1 Attributes

Controller/InstalledSoftware contains no attributes.

B.3.2.4.2 Child elements

Controller/InstalledSoftware contains the following child element:

Name	Subclause	Type	Use
Software	B.3.2.5	*c:ItemDescription*	1 … ∞

B.3.2.5 Controller/InstalledSoftware/Software

Base type: *c:ItemDescription*

Properties: isRef 0, content complex

The *Controller/InstalledSoftware/Software* element shall identify a specific installed software item.

B.3.2.5.1 Attributes

Controller/InstalledSoftware inherits the attributes of *c:ItemDescription* (*name* and *version*).

B.3.2.5.2 Child elements

Controller/InstalledSoftware inherits the child elements of *c:ItemDescription* (*Description*, *Extension*, and *Identification*).

B.3.2.6 Controller/OperatingSystems

Properties: isRef 0, content complex

The *Controller/OperatingSystems* child element shall identify all operating systems installed on the controller.

B.3.2.6.1 Attributes

Controller/OperatingSystems contains no attributes.

B.3.2.6.2 Child elements

Controller/OperatingSystems contains the following child element:

Name	Subclause	Type	Use
OperatingSystem	B.3.2.7	*c:ItemDescription*	1 … ∞

B.3.2.7 Controller/OperatingSystems/OperatingSystem

Base type: Extension of *c:ItemDescription*

Properties: isRef 0, content complex

The *Controller/OperatingSystems/OperatingSystem* element shall identify a specific installed operating system.

B.3.2.7.1 Attributes

Controller/OperatingSystems/OperatingSystem inherits the child elements of *c:ItemDescription* (*name* and *version*).

B.3.2.7.2 Child elements

Controller/OperatingSystems/OperatingSystem contains the following child element, in addition to those inherited from *c:ItemDescription* (*Description*, *Extension*, and *Identification*):

Name	Subclause	Type	Use
OperatingSystemUpdates	B.3.2.8	—	Optional

B.3.2.8 Controller/OperatingSystems/OperatingSystem/OperatingSystemUpdates

Properties: isRef 0, content complex

The *Controller/OperatingSystems/OperatingSystem/OperatingSystemUpdates* child element shall identify all operating system updates installed on the controller.

B.3.2.8.1 Attributes

Controller/OperatingSystems/OperatingSystem/OperatingSystemUpdates contains no attributes.

B.3.2.8.2 Child elements

Controller/OperatingSystems/OperatingSystem/OperatingSystemUpdates contains the following child element:

Name	Subclause	Type	Use
OperatingSystemUpdate	B.3.2.9	*c:NonBlankString*	1 … ∞

B.3.2.9 Controller/OperatingSystems/OperatingSystem/OperatingSystemUpdates/OperatingSystemUpdate

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Controller/OperatingSystems/OperatingSystem/OperatingSystemUpdates/OperatingSystemUpdate* child element shall identify an operating system patch, service pack, etc.

B.3.2.9.1 Attributes

Controller/OperatingSystems/OperatingSystem/OperatingSystemUpdates/OperatingSystemUpdate contains no attributes.

B.3.2.9.2 Child elements

Controller/OperatingSystems/OperatingSystem/OperatingSystemUpdates/OperatingSystemUpdate contains no child elements.

B.3.2.10 Controller/Peripherals

Properties: isRef 0, content complex

The *Controller/Peripherals* child element shall identify all peripherals installed on the controller.

B.3.2.10.1 Attributes

Controller/Peripherals contains no attributes.

B.3.2.10.2 Child elements

Controller/Peripherals contains the following child element:

Name	Subclause	Type	Use
Peripheral	B.3.2.11	*c:ItemDescription*	1 … ∞

B.3.2.11 Controller/Peripherals/Peripheral

Base type: *c:ItemDescription*

Properties: isRef 0, content complex

The *Controller/Peripherals/Peripheral* child element shall identify a peripheral.

B.3.2.11.1 Attributes

Controller/Peripherals/Peripheral Controller inherits the attributes of *c:ItemDescription* (*name* and *version*)

B.3.2.11.2 Child elements

Controller/Peripherals/Peripheral inherits the child elements of *c:ItemDescription* (*Description*, *Extension*, and *Identification*)

B.3.2.12 Controller/PhysicalMemory

Base type: *c:double*

Properties: isRef 0, content complex

The *Controller/PhysicalMemory* child element shall identify the physical memory of the controller.

B.3.2.12.1 Attributes

Controller/PhysicalMemory inherits the attributes of *c:double* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.3.2.12.2 Child elements

Controller/PhysicalMemory inherits the child elements of *c:double* (*Confidence*, *ErrorLimits, Range*, and *Resolution*).

B.3.2.13 Controller/Processor

Properties: isRef 0, content complex

The *Controller/Processor* child element shall identify all of the controller's processor(s).

B.3.2.13.1 Attributes

Controller/Processor contains no attributes.

B.3.2.13.2 Child elements

Controller/Processor contains the following child elements:

Name	Subclause	Type	Use
Architecture	B.3.2.14	*c:NonBlankString*	Optional
Quantity	B.3.2.15	xs:int	Required
Speed	B.3.2.16	*c:double*	Required
Type	B.3.2.17	*c:NonBlankString*	Optional

B.3.2.14 Controller/Processor/Architecture

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Controller/Processor/Architecture* child element shall identify the architecture of the processor (e.g., 8086).

B.3.2.14.1 Attributes

Controller/Processor/Architecture contains no attributes.

B.3.2.14.2 Child elements

Controller/Processor/Architecture contains no child elements.

B.3.2.15 Controller/Processor/Quantity

Base type: xs:int

Properties: isRef 0, content simple

The *Controller/Processor/Quantity* child element shall identify the number of processors.

B.3.2.15.1 Attributes

Controller/Processor/Quantity contains no attributes.

B.3.2.15.2 Child elements

Controller/Processor/Quantity contains no child elements.

B.3.2.16 Controller/Processor/Speed

Base type: *c:double*

Properties: isRef 0, content complex

The *Controller/Processor/Speed* child element shall identify the processor's clock speed.

B.3.2.16.1 Attributes

Controller/Processor/Speed inherits the attributes of *c:double* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.3.2.16.2 Child elements

Controller/Processor/Speed inherits the child elements of *c:double* (*Confidence*, *ErrorLimits, Range*, and *Resolution*).

B.3.2.17 Controller/Processor/Type

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Controller/Processor/Type* child element shall identify the type of processor (e.g., PentiumM, PowerPC).

B.3.2.17.1 Attributes

Controller/Processor/Type contains no attributes.

B.3.2.17.2 Child elements

Controller/Processor/Type contains no child elements.

B.3.2.18 Controller/Storage

Properties: isRef 0, content complex

The *Controller/Storage* child element shall identify the controller's disk drives.

B.3.2.18.1 Attributes

Controller/Storage contains no attributes.

B.3.2.18.2 Child elements

Controller/Storage contains the following child element:

Name	Subclause	Type	Use
Drive	B.3.2.19	—	1 … ∞

B.3.2.19 Controller/Storage/Drive

Properties: isRef 0, content complex

The *Controller/Storage/Drive* element shall identify a specific disk drive.

B.3.2.19.1 Attributes

Controller/Storage/Drive contains the following attributes:

Name	Type	Description	Use
bootDrive	xs:Boolean	A Yes or No indication (1 or 0) of whether this disk drive serves as the controller's boot drive.	Optional
name	*c:NonBlankString*	A descriptive or common name for the disk drive. Examples: External Optical and CDROM.	Optional

B.3.2.19.2 Child elements

Controller/Storage/Drive contains the following child element:

Name	Subclause	Type	Use
Size	B.3.2.20	*c:double*	Required

B.3.2.20 Controller/Storage/Drive/Size

Base type: *c:double*

Properties: isRef 0, content complex

The *Controller/Storage/Drive/Size* child element shall identify the disk drive's storage capacity.

B.3.2.20.1 Attributes

Controller/Storage/Drive/Size inherits the attributes of *c:double* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.3.2.20.2 Child elements

Controller/Storage/Drive/Size inherits the child elements of *c:double* (*Confidence*, *ErrorLimits, Range*, and *Resolution*).

B.3.2.21 Controller/VideoCapabilities

Properties: isRef 0, content complex

The *Controller/VideoCapabilities* child element shall identify the video capabilities of the controller.

B.3.2.21.1 Attributes

Controller/VideoCapabilities contains no attributes.

B.3.2.21.2 Child elements

Controller/VideoCapabilities contains the following child element:

Name	Subclause	Type	Use
Video	B.3.2.22	*c:NonBlankString*	1 … ∞

B.3.2.22 Controller/VideoCapabilities/Video

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *Controller/VideoCapabilities/Video* child element shall identify the type of video (e.g., RGB, Raster).

B.3.2.22.1 Attributes

Controller/VideoCapabilities/Video contains no attributes.

B.3.2.22.2 Child elements

Controller/VideoCapabilities/Video contains no child elements.

B.3.2.23 Path

Properties: isRef 0, content complex

The *Path* complex type shall define a signal path within the test equipment.

B.3.2.23.1 Attributes

Path contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the path. Example: DMM HI to receiver/fixture interface (RFI) Block 1 pin 3.	Optional

B.3.2.23.2 Child elements

Path contains the following child elements:

Name	Subclause	Type	Use
Extension	B.3.2.24	*c:Extension*	Optional
PathNodes	B.3.2.25	—	Required
Resistance	B.3.2.27	c:double	Optional
SignalDelays	B.3.2.28	—	Optional
SParameters	B.3.2.31	—	Optional
VSWRValues	B.3.2.37	—	Optional

B.3.2.24 Path/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *Path/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.3.2.24.1 Attributes

Path/Extension contains no attributes.

B.3.2.24.2 Child elements

Path/Extension inherits the child element of *c:Extension* (*##other*).

B.3.2.25 Path/PathNodes

Properties: isRef 0, content complex

The *Path/PathNodes* child element shall define the beginning and end nodes associated with a single- or multiwire path. Switches may be present within a wire path.

B.3.2.25.1 Attributes

Path/PathNodes contains no attributes.

B.3.2.25.2 Child elements

Path/PathNodes contains the following child element:

Name	Subclause	Type	Use
Node	B.3.2.26	*hc:NetworkNode*	2 ... ∞

B.3.2.26 Path/PathNodes/Node

Base type: Extension of *hc:NetworkNode*

Properties: isRef 0, content complex

The *Path/PathNodes/Node* child element shall identify a specific node.

B.3.2.26.1 Attributes

Path/PathNodes/Node contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the node.	Required

B.3.2.26.2 Child elements

Path/PathNodes/Node inherits the child elements of *hc:NetworkNode* (*Description*, *Extension*, and *Path*).

B.3.2.27 Path/Resistance

Base type: *c:double*

Properties: isRef 0, content complex

The *Path/Resistance* child element shall identify the resistance of the path.

B.3.2.27.1 Attributes

Path/Resistance inherits the attributes of *c:double* (*standardUnit*, *nonStandardUnit, unitQualifier*, and *value*).

B.3.2.27.2 Child elements

Path/Resistance inherits the child elements of *c:double* (*Confidence*, *ErrorLimits, Range*, and *Resolution*).

B.3.2.28 Path/SignalDelays

Properties: isRef 0, content complex

The *Path/SignalDelays* child element shall identify the delay times of the signal through the paths.

B.3.2.28.1 Attributes

Path/SignalDelays contains no attributes.

B.3.2.28.2 Child elements

Path/SignalDelays contains the following child element:

Name	Subclause	Type	Use
SignalDelay	B.3.2.29	*c:Limit*	1 … ∞

B.3.2.29 Path/SignalDelays/SignalDelay

Base type: Extension of *c:Limit*

Properties: isRef 0, content complex

The *Path/SignalDelays/SignalDelay* child element shall identify the delay time of the signal through a particular path.

B.3.2.29.1 Attributes

Path/SignalDelays/SignalDelay contains the following attributes, in addition to those inherited from *c:Limit* (*operator* and *name*):

Name	Type	Description	Use
inputPort	*c:NonBlankString*	A descriptive or common name for the input port.	Required
outputPort	*c:NonBlankString*	A descriptive or common name for the output port.	Required

B.3.2.29.2 Child elements

Path/SignalDelays/SignalDelay contains the following child element, in addition to those inherited from *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*):

Name	Subclause	Type	Use
Frequency	B.3.2.30	*c:Limit*	Optional

B.3.2.30 Path/SignalDelays/SignalDelay/Frequency

Base type: *c:Limit*

Properties: isRef 0, content complex

The *Path/SignalDelays/SignalDelay/Frequency* child element shall identify the frequency range of the delay.

B.3.2.30.1 Attributes

Path/SignalDelays/SignalDelay/Frequency inherits the attributes of *c:Limit* (*name* and *operator*).

B.3.2.30.2 Child elements

Path/SignalDelays/SignalDelay/Frequency inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.3.2.31 Path/SParameters

Properties: isRef 0, content complex

The *Path/SParameters* child element shall identify the S-parameters associated with a path.

B.3.2.31.1 Attributes

Path/SParameters contains no attributes.

B.3.2.31.2 Child elements

Path/SParameters contains the following child element:

Name	Subclause	Type	Use
SParameter	B.3.2.32	—	1 … ∞

B.3.2.32 Path/SParameters/SParameter

Properties: isRef 0, content complex

The *Path/SParameters/SParameter* child element shall identify a specific S-parameter associated with the path.

B.3.2.32.1 Attributes

Path/SParameters/SParameter contains the following attributes:

Name	Type	Description	Use
inputPort	*c:NonBlankString*	A descriptive or common name for the input port.	Required
outputPort	*c:NonBlankString*	A descriptive or common name for the output port.	Required

B.3.2.32.2 Child elements

Path/SParameters/SParameter contains the following child element:

Name	Subclause	Type	Use
SParameterData	B.3.2.33	—	1 … ∞

B.3.2.33 Path/SParameters/SParameter/SParameterData

Properties: isRef 0, content complex

The *Path/SParameters/SParameter/SParameterData* child element shall identify a specific S-parameter.

B.3.2.33.1 Attributes

Path/SParameters/SParameter/SParameterData contains no attributes.

B.3.2.33.2 Child elements

Path/SParameters/SParameter/SParameterData contains the following child elements:

Name	Subclause	Type	Use
Frequency	B.3.2.34	*c:double*	Optional
Magnitude	B.3.2.35	*c:double*	Required
PhaseAngle	B.3.2.36	*c:double*	Optional

B.3.2.34 Path/SParameters/SParameter/SParameterData/Frequency

Base type: *c:double*

Properties: isRef 0, content complex

The *Path/SParameters/SParameter/SParameterData/Frequency* child element shall identify the frequency of the S-parameter.

B.3.2.34.1 Attributes

Path/SParameters/SParameter/SParameterData/Frequency inherits the attributes of *c:double* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.3.2.34.2 Child elements

Path/SParameters/SParameter/SParameterData/Frequency inherits the child elements of *c:double* (*Confidence*, *ErrorLimits, Range*, and *Resolution*).

B.3.2.35 Path/SParameters/SParameter/SParameterData/Magnitude

Base type: *c:double*

Properties: isRef 0, content complex

The *Path/SParameters/SParameter/SParameterData/Magnitude* child element shall identify the magnitude of the S-parameter.

B.3.2.35.1 Attributes

Path/SParameters/SParameter/SParameterData/Frequency inherits the attributes of *c:double* (*nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.3.2.35.2 Child elements

Path/SParameters/SParameter/SParameterData/Magnitude inherits the child elements of *c:double* (*Confidence*, *ErrorLimits, Range*, and *Resolution*).

B.3.2.36 Path/SParameters/SParameter/SParameterData/PhaseAngle

Base type: *c:double*

Properties: isRef 0, content complex

The *Path/SParameters/SParameter/SParameterData/PhaseAngle* child element shall identify the phase angle of the S-parameter.

B.3.2.36.1 Attributes

Path/SParameters/SParameter/SParameterData/Frequency inherits the attributes of *c:double* *nonStandardUnit*, *standardUnit*, *unitQualifier*, and *value*).

B.3.2.36.2 Child elements

Path/SParameters/SParameter/SParameterData/PhaseAngle inherits the child elements of *c:double* (*Confidence*, *ErrorLimits, Range*, and *Resolution*).

B.3.2.37 Path/VSWRValues

Properties: isRef 0, content complex

The *Path/VSWRValues* child element shall identify the voltage standing wave ratio(s) (VSWRs) associated with a single- or multiwire path. Switches may be present within a wire path.

B.3.2.37.1 Attributes

Path/VSWRValues contains no attributes.

B.3.2.37.2 Child elements

Path/VSWRValues contains the following child element:

Name	Subclause	Type	Use
VSWRValue	B.3.2.38	*c:Limit*	1 … ∞

B.3.2.38 Path/VSWRValues/VSWRValue

Base type: Extension of *c:Limit*

Properties: isRef 0, content complex

The *Path/VSWRValues/VSWRValue* child element shall identify the actual VSWR value.

B.3.2.38.1 Attributes

Path/VSWRValues/VSWRValue contains the following attribute, in addition to those inherited from *c:Limit* (*name* and *operator*):

Name	Type	Description	Use
inputPort	—	A descriptive or common name for the input port.	Required

B.3.2.38.2 Child elements

Path/VSWRValues/VSWRValue contains the following child element, in addition to those inherited from *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*):

Name	Subclause	Type	Use
Frequency	B.3.2.39	*c:Limit*	Optional

B.3.2.39 Path/VSWRValues/VSWRValue/Frequency

Base type: *c:Limit*

Properties: isRef 0, content complex

The *Path/VSWRValues/VSWRValue/Frequency* child element shall identify the frequency range at which the VSWR of the path is valid.

B.3.2.39.1 Attributes

Path/VSWRValues/VSWRValue/Frequency inherits the attributes of *c:Limit* (*name* and *operator*).

B.3.2.39.2 Child elements

Path/VSWRValues/VSWRValue/Frequency inherits the child elements of *c:Limit* (*Description*, *Expected*, *Extension*, *LimitPair*, *Mask*, and *SingleLimit*).

B.3.2.40 PathNode

Base type: Extension of *hc:NetworkNode*

Properties: isRef 0, content complex

The *PathNode* complex type shall be the base type for XML schema elements intended to document a node within a path.

B.3.2.40.1 Attributes

PathNode contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the node. Used to reference the node when specifying path loss data.	Required

B.3.2.40.2 Child elements

PathNode inherits the child elements of *hc:NetworkNode* (*Description*, *Extension*, and *Path*).

B.3.2.41 Paths

The *Paths* complex type shall be the base type for XML schema elements intended to document the paths within the test equipment.

B.3.2.41.1 Attributes

Paths contains no attributes.

B.3.2.41.2 Child elements

Paths contains the following child element:

Name	Subclause	Type	Use
Path	B.3.2.42	*te:Path*	1 … ∞

B.3.2.42 Paths/Path

Base type: *te:Path*

Properties: isRef 0, content complex

The *Paths/Path* child element shall define a signal path within the test equipment.

B.3.2.42.1 Attributes

Paths/Path contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the path. Example: DMM HI to RFI Block 1 pin 3.	Optional

B.3.2.42.2 Child elements

Paths/Path inherits the child elements of *te:Path* (*Extension*, *PathNodes*, *SignalDelays*, *SParameters*, and *VSWRValues*)

B.3.2.43 Software

Properties: isRef 0, content complex

The *Software* complex type shall be the base type for XML schema elements intended to document software not installed on the controller (e.g., self-test, calibration).

B.3.2.43.1 Attributes

Software contains no attributes.

B.3.2.43.2 Child elements

Software contains the following child element:

Name	Subclause	Type	Use
SoftwareItem	B.3.2.44	*c:ItemDescription*	1 … ∞

B.3.2.44 Software/SoftwareItem

Base type: *c:ItemDescription*

Properties: isRef 0, content complex

The *Software/SoftwareItem* child element shall identify the software program.

B.3.2.44.1 Attributes

Software/SoftwareItem inherits the attributes of *c:ItemDescription* (*name* and *version*).

B.3.2.44.2 Child elements

Software/SoftwareItem inherits the child elements of *c:ItemDescription* (*Description*, *Extension*, and *Identification*).

B.3.2.45 TestEquipment

Base type: Extension of *hc:HardwareItemDescription*

Properties: base *hc:HardwareItemDescription*

The *TestEquipment* complex type shall be the base type for XML schema elements intended to document a family of test stations or test adapters.

B.3.2.45.1 Attributes

TestEquipment inherits the attributes of *hc:HardwareItemDescription* (*name* and *version*).

B.3.2.45.2 Child elements

TestEquipment contains the following child elements, in addition to those inherited from *hc:HardwareItemDescription* (*CalibrationRequirements*, *Components*, *ConfigurationOptions*, *Control*, *Description*, *Documentation*, *EnvironmentalRequirements*, *Errors*, *Extension*, *FactoryDefaults*, *Identification*, *Interface*, *LegalDocuments*, *NetworkList*, *OperationalRequirements*, *ParentComponents*, *PhysicalCharacteristics*, and *PowerRequirements*):

Name	Subclause	Type	Use
Capabilities	B.3.2.46	*hc:Capabilities*	Optional
Controllers	B.3.2.47	—	Optional
FacilitiesRequirements	B.3.2.49	*hc:FacilitiesRequirements*	Optional
Paths	B.3.2.50	*te:Paths*	Optional
Resources	B.3.2.51	*hc:Resources*	Optional
Software	B.3.2.52	*te:Software*	Optional
Specifications	B.3.2.53	*hc:Specifications*	Optional
Switching	B.3.2.54	*hc:Switching*	Optional
TerminalBlocks	B.3.2.55	—	Optional

B.3.2.46 TestEquipment/Capabilities

Base type: *hc:Capabilities*

Properties: isRef 0, content complex

The *TestEquipment/Capabilities* child element shall identify the capabilities of the test equipment.

B.3.2.46.1 Attributes

TestEquipment/Capabilities contains no attributes.

B.3.2.46.2 Child elements

TestEquipment/Capabilities inherits the child elements of *hc:Capabilities* (*CapabilitiesReference*, *Capability*, and *CapabilityMap*).

B.3.2.47 TestEquipment/Controllers

Properties: isRef 0, content complex

The *TestEquipment/Controllers* element shall identify an ordered list of test station or test adapter controllers.

B.3.2.47.1 Attributes

TestEquipment/Controllers contains no attributes.

B.3.2.47.2 Child elements

TestEquipment/Controllers contains the following child element:

Name	Subclause	Type	Use
Controller	B.3.2.48	*te:Controller*	1 … ∞

B.3.2.48 TestEquipment/Controllers/Controller

Base type: *te:Controller*

Properties: isRef 0, content complex

The *TestEquipment/Controllers/Controller* element shall identify an individual test station or test adapter controller.

B.3.2.48.1 Attributes

TestEquipment/Controllers/Controller inherits the attributes of *te:Controller* (*name* and *version*).

B.3.2.48.2 Child elements

TestEquipment/Controllers/Controller inherits the child elements of *te:Controller* (*AudioCapabilities*, *Description*, *Extension*, *Identification*, *InstalledSoftware*, *OperatingSystems*, *Peripherals*, *PhysicalMemory*, *Processor*, *Storage*, and *VideoCapabilities*).

B.3.2.49 TestEquipment/FacilitiesRequirements

Base type: *hc:FacilitiesRequirements*

Properties: isRef 0, content complex

The *TestEquipment/FacilitiesRequirements* child element shall identify the facility requirements.

B.3.2.49.1 Attributes

TestEquipment/FacilitiesRequirements contains no attributes.

B.3.2.49.2 Child elements

TestEquipment/FacilitiesRequirements inherits the child elements of *hc:FacilitiesRequirements* (*Cooling*, *Extension*, *FacilitiesInterface*, *FacilityRequirementsDocuments*, *Hydraulic*, and *Pneumatic*).

B.3.2.50 TestEquipment/Paths

Base type: *te:Paths*

Properties: isRef 0, content complex

The *TestEquipment/Paths* child element shall identify the characteristics of the signal paths through the test equipment and interface hardware.

B.3.2.50.1 Attributes

TestEquipment/Paths contains no attributes.

B.3.2.50.2 Child elements

TestEquipment/FacilitiesRequirements inherits the child element of *te:Paths* (*Path*).

B.3.2.51 TestEquipment/Resources

Base type: *hc:Resources*

Properties: isRef 0, content complex

The *TestEquipment/Resources* child element shall identify the resources within the test equipment.

B.3.2.51.1 Attributes

TestEquipment/Resources contains no attributes.

B.3.2.51.2 Child elements

TestEquipment/Resources inherits the child element of *hc:Resources* (*Resource*).

B.3.2.52 TestEquipment/Software

Base type: *te:Software*

Properties: isRef 0, content complex

The *TestEquipment/Software* child element shall identify the software within the test equipment.

B.3.2.52.1 Attributes

TestEquipment/Software contains no attributes.

B.3.2.52.2 Child elements

TestEquipment/Software inherits the child element of *te:Software* (*SoftwareItem*).

B.3.2.53 TestEquipment/Specifications

Base type: *hc:Specifications*

Properties: isRef 0, content complex

The *TestEquipment/Specifications* child element shall identify the specifications of the test equipment.

B.3.2.53.1 Attributes

TestEquipment/Specifications contains no attributes.

B.3.2.53.2 Child elements

TestEquipment/Specifications inherits the child elements of *hc:Specifications* (*Certifications*, *Conditions*, *Group*, and *Specification*).

B.3.2.54 TestEquipment/Switching

Base type: *hc:Switching*

Properties: isRef 0, content complex

The *TestEquipment/Switching* child element shall identify the switching within the test equipment.

B.3.2.54.1 Attributes

TestEquipment/Switching contains no attributes.

B.3.2.54.2 Child elements

TestEquipment/Switching inherits the child elements of *hc:Switching* (*CrossPointSwitch*, *MatrixSwitch*, and *Switch*).

B.3.2.55 TestEquipment/TerminalBlocks

Properties: isRef 0, content complex

The *TestEquipment/TerminalBlocks* child element shall identify the terminal blocks within the test equipment.

B.3.2.55.1 Attributes

TestEquipment/TerminalBlocks contains no attributes.

B.3.2.55.2 Child elements

TestEquipment/TerminalBlocks contains the following child element:

Name	Subclause	Type	Use
TerminalBlock	B.3.2.56	*hc:RepeatedItem*	1 … ∞

B.3.2.56 TestEquipment/TerminalBlocks/TerminalBlock

Base type: Extension of *hc:RepeatedItem*

Properties: isRef 0, content complex

The *TestEquipment/TerminalBlocks/TerminalBlock* child element shall identify a terminal block.

B.3.2.56.1 Attributes

TestEquipment/TerminalBlocks/TerminalBlock inherits the attributes of *hc:RepeatedItem* (*baseIndex*, *count*, *incrementedBy*, *name*, and *replacementCharacter*).

B.3.2.56.2 Child elements

TestEquipment/TerminalBlocks/TerminalBlock contains the following child element, in addition to those inherited from *hc:RepeatedItem* (*Description* and *Extension*):

Name	Subclause	Type	Use
Interface	B.3.2.57	*c:Interface*	Required

B.3.2.57 TestEquipment/TerminalBlocks/TerminalBlock/Interface

Base type: *c:Interface*

Properties: isRef 0, content complex

The *TestEquipment/Switching/Interface* child element shall identify the terminal block interface.

B.3.2.57.1 Attributes

TestEquipment/Switching/Interface contains no attributes.

B.3.2.57.2 Child elements

TestEquipment/Switching/Interface inherits the child element of *c:Interface* (*Ports*).

B.3.2.58 TestEquipmentInstance

Base type: Extension of *c:HardwareInstance*

Properties: base *c:HardwareInstance*

The *TestEquipment* complex type shall be the base type for XML schema elements intended to document a specific test station or test adapter.

B.3.2.58.1 Attributes

TestEquipmentInstance contains no attributes.

B.3.2.58.2 Child elements

TestEquipment/Instance contains the following child elements, in addition to those inherited from *hc:HardwareInstance* (*Calibration*, *Components*, *Definition*, *DescriptionDocumentReference*, *ManufactureDate*, *ParentComponent*, *PowerOn*, and *SerialNumber*):

Name	Subclause	Type	Use
Capabilities	B.3.2.59	*hc:Capabilities*	Optional
Configuration	B.3.2.60	*c:NonBlankString*	Optional
Controllers	B.3.2.61	—	Optional
Extension	B.3.2.63	*c:Extension*	Optional
Paths	B.3.2.64	*te:Paths*	Optional
SelfTestRuns	B.3.2.65	—	Optional
Software	B.3.2.70	*te:Software*	Optional
SubSystemCalibration	B.3.2.71	—	Optional

B.3.2.59 TestEquipmentInstance/Capabilities

Base type: *hc:Capabilities*

Properties: isRef 0, content complex

The *TestEquipmentInstance/Capabilities* child element shall identify the capabilities of the specific piece of test equipment.

B.3.2.59.1 Attributes

TestEquipmentInstance/Capabilities contains no attributes.

B.3.2.59.2 Child elements

TestEquipmentInstance/Capabilities inherits the child elements of *hc:Capabilities* (*CapabilitiesReference*, *Capability*, and *CapabilityMap*).

B.3.2.60 TestEquipmentInstance/Configuration

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *TestEquipmentInstance/Configuration* child element shall identify the configuration of the specific piece of test equipment.

B.3.2.60.1 Attributes

TestEquipmentInstance/Configuration contains no attributes.

B.3.2.60.2 Child elements

TestEquipmentInstance/Configuration contains no child elements.

B.3.2.61 TestEquipmentInstance/Controllers

Properties: isRef 0, content complex

The *TestEquipmentInstance/Controllers* element shall identify an ordered list of test station or test adapter controllers.

B.3.2.61.1 Attributes

TestEquipmentInstance/Controllers contains no attributes.

B.3.2.61.2 Child elements

TestEquipmentInstance/Controllers contains the following child element:

Name	Subclause	Type	Use
Controller	B.3.2.62	*te:Controller*	1 … ∞

B.3.2.62 TestEquipmentInstance/Controllers/Controller

Base type: *te:Controller*

Properties: isRef 0, content complex

The *TestEquipmentInstance/Controllers/Controller* element shall identify an individual test station or test adapter controller.

B.3.2.62.1 Attributes

TestEquipmentInstance/Controllers/Controller inherits the attributes of *te:Controller* (*name* and *version*).

B.3.2.62.2 Child elements

TestEquipmentInstance/Controllers/Controller inherits the child elements of *te:Controller* (*AudioCapabilities*, *Description*, *Extension*, *Identification*, *InstalledSoftware*, *OperatingSystems*, *Peripherals*, *PhysicalMemory*, *Processor*, *Storage*, and *VideoCapabilities*).

B.3.2.63 TestEquipmentInstance/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *TestEquipmentInstance/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.3.2.63.1 Attributes

TestEquipmentInstance/Extension contains no attributes.

B.3.2.63.2 Child elements

TestEquipmentInstance/Extension inherits the child element of *c:Extension (##other*).

B.3.2.64 TestEquipmentInstance/Paths

Base type: *te:Paths*

Properties: isRef 0, content complex

The *TestEquipmentInstance/Paths* child element shall identify the signal paths through the specific piece of test equipment.

B.3.2.64.1 Attributes

TestEquipmentInstance/Paths contains no attributes.

B.3.2.64.2 Child elements

TestEquipmentInstance/Paths inherits the child element of *te:Paths* (*Path*).

B.3.2.65 TestEquipmentInstance/SelfTestRuns

Properties: isRef 0, content complex

The *TestEquipmentInstance/SelfTestRuns* child element shall identify self-test end-to-end runs on the specific piece of test equipment.

B.3.2.65.1 Attributes

TestEquipmentInstance/SelfTestRuns contains no attributes.

B.3.2.65.2 Child elements

TestEquipmentInstance/SelfTestRuns contains the following child element:

Name	Subclause	Type	Use
SelfTestRun	B.3.2.66	—	1 … ∞

B.3.2.66 TestEquipmentInstance/SelfTestRuns/SelfTestRun

Properties: isRef 0, content complex

The *TestEquipmentInstance/SelfTestRuns/SelfTestRun* child element shall identify the last self-test end-to-end run.

B.3.2.66.1 Attributes

TestEquipmentInstance/SelfTestRuns/SelfTestRun contains the following attributes:

Name	Type	Description	Use
date	xs:dateTime	The date and time the self-test was run.	Required
name	*c:NonBlankString*	A descriptive or common name for the self-test last executed end to end.	Required
version	*c:NonBlankString*	A string designating the version of the self-test last executed end to end.	Optional

B.3.2.66.2 Child elements

TestEquipmentInstance/SelfTestRuns/SelfTestRun contains the following child elements:

Name	Subclause	Type	Use
Description	B.3.2.67	*c:NonBlankString*	Optional
Extension	B.3.2.68	*c:Extension*	Optional
InstanceDocumentReference	B.3.2.69	*c:DocumentReference*	Optional

B.3.2.67 TestEquipmentInstance/SelfTestRuns/SelfTestRun/Description

Base type: *c:NonBlankString*

Properties: isRef 0, content simple

Facets: minLength 1, whiteSpace replace

The *TestEquipmentInstance/SelfTestRuns/SelfTestRun/Description* child element shall identify the self-test that was run.

B.3.2.67.1 Attributes

TestEquipmentInstance/SelfTestRuns/SelfTestRun/Description contains no attributes.

B.3.2.67.2 Child elements

TestEquipmentInstance/SelfTestRuns/SelfTestRun/Description contains no child elements.

B.3.2.68 TestEquipmentInstance/SelfTestRuns/SelfTestRun/Extension

Base type: *c:Extension*

Properties: isRef 0, content complex

The *TestEquipmentInstance/SelfTestRuns/SelfTestRun/Extension* child element shall provide a specific extension point for use cases that require elements not provided in the basic structure.

B.3.2.68.1 Attributes

TestEquipmentInstance/SelfTestRuns/SelfTestRun/Extension contains no attributes.

B.3.2.68.2 Child elements

TestEquipmentInstance/SelfTestRuns/SelfTestRun/Extension inherits the child element of *c:Extension* (*##other*).

B.3.2.69 TestEquipmentInstance/SelfTestRuns/SelfTestRun/InstanceDocumentReference

Base type: *c:DocumentReference*

Properties: isRef 0, content complex

The *TestEquipmentInstance/SelfTestRuns/SelfTestRun/InstanceDocumentReference* child element shall identify the instance document associated with this self-test run.

B.3.2.69.1 Attributes

TestEquipmentInstance/SelfTestRuns/SelfTestRun/InstanceDocumentReference contains the following attributes:

Name	Type	Description	Use
ID	*c:NonBlankString*	A user-defined string uniquely identifying the instance document.	Required
uuid	*c:Uuid*	The instance document associated with this self-test run.	Required

B.3.2.69.2 Child elements

TestEquipmentInstance/SelfTestRuns/SelfTestRun/InstanceDocumentReference contains no child elements.

B.3.2.70 TestEquipmentInstance/Software

Base type: *te:Software*

Properties: isRef 0, content complex

The *TestEquipmentInstance/Software* child element shall identify the software within the test equipment.

B.3.2.70.1 Attributes

TestEquipmentInstance/Software contains no attributes.

B.3.2.70.2 Child elements

TestEquipmentInstance/Software inherits the child element of *te:Software* (*SoftwareItem*).

B.3.2.71 TestEquipmentInstance/SubSystemCalibration

Properties: isRef 0, content complex

The *TestEquipmentInstance/SubSystemCalibration* child element shall identify the subsystem(s) calibrated independently of system calibration.

B.3.2.71.1 Attributes

TestEquipmentInstance/SubSystemCalibration contains no attributes.

B.3.2.71.2 Child elements

TestEquipmentInstance/SubSystemCalibration contains the following child element:

Name	Subclause	Type	Use
SubSystem	B.3.2.72	—	1 … ∞

B.3.2.72 TestEquipmentInstance/SubSystemCalibration/SubSystem

Properties: isRef 0, content complex

The *TestEquipmentInstance/SubSystemCalibration/SubSystem* child element shall identify the subsystem calibrated.

B.3.2.72.1 Attributes

TestEquipmentInstance/SubSystemCalibration/SubSystem contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the subsystem calibrated.	Required

B.3.2.72.2 Child elements

TestEquipmentInstance/SubSystemCalibration/SubSystem contains the following child elements:

Name	Subclause	Type	Use
CalibrationDate	B.3.2.73	xs:dateTime	Required
CalibrationFrequency	B.3.2.74	xs:duration	Optional

B.3.2.73 TestEquipmentInstance/SubSystemCalibration/SubSystem/CalibrationDate

Base type: xs:dateTime

Properties: isRef 0, content simple

The *TestEquipmentInstance/SubSystemCalibration/SubSystem/CalibrationDate* child element shall identify the date and time the subsystem was last calibrated.

B.3.2.73.1 Attributes

TestEquipmentInstance/SubSystemCalibration/SubSystem/CalibrationDate contains no attributes.

B.3.2.73.2 Child elements

TestEquipmentInstance/SubSystemCalibration/SubSystem/CalibrationDate contains no child elements.

B.3.2.74 TestEquipmentInstance/SubSystemCalibration/SubSystem/CalibrationFrequency

Base type: xs:duration

Properties: isRef 0, content simple

The *TestEquipmentInstance/SubSystemCalibration/SubSystem/CalibrationFrequency* child element shall identify the how frequently the subsystem calibration is to be run.

B.3.2.74.1 Attributes

TestEquipmentInstance/SubSystemCalibration/SubSystem/CalibrationFrequency contains no attributes.

B.3.2.74.2 Child elements

TestEquipmentInstance/SubSystemCalibration/SubSystem/CalibrationFrequency contains no child elements.

B.3.3 Simple types

None

B.3.4 Attribute groups

None

Annex C

(normative)

ATML internal model schemas

C.1 ATML internal model schema—Capabilities.xsd

C.1.1 Elements

C.1.1.1 Capabilities root (or document)

Exactly one element exists, called the root or the document element, of which no part appears in the content of any other element. This root element serves as the parent for all other elements of the ATML Capabilities XML schema.

The ATML Capabilities XML schema root element is defined as follows:

Name	Set to
Attribute Form Default	unqualified (see NOTE)
Element Form Default	qualified (see NOTE)
Encoding	UTF-8
Included Schema	—
Imported Schema	urn:IEEE-1671:2010:Common urn:IEEE-1671:2010:HardwareCommon
Target Namespace	urn:IEEE-1671:2010:Capabilities
Version	1.10
XML Schema Namespace Reference[a]	
NOTE: Qualified and unqualified are described in A.3.6.	

[a] The namespace reference URL is http://www.w3.org/2001/XMLSchema.

C.1.1.2 Capabilities

Base type: Extension of *ca:Capabilities*

Properties: content complex

The *Capabilities* element defines a static list of Capabilities of the item, independent of any particular ATML family of standards XML schema (e.g., documenting the capabilities of an instrument, not within an ATML InstrumentDescription document).

Figure C.1 illustrates the XML types inherited and the XML types (both simple and complex) that shall be defined in this standard that together constitute *Capabilities*.

Within Figure C.1, “solid lined boxes” indicate that the XML element shall be required.

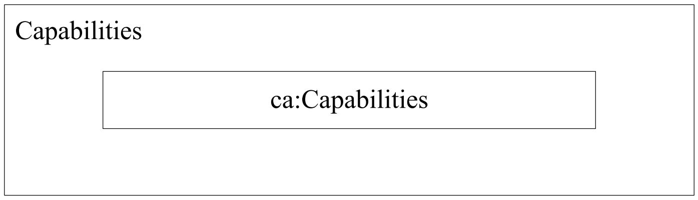


Figure C.1—Capabilities element

C.1.1.2.1 Attributes

Capabilities contains the following attributes, in addition to those inherited from the *c:DocumentRootAttributes* attribute group of the Common XML schema defined in B.1 (*classified*, *securityClassification* and *uuid*):

Name	Type	Description	Use
name	*c:NonBlankString*	The name of the instance document. Example: Acme Widget.	Optional
version	*c:NonBlankString*	The version of the instance document. Example: 2.	Optional

C.1.1.2.2 Child elements

Capabilities inherits the child element of *ca:Capabilities* (*Capability*).

C.1.2 Complex types

C.1.2.1 Capabilities

The *Capabilities* complex type groups a list (one or more) of capabilities.

C.1.2.1.1 Attributes

Capabilities contains no attributes.

C.1.2.1.2 Child elements

Capabilities contains the following child element:

Name	Subclause	Type	Use
Capability	C.1.2.2	*hc:capability*	1 … ∞

C.1.2.2 Capabilities/Capability

Base type: *hc:Capability*

Properties: isRef 0, content complex

The *Capabilities/Capability* element identifies a capability.

C.1.2.2.1 Attributes

Capabilities/Capability contains the following attribute:

Name	Type	Description	Use
name	*c:NonBlankString*	A descriptive or common name for the capability.	Required

C.1.2.2.2 Child elements

Capabilities/Capability inherits the child elements of *hc:Capability* (*Description*, *Extension*, *Interface*, and *SignalDescription*).

C.1.3 Simple types

None

C.1.4 Attribute groups

None

C.2 ATML internal model schema—WireLists.xsd

C.2.1 Elements

C.2.1.1 WireLists root (or document)

Exactly one element exists, called the root or the document element, of which no part appears in the content of any other element. This root element serves as the parent for all other elements of the ATML WireLists XML schema.

The ATML WireLists XML schema root element is defined as follows:

Name	Set to
Attribute Form Default	unqualified (see NOTE)
Element Form Default	qualified (see NOTE)
Encoding	UTF-8
Included Schema	—
Imported Schema	urn:IEEE-1671:2010:Common urn:IEEE-1671:2010:HardwareCommon
Target Namespace	urn:IEEE-1671:2010:WireLists
Version	1.11
XML Schema Namespace Reference[a]	
NOTE: Qualified and unqualified are described in A.3.6.	

[a] The namespace reference URL is http://www.w3.org/2001/XMLSchema.

C.2.1.2 WireLists

Base type: Extension of *w:Wirelists*

Properties: content complex

The *WireLists* element is used as a collector of one or more wire lists and/or test wire lists.

Figure C.2 illustrates the XML types inherited and the XML types (both simple and complex) that shall be defined in this standard that together constitute *WireLists*.

Within Figure C.2, “solid lined boxes” indicate that the XML element shall be required.

Figure C.2—WireLists element

C.2.1.2.1 Attributes

WireLists contains the following attributes, in addition to those inherited from the *c:DocumentRootAttributes* attribute group of the Common XML schema defined in B.1 (*classified*, *securityClassification* and *uuid*):

Name	Type	Description	Use
name	*c:NonBlankString*	The name of the instance document. Example: Test Oriented Wire List for Test 0500.	Optional
version	*c:NonBlankString*	The version of the instance document. Example: 2.	Optional

C.2.1.2.2 Child elements

WireLists inherits the child elements of *w:Wirelists* (*Items*, *TestDescription*, *WireList*, and *TestWireList*).

C.2.2 Complex types

C.2.2.1 AssetWireList

The *AssetWireList* complex type is used as a container of wires associated with an asset.

C.2.2.1.1 Attributes

AssetWireList contains no attributes.

C.2.2.1.2 Child elements

AssetWireList contains the following child elements:

Name	Subclause	Type	Use
Asset	C.2.2.2	*hc:NetworkNode*	Required
Wire	C.2.2.3	*hc:Network*	1 … ∞

C.2.2.2 AssetWireList/Asset

Base type: *hc:NetworkNode*

Properties: isRef 0, content complex

The *AssetWireList/Asset* element identifies an asset.

C.2.2.2.1 Attributes

AssetWireList/Asset contains no attributes.

C.2.2.2.2 Child elements

AssetWireList/Asset inherits the child elements of *hc:NetworkNode* (*Description*, *Extension*, and *Path*).

C.2.2.3 AssetWireList/Wire

Base type: *hc:Network*

Properties: isRef 0, content complex

The *AssetWireList/Wire* element identifies the wires to and from the identified asset. The signal(s) on the wire(s) may be identified via XPath reference to either the actual IEEE 1641 signal or the signal in the TP.

C.2.2.3.1 Attributes

AssetWireList/Wire inherits the attributes of *hc:Network* (*baseIndex*, *count*, *incrementedBy*, and *replacementCharacter*).

C.2.2.3.2 Child elements

AssetWireList/Wire inherits the child elements of *hc:Network*. (*Description*, *Extension*, and *Node*).

C.2.2.4 TestWireList

The *TestWireList* complex type is used as a container of wires associated with a test.

C.2.2.4.1 Attributes

TestWireList contains no attributes.

C.2.2.4.2 Child elements

TestWireList contains the following child elements:

Name	Subclause	Type	Use
AssetWireList	C.2.2.5	*w:AssetWireList*	1 … ∞
Test	C.2.2.6	*hc:NetworkNode*	Required

C.2.2.5 TestWireList/AssetWireList

Base type: *w:AssetWireList*

Properties: isRef 0, content complex

The *TestWireList/AssetWireList* element identifies, for a particular test, the wires associated with an asset.

C.2.2.5.1 Attributes

TestWireList/AssetWireList contains no attributes.

C.2.2.5.2 Child elements

TestWireList/AssetWireList inherits the child elements of *w:AssetWireList* (*Asset* and *Wire*).

C.2.2.6 TestWireList/Test

Base type: *hc:NetworkNode*

Properties: isRef 0, content complex

The *TestWireList/Test* element identifies the particular test.

C.2.2.6.1 Attributes

TestWireList/Test contains no attributes.

C.2.2.6.2 Child elements

TestWireList/Test inherits the child elements of *hc:NetworkNode* (*Description*, *Extension*, and *Path*).

C.2.2.7 WireList

The *WireList* complex type is used as a container of one or more wires.

C.2.2.7.1 Attributes

WireList contains no attributes.

C.2.2.7.2 Child elements

WireList contains the following child element:

Name	Subclause	Type	Use
Wire	C.2.2.8	*hc:Network*	1 … ∞

C.2.2.8 WireList/Wire

Base type: *hc:Network*

Properties: isRef 0, content complex

The *WireList/Wire* element identifies a particular wire.

C.2.2.8.1 Attributes

WireList/Wire inherits the attributes of *hc:Network* (*baseIndex*, *count*, *incrementedBy*, and *replacementCharacter*).

C.2.2.8.2 Child elements

WireList/Wire inherits the child elements of *hc:Network* (*Description*, *Extension*, and *Node*)

C.2.2.9 WireLists

The *WireLists* complex type is used as a container of one or more *WireList* complex types.

C.2.2.9.1 Attributes

WireLists contains no attributes.

C.2.2.9.2 Child elements

WireLists contains the following child elements:

	Name	Subclause	Type	Use
	Items	C.2.2.10	—	Required
	TestDescription	C.2.2.12	*c:DocumentReference*	Optional
Choice	TestWireList	C.2.2.13	*w:TestWireList*	1 … ∞
	WireList	C.2.2.14	*w:WireList*	
NOTE—Choice indicates that only one of these elements may be specified.				

C.2.2.10 WireLists/Items

Properties: isRef 0, content complex

The *WireLists/Items* element identifies a list of item description or item instance documents that contain the nodes referenced in the wire list.

C.2.2.10.1 Attributes

WireLists/Items contains no attributes.

C.2.2.10.2 Child elements

WireLists/Items contains the following child element:

Name	Subclause	Type	Use
Item	C.2.2.11	c:DocumentReference	1 … ∞

C.2.2.11 WireLists/Items/Item

Base type: *c:DocumentReference*

Properties: isRef 0, content complex

The *WireList/Items/Item* element identifies an item description or item instance document that contains the nodes referenced in the wire list.

C.2.2.11.1 Attributes

WireList/Items/Item inherits the attributes of *c:DocumentReference* (*ID* and *uuid*)

C.2.2.11.2 Child elements

WireList/Items/Item contains no child elements.

C.2.2.12 WireLists/TestDescription

Base type: *c:DocumentReference*

Properties: isRef 0, content complex

The *WireLists/TestDescription* element identifies the ATML Test Description instance document that describes the tests referenced in the *w:TestWireList*.

C.2.2.12.1 Attributes

WireLists/TestDescription inherits the attributes of *c:DocumentReference* (*ID* and *uuid*).

C.2.2.12.2 Child elements

WireLists/TestDescription contains no child elements.

C.2.2.13 WireLists/TestWireList

Base type: *w:TestWireList*

Properties: isRef 0, content complex

The *WireLists/TestWireList* element identifies a list of test-oriented wires.

C.2.2.13.1 Attributes

WireLists/TestWireList contains no attributes.

C.2.2.13.2 Child elements

WireLists/TestWireList inherits the child elements of *w:TestWireList* (*AssetWireList* and *Test*).

C.2.2.14 WireLists/WireList

Base type: *w:WireList*

Properties: isRef 0, content complex

The *WireLists/WireList* element identifies a list of connections that associate the interface pins from one XML instance document to another XML instance document (e.g., a UUTDescription XML instance document to a TestAdapter instance document).

C.2.2.14.1 Attributes

WireLists/WireList contains no attributes.

C.2.2.14.2 Child elements

WireLists/WireList inherits the child element of *w:WireList* (*Wire*).

C.2.3 Simple types

None

C.2.4 Attribute groups

None

Annex D

(normative)

ATML runtime services

D.1 Messages

As a guideline, messages and user display information used in an ATML-compliant ATS are to utilize well-formed hypertext markup language (HTML)[13] (see HTML 4.01 Specification [B10]) for representing their display information. The use of HTML allows the use of standard browser technology to display and interact with the user in a common format across platforms.

If a requirement exists where XML instance documents need to include portions of HTML and HTML documents need to include portions of other markup languages, then the use of the extensible hypertext markup language (XHTML)[14] (see XHTML 1.1 [B57]) should be considered.

D.2 Executive system service

ATML services should be described using a web services definition language (WSDL) definition. This approach allows a multitude of implementations including straight function calls, dynamic link library (DLL) calls, COM methods, common object request broker architecture (CORBA) services, or Web services.

A WSDL document will define ATML **services** as collections of network endpoints, or **ports**. In WSDL, the abstract definition of endpoints and messages is separated from the concrete network deployment or data format bindings of the endpoints and messages. This approach allows the reuse of abstract definitions: **messages**, which are abstract descriptions of the data being exchanged, and **port types**, which are abstract collections of **operations**. The concrete protocol and data format specifications for a particular port type constitute a reusable **binding**. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Hence, a WSDL document uses the following elements in the definition of ATML services:

a) **types**: a container for data type definitions using some type system (such as an XSD)

b) **message:** an abstract, typed definition of the data being communicated

c) **portType:** an abstract set of operations supported by one or more endpoints

   1) **operation:** an abstract description of an action supported by the service

d) **binding:** a concrete protocol and data format specification for a particular port type

e) **service:** a collection of related endpoints

   1) **port:** a single endpoint defined as a combination of a binding and a network address

[13] This information is given for the convenience of users of this standard and does not constitute an endorsement by the IEEE of these products. Equivalent products may be used if they can be shown to lead to the same results.
[14] This information is given for the convenience of users of this standard and does not constitute an endorsement by the IEEE of these products. Equivalent products may be used if they can be shown to lead to the same results.

D.3 Example WSDL service definition

An example of using the ATML family XML schemas would be a service that consumes ATML Test Descriptions and returns Software Interface for Maintenance Information Collection and Analysis (SIMICA) TestResults.

The ATML Test Description is an XML instance document conforming to TestDescription.xsd. The SIMICA TestResults is an XML instance document conforming to TestResults.xsd. Such a service is described in terms of its inputs and outputs; this description does not prevent the service from providing additional logging information to internal systems such as an operating system event logger.

The following example utilizes the uniform resource name (URN) naming conventions described in A.3. Run-time services are provided as part of an implementation and have been included in the following XML snippet example to demonstrate inputting and outputting messages.

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions
     xmlns=http://schemas.xmlsoap.org/wsdl/
     xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap
     xmlns:http="http://schemas.xmlsoap.org/wsdl/http"
     xmlns:xs="http://www.w3.org/2001/XMLSchema"
     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
     xmlns:td="urn:IEEE-1671.1:2009:TestDescription"
     xmlns:tr="http://www.ieee.org/atml/2007/TestResults"
     xmlns:y="RuntimeServices"
     <targetNamespace="RuntimeServices">
     <message name="testDescriptionIn">
  <part name="parameters" element="td:TestDescription" />
     </message>
     <message name="testResultsOut">
  <part name="parameters" element="tr:TestResults" />
     </message>
     <portType name="ExecutiveRuntimeSystem">
  <operation name="Execute">
    <input message="y:testDescriptionIn" />
    <output message="y:testResultsOut" />
  </operation>
     </portType>
</definitions>
```

Because this definition does not contain any specific bindings or services, it can be used as part of a Web service call or to define a traditional C/C++ (see ISO/IEC 9899:1999 [B40]) function:

```
std:string Execute( const std:string parameters );
```

The input and output strings are to be constrained to conform to the relevant ATML family.

Annex E

(informative)

Pins, ports, connectors, and wire lists in ATML

E.1 Introduction

To describe instruments, test systems, and their capabilities at the instruments' pins, ATML uses constructs called ports, pins, and connectors.

To describe how instruments, test systems, interface adapters, and UUTs are interconnected, ATML uses a construct called a wire list.

E.1.1 Pins, ports, and connectors

At first glance, the definition and use of these constructs may seem obvious, and in many ways, that assumption is true. But these items are interrelated and must be mapped both to each other and to the signal channels described by the capabilities. In addition, it is necessary to use these constructs to model the overall capabilities of an instrument.

The techniques for doing so are explained in this annex.

NOTE—ATML Capabilities is fully described in Annex F.

In ATML, a connector is a physical part of a hardware description, e.g., UUT, instrument, test station, or test adapter. Connectors can be of any type and can carry any and many types of signal: ac, dc, or radio frequency (RF); male or female; analog or digital. Connectors are often industry-standard types, but can also be defined by custom types. Connectors can contain a collection of pins definitions that identify the physical pin characteristics. Pins must be unique on connectors.

In contrast, a port is a logical entity. It describes an abstract interface that may be mapped to a signal (or collection of signals) going into or out of any hardware description, e.g., UUT, instrument, test station, or test adapter (or, when used in ATML Capabilities, a resource). When used to describe a physical interface, a port can also include any number of pin references. In this case, the pin references refer to physical pins on a connector. When used with resources to describe capabilities, ports do not include pins. Different physical ports can reference the same pins.

It is expected that a connector will have at least one port associated with it where a connector can carry many signals. Each signal corresponds with a port; in other words, a single connector (a physical object) can carry many ports (logical objects).

Ports, pins, and connectors are all defined by XML elements within the ATML Common XML schema (B.1), which is in turn utilized within the hardware descriptions, e.g., Instrument Description, Test Station Description, Test Adapter Description, and the XML schemas associated with the UUT Description standards (through their inclusion of Common.xsd).

E.1.2 Wire lists

The WireList element of the WireLists schema contains a system's interconnect information, which describes how different ATML family component standards connect. The WireList element allows for assigning interface ports from different hardware descriptions, such as from the Test Station to the Test Adapter and then from the Test Adapter to the UUT, to provide the complete path.

The Wirelists schema also includes the TestWireList element, which is a collection of signal stimulus and measurement paths associated with a subelement of a test description (e.g., test). This element supports the generation of test diagrams for a TP providing the complete signal paths from test station instruments to UUT ports. The TestWireList element has a child element called Tests, which references the Test ID from an ATML Test Description instance document. The child element AssetWireList contains the test station asset and port where the signal originates. The Wire child element is a list of all wires used in the associated test in the TP.

ATML family component standard interconnections are defined by the ATML WireList XML schema (C.2).

E.2 Overview of the base types

E.2.1 Ports

A port simply describes the fact that there is a logical interface through which a signal is going into or out of an instrument (or test station, or test adapter, or UUT). This informaiton does not include a physical description that describes how that signal is transmitted; that information is the function of a connector. The existence of a port merely states that a logical interface and some signal(s) exist.

E.2.1.1 Two types of <Port> elements

An examination of the ATML Common XML schema (see B.1) will reveal that there are actually two types of `<Port>` elements, the second of which is derived from the first.

The basic `<Port>` description does not include pin references. This port is used primarily for ATML Capabilities descriptions and is not associated with physical connectors. The second `<Port>` is defined and used within the `<PhysicalInterface>` element. There, pin references are added to the base `<Port>` element so that ports can be mapped onto physical connectors via their pins.

E.2.1.2 Parts of a port

The base `<Port>` is a simple element that includes only the small amount of information needed to describe the type of signal that is carried through the port: Name, Direction, and Type.

NOTE—One common mistake is to consider a port equivalent to a pin. A simple test is to ask, "does the signal flow through the port?" If it does not, the entity is probably a pin. As an example, the HI pin on a DMM banana jack is not a port because a signal also flows through the LO pin.

E.2.1.2.1 Name

The name is simply an arbitrary string. Intended to be descriptive, this element distinguishes one port from another.

E.2.1.2.2 Direction

The <PortDirection> attribute simply specifies whether the port is an input port, an output port, or a bidirectional port.

The <PortDirection> attribute is optional. When a port describes instrument capabilities, resources, or switching, this element is not used (and is not needed).

E.2.1.2.3 Type

The <PortType> attribute describes the type of signal that is carried by the port. The predefined possible values (all of string type) are "Ground", "Analog", "Digital", "Power", "Optical", and "Software".

Like the <PortDirection> attribute, the <PortType> attribute is optional. It is not used when describing instrument capabilities, resources, or switching.

E.2.1.2.3.1 Software ports

Although the definitions of most of the predefined port types are obvious, the software port type is a little mysterious. This type of port describes signals that travel through a software interface of some type and is commonly associated with measurements (as opposed to signal generation). For instance, the trace on an oscilloscope screen is not available as an electrical signal on a physical connector. It can be viewed on the oscilloscope's front panel screen; but if the data contained in the trace are to be available to test system software, they must be retrieved via software. A software port would then be declared to exist on the instrument, and the instrument's capability to deliver trace data would then be mapped to this software port.

E.2.1.2.4 Extension element

The <Extension> element is available to allow users to define custom port types of their own or to add extra information to the definition of a port.

E.2.1.2.5 Connector pins

When used within a <PhysicalInterface> element, the <Port> element may include any number of pin references using the <ConnectorPin> elements. This option allows the port to be logically mapped to the pins of a connector. <ConnectorPin> elements are described in E.2.3.

E.2.2 Connectors

Since a connector refers to a physical item, the information included in the <Connector> element includes data that describe both the physical type of connector and its location on a piece of hardware such as an instrument.

NOTE 1—The <Connector> type is derived from the <ItemDescription> type. An <ItemDescription> includes several optional fields that can be used to describe any physical item, e.g., the manufacturer's name, a serial number, or a model number. These fields are not described within this annex as they are self-explanatory.

NOTE 2—Both the <Connector> definition and the <Port> definition include a list of pins. In <Connector>, they are a pin definition and called <Pin>; and in <Port>, they are a pin reference and called <ConnectorPin>. Although this setup may appear redundant at first glance, the intended use of these elements is different. In a pin definition <Connector>, the list of pins is designed to capture physical information about each pin: the name, description, and perhaps the part number. In a pin reference <Port>, the list of pins lists the pins that carry a specific signal. In both cases, the description of a pin includes an ID, and that ID should be the same in both locations.

E.2.2.1 Pins

This element contains an optional list of pin definitions that make up the connector. Each pin definition has a required ID and an optional name that can be used to describe the use of the pin. In addition, each pin contains a <Definition> element that is intended to capture specific information like part numbers. If pins are described in a <Connector>, the <pinID> value used in <Port> should match the corresponding <ID> in the <Connector> description.

E.2.2.2 ID

The ID of a connector is an arbitrary string that identifies the connector. This same identifier should be used when defining each pin on the connector.

E.2.2.3 Location (front or back)

This enumeration, with values of "front" and "back", specifies the connector's location on an instrument.

E.2.2.4 Type

This string specifies the type of the connector (e.g., Bayonet Neill Concelman (BNC), subminiature A (SMA), or 25-pin D connector), the sex of the connector, and/or the connector's part number. Any arbitrary string can be entered in this field, and this flexibility allows users to specify custom connector types. However, if an industry-standard connector is used, it is expected that the standard name will be used here.

E.2.2.5 Mating connector type

This optional field, which contains an arbitrary string, specifies a mating connector, i.e., the type of connector that mates to the <Connector> that is being described. Like the connector's Type, this field may contain a generic connector description or a specific part number.

E.2.3 Pins

A <ConnectorPin> element is a very simple item that contains just enough information to describe a pins reference or location. A <ConnectorPin> element includes an ID for each pin and also the ID of the connector that includes the pin. This information creates a logical mapping between the port and the connector.

The use of these elements is straightforward. The ID of the connector should match the ID contained in the connector's description in the <Connector> element. The pin ID should contain a string that defines the specific pin within the given connector. The pin ID is simply a string and can be given a name like "Pin 12-9" or "External Clock Input". If a pin is also described in the <Connector> element, the ID and pinID in both locations should match.

E.2.3.1 Pins in IEEE Std 1641 [B29]

ATML Capabilities descriptions utilize IEEE 1641-based signal descriptions, which define their own connectors (i.e., collection of pins). Because eventually there is a need to correspond the pins defined in ATML and by IEEE Std 1641, it is necessary to understand a little about the treatment of pins in IEEE Std 1641.

Where IEEE Std 1641 requires a compound pin name such as TwoWire hi="J1-1", the ConnectorPins compound name shall be used. The compound name cannot contain spaces, commas, or semicolons. Within ATML, these free-form descriptive names can be entered in the connectorID/name attribute, e.g., ID="GND" name="shielded ground".

When creating ATML Capability descriptions, signal connections are mapped to instrument pins in the same order as in IEEE Std 1641. This order is illustrated in the example shown in E.3.1. Connections defined in IEEE Std 1641 have a specific, defined order, where ground pins can be either explicitly or implicitly defined.

IEEE Std 1641-2004 includes the following description:

> The number of parallel connections is specified by the <**channelWidth**> of the signal. Each pin name is associated with its corresponding channel. Ground or signal return connections may be added after the active channel pins. The last ground pin will be used to return any remaining channels without a specified signal return pin. If no return pin is specified, a common ground return is assumed.

E.3 Using ports, pins, and connectors together

In ATML, the <Port>, <ConnectorPin>, and <Connector> types are used together in two different ways. The first is to describe the physical characteristics of an instrument's interface to the outside world. The second is to describe the instrument's capabilities.

E.3.1 <PhysicalInterface>

The <PhysicalInterface> type describes all of the connectors on a piece of hardware, together with their pins plus the ports that represent the signals that travel through those connectors.

The <HardwareItemDescription> type (defined within the HardwareCommon XML schema in B.2) includes an element of type <PhysicalInterface> as part of an overall description of any type of hardware. In <HardwareItemDescription>, this element is called <Interface>.

The <PhysicalInterface> type includes two lists: a list of connectors and a list of ports. Each port includes a list of pin references; and each pin includes the ID of the connector of which it is a part.

As a simple example, consider a connector that has 6 pins. Of these, two are connected to ground, two are connected to dc power, and two carry a differential signal. The <PhysicalInterface> element (remember, this element is called <Interface> within a <HardwareItemDescription>) for this connector might be as follows:

```
<hc:Interface>
  <c:Ports>
    <c:Port name="Gnd" direction="Input" type="Ground">
      <c:ConnectorPins>
        <c:ConnectorPin connectorID="PL1" pinID="Gnd1" />
        <c:ConnectorPin connectorID="PL1" pinID="Gnd2" />
      </c:ConnectorPins>
    </c:Port>
    <c:Port name="Pwr" direction="Input" type="Power">
      <c:ConnectorPins>
        <c:ConnectorPin connectorID="PL1" pinID="Pwr1" />
        <c:ConnectorPin connectorID="PL1" pinID="Pwr2" />
      </c:ConnectorPins>
    </c:Port>
    <c:Port name="DiffIn" direction="Input" type="Analog">
      <c:ConnectorPins>
        <c:ConnectorPin connectorID="PL1" pinID="V+" />
        <c:ConnectorPin connectorID="PL1" pinID="V-" />
      </c:ConnectorPins>
    </c:Port>
  </c:Ports>
  <c:Connectors>
    <c:Connector ID="PL1" name="Input Connector" location="Front" type="6-pin">
      <c:Identification>
        <c:ModelName>XYZ123</c:ModelName>
      </c:Identification>
      <!-- …other data from <ItemDescription> as needed… -->
    </c:Connector>
    <c:Pins>
      <c:Pin ID="Gnd1"/>
      <c:Pin ID="Gnd2"/>
      <c:Pin ID="Pwr1"/>
      <c:Pin ID="Pwr2"/>
      <c:Pin ID="V+"/>
      <c:Pin ID="V-"/>
    </c:Pins
  </c:Connectors>
</hc:Interface>
```

This example contains almost all of the information needed to describe the physical interface to an instrument. It gives the name, location, and type of the connector on the instrument's front panel; it gives the name of each pin and optionally any hardware pin characteristics; and it separates the pin references into distinct signal ports. In this case, one port is grounded, and the other carries dc power. The third port carries some type of signal. The only remaining question is exactly what kind of signal can be carried on this port. The answer lies in ATML Capabilities, which describes exactly what kind of signals an instrument is capable of delivering. An example of this is included in E.4.1.

E.3.2 <Interface>

The ATML Common XML schema (B.1) defines an element type called <Interface>. This type is not to be confused with the <Interface> child element of the <HardwareItemDescription> element. Instead, this is a distinct type of its own that is used elsewhere in ATML family XML schemas.

The <Interface> type simply contains a list of ports. These ports are intended to be logical ports, used to describe features and capabilities of an instrument or resource but not the physical real-world signal connections that a physical interface provides. The <Interface> type does not include the extra information (like pins and connectors) that is included in the <PhysicalInterface> type. It is used in a number of places within the ATML family of standards:

a) Used by <Resource> in HardwareCommon for ATML Capabilities use

b) Used by <Capability> in HardwareCommon for ATML Capabilities use

c) Used by <Switch> in HardwareCommon to define switch capabilities

d) Used by <FacilitiesRequirements> in HardwareCommon to define other interfaces (except power connections)

E.4 Ports, pins, and capabilities

The feature of ATML known as Capabilities is described in Annex F briefly; ATML Capabilities allows instruments to be described as a collection of resources. Each resource has one or more ports. Separately, the signals (or measurements) that the instrument can deliver are described as a list of IEEE 1641-based signal descriptions. Each of these signals also has one or more ports. Through the use of these ports, the signals are mapped onto the resources to define the capabilities of each resource.

Although the resources defined by ATML Capabilities are logical entities, they define the real-world signals that the instrument can deliver. Therefore, each resource must ultimately be mapped to a physical signal connector on the instrument with a <NetworkList> element.

Briefly, this mapping is accomplished through the use of <NetworkList> elements by using their port names within a <Node><Path> pair. Each <Resource> element has a <Port>; each <Port> has a name. If the name of the resources <Port> is mapped to the name of a physical instrument <Port>, then that mapping defines a connection. The instrument's <Port> includes all of the physical pin references with a reference to the connector.

```
<hc:NetworkList>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="1"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
</hc:NetworkList>
```

This concept is best described in terms of an example. E.4.1 and E.4.2 include two examples: the first, simple, and the second, a little more complex.

E.4.1 Simple example: a simple sine wave

Consider the case of a simple signal source that can generate only a sine wave. Described in terms of ATML, this source has one connector with one pin, a single resource, and a single signal capability.

Signals are described using the <Capabilities> element, which is documented fully in C.1. For this case, a simple signal source might have a <Capabilities> description like this example:

```
<inst:Capabilities>
  <hc:Capability name="sinewave">
    <hc:Interface>
      <c:Ports>
        <c:Port name="sineWave" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="sinewaveSignal" Out="sineWave">
        <std:Sinusoid
name="sineWave"
frequency="10kHz range 1kHz to 10MHz errlmt 0.1Hz res 1Hz"
amplitude="trms 1V range 1uV to 1V errlmt 0.1% range 1V to 10V errlmt 1%"/>
      </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
```

```
</inst:Capabilities>
```

The instrument would likewise have a <Resources> description that includes only a single one-port resource:

```
<inst:Resources>
  <hc:Resource name="Resource_1">
    <hc:Interface>
      <c:Ports>
        <c:Port name="P1"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
</inst:Resources>
```

These two items would be mapped together through the use of a <CapabilityMap> element. The <CapabilityMap> will not be described here in detail, but it is an element that defines the mapping between the signal and the resource. For this example, it would map the signals output port sineWave to the resources port P1; in other words, the resource is capable of delivering the given signal. The use of the mapping elements <CapabilityMap> and <NetworkList> allow ATML to map physical and logical ports and resource and capability ports without the need to mandate another standard or leave the ATML common types.

In addition, the instrument's physical output connections would be described in an <PhysicalInterface> element that lists a single port, a single pin, and a single connector:

```
<hc:Interface>
  <c:Ports>
    <c:Port name="P1" direction="Input" type="Analog">
      <c:ConnectorPins>
        <c:ConnectorPin connectorID="RF" pinID="Out" />
      </c:ConnectorPins>
    </c:Port>
  </c:Ports>

  <c:Connectors>
    <c:Connector name="RF Output" ID="RF" location="Front" type="6-pin">
      <c:Identification designator="RFConn1">
        <c:ModelName>XYZ123</c:ModelName>
      </c:Identification>
      <!-- …other data from <ItemDescription> as needed… -->
    </c:Connector>
  </c:Connectors>
</hc:Interface>
```

NOTE—The name of the physical port "P1" matches the name of the logical resource port. This match does not, however, indicate any relationship between the signal delivered by resource "Resource_1" and the physical port "P1", on connector pin "Out" in connector "RF" (e.g., RF-Out).

To map the two port names (physical and resource) together, a <NetworkList> element is used where the <Path> element contents can be any unique XPath expression that evaluates to a single item:

```
<hc:NetworkList>
    <hc:Network>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/id:Resources/hc:Resource[@name="Resource_1"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
    </hc:Network>
  </hc:NetworkList>
```

E.4.2 Three-phase wye power source

Examples like the one in E.4.1 are very common in test systems, particularly for RF instruments that carry only a single signal. But more complicated signals may utilize more than one pin, and the mapping from <Capability> to <Connector> becomes more complicated.

Consider the case of a three-phase wye power source. This case requires a more complicated signal description. The signal description itself has only a single port, but it requires multiple pins. It is necessary, then, to map the various parts of the signal description to the physical pins on the instrument's connector.

To illustrate, the signal description for three-phase power would be as follows:

```
<inst:Capabilities>
  <hc:Capability name="ThreePhaseWye" >
    <hc:Interface>
      <c:Ports>
        <c:Port name="TPW" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription
      xmlns:tpw="urn:IEEE-1671:2010:Examples/ThreePhaseWye">
      <std:Signal Out="TPW" >
        <tpw:ThreePhaseWye amplitude="trms 115V +-10%" frequency="50Hz" />
      </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
</inst:Capabilities>
```

This example references an IEEE 1641 TSF library element called ThreePhaseWye. This TSF element includes a connection for all three phases (channels) of the power signal plus the neutral (common ground), with connections (pins) respectively named A, B, C and N. (When using a TSF element, these pin names are not visible in the ATML TSF source; the user must refer to IEEE Std 1641 [B29]. In other cases, the various pins may be defined directly within the <SignalDescription> element.)

This IEEE 1641 signal description, therefore, defines two distinct things:

a) Three-phase signal consisting of three coherent channels and a common neutral.

b) A port that can be mapped only to a connector that has three signal pins plus a common neutral ground. In this example, the port could not be directly mapped onto a six-pin connector, each with its own signal-return pin pair; a common return pin is required. (Of course, the connector may have other pins that are not related to this port.)

This signal must be mapped to a simple resource. This resource only has one port:

```
<inst:Resources>
  <hc:Resource name="ThreePW">
    <hc:Interface>
      <c:Ports>
        <c:Port name="TPW"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
</inst:Resources>
```

NOTE—This resource has only a single port, which is matched to the single port of the signal description, even though the signal actually requires multiple conductors. To match, the instrument's physical interface would be defined as follows:

```
<hc:Interface>
  <c:Ports>
    <c:Port name="TPW" direction="Output" type="Power" >
      <c:ConnectorPins>
        <c:ConnectorPin connectorID="PL1" pinID="A" />
```

```
        <c:ConnectorPin connectorID="PL1" pinID="B" />
        <c:ConnectorPin connectorID="PL1" pinID="C" />
        <c:ConnectorPin connectorID="PL1" pinID="N" />
      </c:ConnectorPins>
    </c:Port>
  </c:Ports>

  <c:Connectors>
    <c:Connector ID="PL1" location="Front" type="ThreePhaseWye">
      <c:Identification>
        <c:ModelName>KL1m-1</c:ModelName>
      </c:Identification>
    </c:Connector>
  </c:Connectors>
</hc:Interface>
```

With a NetworkList mapping the two “ThreePhaseWye” ports together:

```
<hc:NetworkList>
    <hc:Network>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="TPW"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/id:Resources/hc:Resource[@name="ThreePW"]/
          hc:Interface/c:Ports/c:Port[@name="TPW"]
        </hc:Path>
      </hc:Node>
    </hc:Network>
  </hc:NetworkList>
```

NOTE—The `pinID` attribute of each pin is mapped to the corresponding part of the “`ThreePhaseWye`” element of the signal description.

In this example, the pins have the same names as the corresponding outputs in the “`ThreePhaseWye`” element. However, the names are not important. The pins are mapped by the order in which they appear, not by name. This order dependence is necessary because signal descriptions are often contained in external libraries. These libraries contain fixed signal names but are used in many different instrument descriptions. The instruments themselves may have very different pin names; therefore, it may be impractical to map signals to pins by using the names. In the example, the “`ThreePhaseWye`” signal is a three-channel signal where Pin A is mapped onto the channel 1 signal, Pin B is mapped onto the channel 2 signal, Pin C is mapped onto the channel 3 signal, and all signals use a common return pin N.

E.5 Wire lists

The ATML WireList XML schema supports the description of the interconnections between ATML family component standards utilized within a particular ATS implementation.

Using Figure E.1 as a reference, wire lists may consist of per-test information (e.g., for test 100, cable W1 P1 is connected to interface test adapter J1) as well as overlying static information pertaining to more than just this test (e.g., the test fixture safety ground GS1 is connected to the ATE station ground lug E1 and to the test fixture safety ground lug E1).

`<WireList>` elements are analogous to the `<NetworkList>` elements that are used elsewhere in the ATML architecture, but they are used to “connect together” ports or connectors that are defined in completely different ATML instance documents. The WireList schema, then, is the place where everything comes together—the test station, the UUT, the adapters—to form a completely defined test setup.

Note that a `<WireList>` does not include electrical path characteristics. A `<WireList>` defines only connections. If electrical path characteristics must be captured, then a Test Adapter instance should be used. As a very simple example, consider the use of an RF cable that is connected from the test station to the UUT. This cable may have some loss. To describe the cable, create a Test Adapter instance file. The Test Adapter description can describe the cable's electrical characteristics through the use of a `<Path>` element and will naturally include a pair of `<Port>` elements. This approach completely describes the cable. The cable's ports can then be referenced in a `<WireList>` that defines the places to which the cable is connected.

E.5.1 UUT and ATS interconnections and XML schemas

Consider the case of executing UUT performance tests on a particular ATS.

Figure E.1 depicts an example of the type of interconnections that may be required to execute the performance tests of the UUT.

- The UUT contains two connectors (P1A and P1B). These connectors and pins should be defined within a ATML UUT Description.
- The Test Fixture, ITA, MIL-STD-1553 Cable, W1, W2, GS1, and Facility Power Cable connectors and pins should be defined within a ATML Test Adapter Description.
- The ATE I/O connectors and pins should be defined within a ATML Test Station Description.

The **assembly of these three distinct XML instance documents** together, in a manner that reflects the connections depicted in Figure E.1, is accomplished by an ATML WireList document.

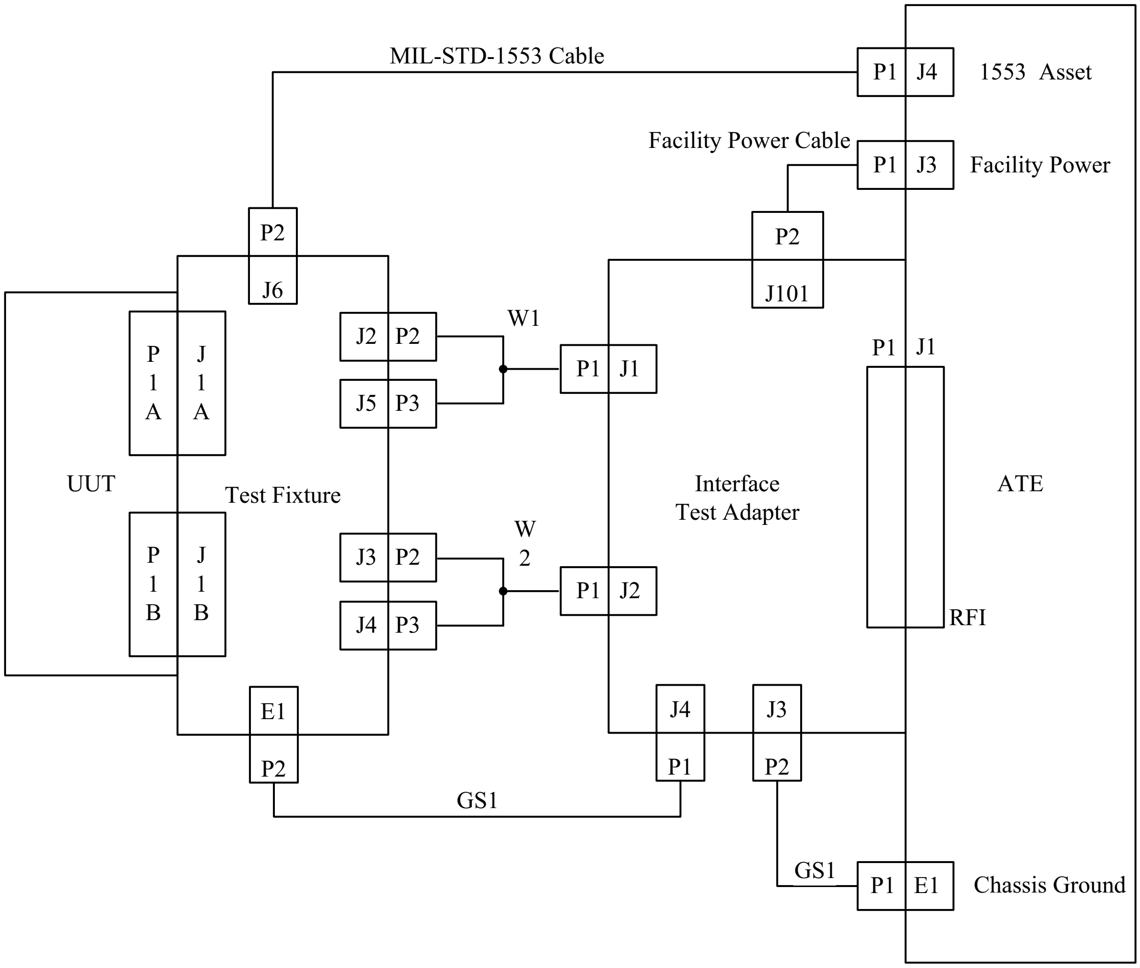


Figure E.1—Example UUT test-oriented cabling

E.5.2 Simple example: a wire list for a interface device signature test

Consider the case of a simple interface device signature test (e.g., a resistance measurement using a `DMM`). The `DMM` is available at the station interface connector and pin `P3-1A` and `P3-1B`; the interface device signature resistor is available at connector and pins `P9-67C(E5)` and `P9-68C(E4)`. The `DMM` is wired through ATE station switching and, therefore, can be connected to the interface device signature resistor for this test (test `0500`).

For this case, a wire list instance document might have a description like this example:

```
<?xml version="1.0" encoding="UTF-8"?>
<w:WireLists uuid="11111111111111111111111111111111"
xmlns:hc="urn:IEEE-1671:2010:HardwareCommon"
xmlns:w="urn:IEEE-1671:2010:WireLists"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="urn:IEEE-1671:2010:WireLists WireLists.xsd">
  <w:Items>
    <w:Item ID="ts1" uuid="33333333333333333333333333333333"/>
    <w:Item ID="A101" uuid="44444444444444444444444444444444"/>
  </w:Items>
  <w:TestDescription ID="tps1" uuid="22222222222222222222222222222222" />
  <w:WireList>
    <w:Wire>
      <hc:Node>
        <hc:Path documentId="A101">
```

```
        /tad:TestAdapterDescription/hc:Interface/c:Ports/c:Port[@name="A101:P3-1A"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path documentId="ts1">
        /ts:TestStationDescription/hc:Interface/c:Ports/c:Port/@name="GPI: DMM HI"
      </hc:Path>
    </hc:Node>
  </w:Wire>
  <w:Wire>
    <hc:Node>
      <hc:Path documentId="A101">
        /tad:TestAdapterDescription/hc:Interface/c:Ports/c:Port[@name="A101:P3-1B"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path documentId="ts1">
        /ts:TestStationDescription/hc:Interface/c:Ports/c:Port/@name="GPI: DMM LO"
      </hc:Path>
    </hc:Node>
  </w:Wire>
</w:WireList>
<w:TestWireList>
  <w:Test>
    <hc:Path documentId="td1">
      /td:TestDescription/td:DetailedTestInformation/td:Actions/td:Action[@name="0500"]
    </hc:Path>
    <hc:Description>
      *****************************************************************
      *                      PREPERFORMANCE TEST                    *
      *                                                               *
      * T0500:   ID SIGNATURE TEST                                    *
      *                                                               *
      * PURPOSE: Verify proper connection of the Interface Device    *
      *          to the station interface panel. A resistance of     *
      *          2550 ohms will be measured at pins P9-67C(E5)       *
      *          and P9-68C(E4) using the Digital Multimeter (DMM).  *
      *          Limits are: 2250 to 2750 ohms                       *
      *                                                               *
      *****************************************************************
    </hc:Description>
  </w:Test>
  <w:AssetWireList>
    <w:Asset>
      <hc:Path documentId="ts1">
        /ts:TestStationDescription/hc:Interface/c:Ports/c:Port/@name="GPI: DMM HI"
      </hc:Path>
    </w:Asset>
    <w:Wire>
      <hc:Node>
        <hc:Path documentId="A101">
          /tad:TestAdapterDescription/hc:Interface/c:Ports/c:Port[@name="P3-1A"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path documentId="A101">
          /tad:TestAdapterDescription/hc:Interface/c:Ports/c:Port[@name="P9-67C"]
        </hc:Path>
      </hc:Node>
    </w:Wire>
    <w:Wire>
      <hc:Node>
        <hc:Path documentId="A101">
          /tad:TestAdapterDescription/hc:Interface/c:Ports/c:Port[@name="P9-67A"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path documentId="A101">
          /tad:TestAdapterDescription/hc:Interface/c:Ports/c:Port[@name="E5"]
        </hc:Path>
      </hc:Node>
    </w:Wire>
  </w:AssetWireList>
  <w:AssetWireList>
    <w:Asset>
      <hc:Path documentId="ts1">
        /ts:TestStationDescription/hc:Interface/c:Ports/c:Port/@name="GPI: DMM LO"
      </hc:Path>
    </w:Asset>
    <w:Wire>
      <hc:Node>
```

```
          <hc:Path documentId="A101">
            /tad:TestAdapterDescription/hc:Interface/c:Ports/c:Port[@name="P3-1B"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path documentId="A101">
            /tad:TestAdapterDescription/hc:Interface/c:Ports/c:Port[@name="P9-68C"]
          </hc:Path>
        </hc:Node>
      </w:Wire>
      <w:Wire>
        <hc:Node>
          <hc:Path documentId="A101">
            /tad:TestAdapterDescription/hc:Interface/c:Ports/c:Port[@name="P9-68A"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path documentId="A101">
            /tad:TestAdapterDescription/hc:Interface/c:Ports/c:Port[@name="E4"]
          </hc:Path>
        </hc:Node>
      </w:Wire>
    </w:AssetWireList>
  </w:TestWireList>
</w:WireLists>
```

Annex F

(informative)

ATML capabilities

F.1 Introduction

During the development of the ATML family of standards, the development team realized that the ATML family XML schemas must (collectively) include enough information to allow cleverly written software to determine whether a given test system can run a given test. In total, this goal requires data that specify the needs of the test, the abilities of the instruments in the test system, the topology of the test system itself, and the measurement errors introduced by everything.

F.1.1 Background

An effort began with the goal of guaranteeing that the necessary information was contained in the ATML family of standards. This new effort required a name, and for lack of any better ideas the effort became known as **Capabilities**. This name stretches the strict definition of the term somewhat since ATML Capabilities subsumes both hardware capabilities and test requirements. However, the name stuck, and it should be understood that **the term Capabilities in ATML refers to something more than the standard dictionary definition of the term would lead one to believe**.

The AUTOTESTCON 2007 paper “ATML Capabilities Explained” [B9] provides, through the use of simple examples and use cases, a high-level description of the major concepts of Capabilities and the ways in which they are used to describe instrument performance and capability.

F.1.2 Purpose

The ATML Capabilities effort is driven by specific use cases that are described in F.1.3. In general, it is sufficient to say that ATML Capabilities allows tests to be mapped onto instruments (and test systems) in a way that makes it possible to tell whether a test system is able to execute a given test.

Although this concept is simple, it becomes more complicated when the full range of test system components and variations thereof are considered. Important points to remember are as follows:

a) The inherent capability of an instrument is not by itself sufficient information. It is **the capability that is available to the user** that is important. This concept includes not only the instrument itself but also the instrument software driver and any software “wrappers” in the system that might constrain the use of the instrument. For instance, many IVI drivers do not expose the full functionality of their corresponding instrument. ATML Capabilities must allow sufficient flexibility to allow differing levels of capabilities to be defined according to the situation at hand.

b) Some instruments are composite instruments; in other words, they are made up of several distinct hardware modules. SI is perhaps the most common example of composite instruments. In these cases, the capability of the instrument can be determined only at the system level. ATML Capabilities descriptions must, therefore, be applicable to both the instrument and the test system itself.

It is important to note that the ATML Capabilities includes the accuracy requirements of a test. Knowing whether an instrument is able to measure an analog signal, for instance, is not enough information. The instrument must be able to measure the signal with sufficient accuracy or the measurement will not be useful. Thus, ATML must also include enough information to allow metrology calculations to be made. This level of detail is not required by ATML, but it is possible for it to be included.

F.1.3 Use cases

The ATML Capabilities effort was originally driven by two very specific use cases:

— Allow system software developers to implement resource management systems that are capable of automatically allocating available system resources to a given set of test requirements.

— Allow for the creation of a system design tool that, using a database of instrument capabilities, can be used to choose the instruments that will be used for a given test (or to design a test system that will be capable of running the test).

During development, the following additional use cases were added. In general, these use cases will automatically be satisfied if the original pair of use cases is satisfied:

— Verify the tester configuration at runtime to determine whether the required instruments, signal paths, and other resources are present.

— Allow for runtime switch path allocation.

— Support the ability to map the capabilities of an old instrument to a new instrument.

F.1.4 Requirements

The use cases listed in F.1.3 led to the following short (but important) list of requirements for ATML Capabilities:

a) Must describe capabilities in a way that can be used for instruments, test stations, UUTs, and tests.

b) Must be able to use the descriptions to allocate resources needed for a test.

c) Must be able to describe instruments that supply signals, instruments that measure signals, and instruments that condition signals.

d) Must include the measurement uncertainty needed to support a required test accuracy ratio (TAR) for each test.

e) Must include uncertainty of all measurements and generated signal parameters.

f) Must include measurement effects (through signal paths, switches, amplifiers, etc.) that will cause test signals to degrade in system descriptions.

g) Must include timing requirements.

h) Must support parallel/simultaneous operations and complex timing relationships.

i) Must support test platform independence.

j) Must include information in a way that supports various implementations.

k) Must be machine-readable.

F.1.5 Design goals

In addition, the following design goals were agreed upon:

a) Should be scalable and easy to maintain.

b) Should not be more difficult to create capability descriptions than it inherently already is.

c) Should be human-readable.

d) Should be compact.

e) Should be able to filter capability descriptions to match the intended use of a test system.

F.2 Overview

For ATML Capabilities to be useful, it must enable the easy comparison of the needs of a test to the abilities of a test system. Obviously, the test and the test system must be described using a common language. The ATML development team chose to rely on IEEE Std 1641 [B29].

F.2.1 Reliance on IEEE Std 1641 [B29]

Briefly, IEEE Std 1641 defines a method for describing signals and their parameters. By allowing parameters to be unknown values, IEEE Std 1641 also allows measurements to be described. (In the parlance of IEEE Std 1641, the term *signals* can refer to either a signal that is to be generated or a measurement of a signal. In the context of this standard, this description is a reasonable stretch of the term *signal* because IEEE Std 1641 describes signals, events, signal conditioning, and measurements in the same way.)

By including parameters, IEEE Std 1641 allows a single signal definition to span a wide range of actual signals. For instance, a simple sine wave generator would be described in IEEE Std 1641 constructs with a single sinusoidal function and a list of parameters (and limits) that describe the total frequency and power range that the generator can supply. In this case, a single IEEE 1641 signal description would be sufficient to describe every type of signal that the signal generator could provide, for example:

```
<Sinusoid name="sineWave"
frequency="10kHz range 1kHz to 10MHz errlmt 0.1Hz res 1Hz"
amplitude="trms 1V range 1uV to 1V errlmt 0.1% range 1V to 10V errlmt 1%"/>
```

For more complicated instruments, IEEE Std 1641 defines basic mathematical constructs that define signals and also allows those constructs to be assembled into libraries of more complex signals.

IEEE Std 1641 defines signals in a language-independent way. It allows for ATE-oriented languages to carry the specific syntax that defines a signal. For example, a signal can be described in either XML or ATLAS and still fully conform to IEEE Std 1641. For the purposes of this annex, XML will be used as the ATE-oriented language.

F.2.2 Information needed to implement capabilities descriptions

In order to satisfy the requirements of ATML Capabilities, the various ATML family XML schemas must allow for the specification of the following information:

a) Descriptions of signals that must be generated.

b) The relative timing of those signals.

c) The location at which the signals are to be applied.

d) The required measurement uncertainty.

e) The topology of the test system (including test adapters, all switchable signal paths, and all fixed signal paths).

f) Information about signal transmission path degradation (e.g., losses and VSWR).

g) A description of the types of signals that can be generated by each instrument in the test system, including uncertainties.

h) A description of the types of measurements that can be executed by each instrument in the test system, including uncertainties.

There is no single ATML family XML schema that requires all of this information to be specified in an associated XML instance document. All of this information is addressed within all the ATML family XML schemas.

Additionally, a simple XML schema, called Capabilities, has been defined as a part of this standard. This XML schema is used to collect libraries of reuseable capabilities definitions, and its use is optional. The use of the Capabilities XML schema is described in F.3.2.2, and the full XML schema is documented in C.1.

The remainder of this annex describes the collective ATML family XML schemas that together make up ATML Capabilities.

F.2.2.1 Some information is optional

There is no requirement that all of the information listed in F.2.2 be included in every ATML family XML instance document because many ATSs will have no need for it. Within the ATML family of standards, some elements are required while some are optional. Likewise, ATML Capabilities does not require all possible information to be included in all ATML XML instance documents. The system integrator is free to exclude any optional information that is not relevant to the problem at hand.

For instance, some ATSs will have no need to calculate measurement uncertainties at runtime. Those systems, therefore, have little need for measurement uncertainties or analog transmission path characterizations since the relevant performance parameters will have been calculated in advance. In this type of case, unneeded information can simply be omitted from the ATML XML instance documents that describe the test system component it supports.

The requirement, therefore, is that ATML Capabilities must be able to represent this information, but not that all users using ATML Capabilities will include the information. ATML is an information exchange standard; it mandates the format through which information is to be exchanged but not what specific information is to be exchanged.

F.2.2.2 Instruments need not be completely described

Some instruments are simple to describe. For instance, a dc power supply can create only a small number of signals (i.e., a dc voltage within certain current and voltage limits). Such instruments are easily described using IEEE 1641 constructs, and their descriptions should then become a part of their ATML Instrument Description document.

Other instruments are very difficult to describe. Modern instruments such as flexible RF signal sources, spectrum analyzers with built-in measurement applications, or ARBs are capable of generating thousands of different types of signals and measurements. While IEEE Std 1641 [B29] provides the constructs to completely describe these signals and measurements, the sheer size of the task makes it difficult to accomplish. In addition, **there is generally no reason for these instruments to be completely described**. In any given ATS, only a small number of signals are of interest and used, and it is only those signals that must be described in the associated XML instance documents.[15] To solve the problem, the ATML Instrument Description allows instruments to be only partially described. IEEE 1641-based signal descriptions may cover only the types of signals that are of interest to a specific test system; however, in this case, the signal descriptions are to be placed in a different location. The ATML Instrument Description includes two different XML schemas that are used to describe instruments: the Instrument Description XML schema and the Instrument Instance XML schema. The former is used to describe broad classes of instruments (usually by model number). The latter is used to describe a specific instrument in a specific ATS (usually by serial number). If instruments are not fully described, the signal descriptions are placed in the ATML Instrument Instance document, not the ATML Instrument Description instance document. Because the ATML Instrument Instance document is specific to a particular instrument (and hence to its use within a particular test system), this place is appropriate to put partial signal descriptions (which are specific to the test system).

If an instrument is to be moved from one ATS to another dissimilar ATS and used there for an entirely different purpose, then its ATML Instrument Instance document will have to be changed. This requirement is again consistent with the intended use of the ATML Instrument Instance document, which is allowed to change from time to time when instrument options are added or deleted or when the instrument is recalibrated. The ATML Instrument Description document is intended to describe instrument models, not specific instruments, and should be provided by the manufacturer and remain unchanged.

F.2.3 Information is distributed

As previously stated, the information needed for ATML Capabilities is distributed among several different ATML family XML schemas. This section describes the key sections of those XML schemas and their associated XML instance documents.

F.2.3.1 Test description

The ATML Test Description XML schema includes the ability to describe the requirements of each test. Portions of this information could be compared to the capability of an instrument in the test system for example.

In addition to signal descriptions, the ATML Test Description document will include information about the timing and required uncertainties of all relevant signals. This information is detailed in F.5.

F.2.3.2 Instrument description

The ATML Instrument Description XML schema includes the ability to describe information about the types of signals that an instrument can produce (or measure) and the uncertainties involved. It also provides for the description of the ports of each instrument, tells the user which signals can be routed to each port, and gives the physical location of each port. Together with the system topology information provided for by the ATML Test Station Description XML schemas (see F.2.3.4), the user can determine whether a signal can be delivered to any specific port on the UUT.

---

[15] Therefore, it is expected that system integrators, rather than instrument vendors, will often be responsible for creating ATML Capabilities descriptions. Although it is not feasible for an instrument vendor to completely describe many instruments, it is feasible for a system integrator to describe the subset of an instrument's capabilities that are of interest to a given test station.

Importantly, the ATML Instrument Description document also contains a listing of resources that are contained within the instrument. Resources are logical entities that map signals to ports and also specify any restrictions on signal generation or measurement. Together, the descriptions of signals, ports, and resources describe an instrument's capabilities.

ATML Instrument Description documents can contain information about either the basic instrument model or one of the options that can be installed in the basic instrument. A complete description of any individual instrument requires ATML Instrument Description documents for both the basic instrument and any installed options.

F.2.3.3 Instrument instance

The ATML Instrument Instance document contains information about a specific instrument (a single physical unit). As described in F.2.2.2, this description may also contain descriptions of signals that can be generated (or measured) by the instrument that are not in its ATML Instrument Description document.

The available capabilities of an instrument are determined by the combination of the instrument, the driver software, and the system software. Because this combination can change from one system to the next, the ATML Instrument Instance document is the container that defines which resources (which are defined in the ATML Instrument Description XML schema) can supply which capabilities (which will typically be defined within a separate file included in the ATML Instrument Description document). The ATML Instrument Instance XML schema allows for the description of mapping information for this purpose. When an instrument is moved from one test system to another, the ATML Instrument Instance document can be changed to reflect the actual available capabilities of the instrument after it has been installed in the new test system.

In addition, the ATML Instrument Instance document contains a list of all installed options. Since some options can extend the instrument's capabilities (or restrict them), this information is critical for determining a specific instrument's capabilities. Information about the options themselves is contained in the ATML Instrument Description document for each option, including any new instrument resources or ports that the option may add to the instrument. The ATML Instrument Instance document lists all of the installed options but does not include information about the effects of those options.

F.2.3.4 Test station and test adapter

Both the ATML Test Station Description and ATML Test Adapter Description XML schemas include the ATML Test Equipment XML schema (defined in B.3). The elements within the ATML Test Equipment XML schema allow information about test system topology (for either a test station or a test adapter), including the signal paths that are available (switchable or not) and the instrument ports to which the signal paths are connected to be specified. This information determines whether a given signal can be delivered to a UUT (or to any other point in the test system).

The ATML Test Equipment XML schema elements provide for the description of information that characterizes the performance of each signal path, e.g., loss and VSWR versus frequency. This information is needed to complete metrology calculations or to simply determine whether signals can be delivered with sufficient accuracy.

The ATML Test Equipment XML schema elements provide for the specification of capability descriptions that are structurally identical to instrument descriptions. Some of the capabilities of an ATS may require the use of more than one instrument, or they may restrict the use of some components and complicate resource management and capability analysis functions. For instance, a SI may require the use of more than one hardware module; and while that SI is in use, some other system capabilities (i.e., other SI) may not be available because the necessary hardware is already in use. Fortunately, it is possible to describe system capabilities and instrument capabilities in the same terms.

Finally, the ATML WireLists XML schema contains system interconnect information and allows for the description of how ATML family component standards connect. This schema allows for the assignment of interface ports from the Test Station to the Test Adapter, and then from the Test Adapter to the UUT, and provides for a complete path description.

F.3 Describing instrument capabilities

ATML Instrument Description documents that are valid to either the ATML Instrument Description XML schema or the ATML Instrument Instance XML schema describe an instrument's overall capabilities. In summary, the capabilities are described via these steps:

a) Describe the instrument interface (physical input and output ports)
b) Describe the capabilities (signals)
c) Define the resources
d) Map the capabilities to resources
e) Wire the resource interfaces to the instrument interfaces

F.3.1 Describe the instrument interface

The instrument interface is described by the `<hc:Interface>` element of the ATML Instrument Description XML schema. `<hc:Interface>` simply provides for the specification of all the physical ports on the instrument and the assignment of a name to each port.

For instance, an instrument with two ports, one on the front of the instrument and one on the back, might be as described by this XML snippet:

```
<hc:Interface>
  <c:Ports>
    <c:Port name="Front" />
    <c:Port name="Back" />
  </c:Ports>
</hc:Interface>
```

The port names are, of course, arbitrary with the restriction that each port must have a unique name. (Port names must be unique within the description of the instrument. Different instruments can have identical port names.)

F.3.2 Describe the capabilities

The `<Capabilities>` element of the ATML Instrument Description XML schema contains information about the types of signals that can be created or measured by the instrument. As described in F.2.1, the detailed description of these capabilities relies on IEEE Std 1641 [B29], with some specific extensions defined within the ATML Instrument Description XML schemas.

Each `<Capability>` has one or more ports of its own, but they are not physical ports (they are merely logical inputs and outputs that properly attach capabilities to resources that can create the capability). Signals can have more than one port, where each port must have a unique name within the description of that resource (different resources can have identical port names).

Signals are listed by this <Capabilities> element. There may be any number of defined signals. Each of these has a name, an IEEE 1641-based signal description, and one or more ports. For example, an instrument that can generate a single type of signal (a sine wave) might have a <Capabilities> XML snippet like this:

```
<inst:Capabilities>
  <hc:Capability name="sinewave">
    <hc:Interface>
      <c:Ports>
        <c:Port name="sineWave" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="sinewaveSignal" Out="sineWave">
        <std:Sinusoid
          name="sineWave"
          frequency="10kHz range 1kHz to 10MHz errlmt 0.1Hz res 1Hz"
          amplitude="trms 1V range 1uV to 1V errlmt 0.1% range 1V to 10V errlmt 1%"/>
        </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
</inst:Capabilities>
```

Likewise, a capability to measure an root-mean-square (rms) value on the “Input” might have a <Capabilities> XML snippet like this:

```
<inst:Capabilities>
  <hc:Capability name="measRMS">
    <hc:Interface>
      <c:Ports>
        <c:Port name="Input" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="RMSSignal" In="Input" Out="rmsMeas">
        <std:RMS
          name="rmsMeas"
          In="Input"
          nominal ="trms range 1uV to 10V errlmt 0.1% range 10V to 150V errlmt 1%"/>
        </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
</inst:Capabilities>
```

In capabilities descriptions, it is important to match the port names with the names of the signal channels in the IEEE 1641 signal description. This step provides the mapping of the signal inputs and outputs to the capabilities logical ports, which in turn map the capability onto resources (described in F.3.3). In this example, the signal description includes a signal channel `Out="sineWave"` that is matched, by name, to the corresponding port named `sineWave`.

F.3.2.1 Using IEEE 1641 TSF-based signals

Within IEEE Std 1641 [B29], the basic component for signal definitions is the BSC layer. The examples in F.3.2 use this layer. In addition, IEEE Std 1641 allows BSC signal definitions to be combined into more complex entities and grouped into libraries. Within IEEE Std 1641, this level is called the test signal framework (TSF). To refer to a signal definition at this level in ATML, the namespace of the TSF file must be included. Otherwise, referencing a TSF signal definition is the same as referencing a BSC definition. An example XML snippet of a signal definition that references a TSF would look like the following:

```
<inst:Capabilities>
  <hc:Capability name="ACSig">
    <hc:Interface>
      <c:Ports>
        <c:Port name="ACSignal" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
    <std:Signal name="ACSig" Out="ACSignal" xmlns:tsf716="urn:IEEE-1641:2010:STDTSFLib">
        <tsf716:AC_SIGNAL
```

```
        name="ACSignal"
        type="Voltage"
        ac_ampl="range 0V to 15V errlmt 100mV range 15V to 30V errlmt 250mV"
        dc_offset="0V range -5V to +5V errlmt 1%"
        freq="1kHz range 0.1Hz to 10MHz errlmt 0.1%"
        phase="0 rad range 0 rad to 0 rad errlmt 0.5 rad"/>
      </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
</inst:Capabilities>
```

F.3.2.2 Using external libraries of signal definitions

A <Capabilities> definition includes a list of capabilities. Within this framework, it is possible to specify either <Capability> elements (as described in F.3.2) or <CapabilitiesReference> elements. <CapabilitiesReference> elements allow the use of external libraries of signal definitions.

This definition of capabilities makes it possible for entire libraries of signal definitions to be created independently of any instrument. An ATML Instrument Description document could then simply include a reference to the library to obtain access to all of the definitions in that library. These capabilities would then be mapped to resources in the instrument. Any signal definitions that are not mapped to a resource would not be available.

A <CapabilitiesReference> is really nothing more than a UUID that refers to an external document and is used in <Capabilities> lists as in this example:

```
<inst:Capabilities>
  <hc:CapabilitiesReference uuid="{205F01E4-377F-4369-AADF-C7EF654BD15E}"/>
</inst:Capabilities>
```

This simple example would include all of the contents of the referenced file and make the capabilities described in it available to the current active instance document.

The ATML Capabilities XML schema is intended to be used as a container for libraries of capability definitions. A <CapabilitiesReference> element should refer to a ATML Capabilities document.

F.3.3 Define the resources

The ATML Instrument Description XML schema allows every instrument to contain one or more resources. A resource is a logical entity that usually (but not always) represents some internal capability of the instrument. For instance, a signal generator may have one resource: the hardware that generates signals. That hardware resource may have the ability to generate many different signals, but it is a single resource. An instrument may also have several resources.

Resources are the unit of allocation at runtime. When some source or measurement function is used, a resource is allocated. Each resource is capable of generating one or more signals, and the output of each resource is capable of being routed to one or more ports. An allocated resource fixes the signals, which may be shared by other resources, and thus constrains the functionality (capability) of other resources within the instrument.

One important aspect of resources is that resources do not have to represent real or physical parts of an instrument. Although resources can (and frequently will) correspond to a real-world instrument subsystem, in ATML they are simply logical entities that help define signal-to-port mappings. In particular, resources are useful for defining restrictions and dependencies between different capabilities in a single instrument.

Like signal capabilities, resources have a name and a port. Resources can have multiple ports, each with a unique name. These ports match the resource to the signal, but the resource ports do not have to have the same name as the signal capability ports.

Resources are listed by the <Resources> element within an ATML Instrument Description document. An instrument with a pair of two-port resources would have a <Resources> like this example:

```
<inst:Resources>
  <hc:Resource name="Resource_1">
    <hc:Interface>
      <c:Ports>
        <c:Port name="P1"/>
        <c:Port name="P2"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
  <hc:Resource name="Resource_2">
    <hc:Interface>
      <c:Ports>
        <c:Port name="P1"/>
        <c:Port name="P2"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
</inst:Resources>
```

F.3.4 Map capabilities to resources

Every instrument has one or more resources, and each resource is associated with one or more capabilities. Sometimes an instrument will have multiple resources that are associated with the same capability (e.g., a multichannel sensor will have a resource for each channel, but each resource has the same capabilities). Other times an instrument will have a single resource that can supply many different capabilities. As was described earlier, capabilities and resources are separate elements within the ATML Instrument Description XML schema. Therefore, there is a need for a mapping between capabilities and resources.

This mapping is specified by the <CapabilityMap> element. A <CapabilityMap> is a list of <Mapping> elements. Using the <Map> type, the <Mapping> element's associate resources (and their ports) can be mapped to capabilities. Each node in the <Map> specifies a resource and a port and matches them with a capability and its port. The existence of a node implies that the given resource is capable of generating or measuring the given signal.

For example, consider an instrument with a single resource and a single capability. The <CapabilityMap> for this instrument would contain only a single <Mapping> element:

```
<hc:CapabilityMap>
  <hc:Mapping>
    <hc:Map>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="sinewave"]/
          hc:Interface/c:Ports/c:Port[@name="sineWave"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
    </hc:Map>
  </hc:Mapping>
</hc:CapabilityMap>
```

The first <Node> in the capability map refers to a signal called `sinewave`, which has an output port called `sineWave`. The second node element refers to a resource called `Resource_1` with an output port `P1`. These <Node> elements are logically connected together; in other words, in this context `Resource_1` is capable of generating a `sinewave`, and the port called `P1` on `Resource_1` maps to the port called `sineWave` in the capability `sinewave`.

F.3.4.1 Specifying a single resource that is mapped to multiple capabilities

Many instruments will contain resources that can create or measure several different types of signals. These resources need to be mapped to multiple capability definitions. This mapping simply requires more capability references in the <CapabilityMap>. A single resource that is associated with two capabilities might look like this example:

```
<hc:CapabilityMap>
  <hc:Mapping>
    <hc:Map>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="measRMS"]/
          hc:Interface/c:Ports/c:Port[@name="Input"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
    </hc:Map>
  </hc:Mapping>

  <hc:Mapping>
    <hc:Map>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="sinewave"]/
          hc:Interface/c:Ports/c:Port[@name="sineWave"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
    </hc:Map>
  </hc:Mapping>
</hc:CapabilityMap>
```

In this example, each <Mapping> element refers to the same resource but a different capability. In other words, Resource_1 (through its P1 port) can either supply a sinewave signal using the sinewave capability or perform a signal-based measurement of rmsMeas using the measRMS capability (in which case, P1 would become an Input), but not perform both functions at the same time.

F.3.4.2 Defining a capability that depends on the value of a parameter

Some instrument capabilities depend on the value of a related parameter. For instance, a signal source may be able to deliver high power at low frequencies and lower power at higher frequencies, but not high power at high frequencies. This type of case is handled by defining separate signal definitions for each range and then mapping each signal definition to the same resource.

For example, consider the simple sinewave signal definition, which describes a source that operates from 1 kHz to 10 MHz with signal amplitudes from 1 uV to 1 V:

```
<inst:Capabilities>
  <hc:Capability name="sinewave">
    <hc:Interface>
      <c:Ports>
        <c:Port name="sineWave" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="sinewaveSignal" Out="sineWave">
        <std:Sinusoid
          name="sineWave"
```

```
        frequency="10kHz range 1kHz to 10MHz errlmt 0.1Hz res 1Hz"
        amplitude="trms 1V range 1uV to 1V errlmt 0.1% range 1V to 10V errlmt 1%"/>
      </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
</inst:Capabilities>
```

To expand on this example, assume that this source is also capable of operating in a frequency range up to 20 MHz, but in this higher frequency range, it is capable of delivering only a 100 mV signal. This case requires a second capability definition that is added to the first:

```
<inst:Capabilities>
  <hc:Capability name="sinewave">
    <hc:Interface>
      <c:Ports>
        <c:Port name="sineWave" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="sinewaveSignal" Out="sineWave">
        <std:Sinusoid
          name="sineWave"
          frequency="10kHz range 1kHz to 10MHz errlmt 0.1Hz res 1Hz"
          amplitude="trms 1V range 1uV to 1V errlmt 0.1% range 1V to 10V errlmt 1%"/>
        </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>

  <hc:Capability name="HFsinewave">
    <hc:Interface>
      <c:Ports>
        <c:Port name="HFsineWave" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="sinewaveSignal" Out="HFsineWave">
        <std:Sinusoid
          name="HFsineWave"
          frequency="0MHz range 10MHz to 20MHz errlmt 0.1% res 1Hz"
          amplitude="trms 1uV range 1uV to 100mV errlmt 0.1%"/>
        </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
</inst:Capabilities>
```

These two capabilities would then be mapped onto a single resource in the manner illustrated in F.3.4.1:

```
<hc:CapabilityMap>
  <hc:Mapping>
    <hc:Map>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="sinewave"]/
          hc:Interface/c:Ports/c:Port[@name="sineWave"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
    </hc:Map>
  </hc:Mapping>
  <hc:Mapping>
    <hc:Map>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="HFsinewave"]/
          hc:Interface/c:Ports/c:Port[@name="HFsineWave"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
```

```
    </hc:Map>
  </hc:Mapping>
</hc:CapabilityMap>
```

F.3.4.3 Specifying a single capability that is used by multiple resources

Often, a single capability definition will be used by multiple resources. This capability is common for multichannel instruments like oscilloscopes, which contain multiple copies of identical hardware that can accomplish the same task. These instruments are easily defined by using multiple `<Mapping>` elements, each of which refers to the same signal description but a different resource. An instrument with a single capability and two resources would have a `<Mapping>` element such as the following:

```
<hc:CapabilityMap>
  <hc:Mapping>
    <hc:Map>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="sinewave"]/
          hc:Interface/c:Ports/c:Port[@name="sineWave"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
    </hc:Map>
  </hc:Mapping>

  <hc:Mapping>
    <hc:Map>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="sinewave"]/
          hc:Interface/c:Ports/c:Port[@name="sineWave"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
    </hc:Map>
  </hc:Mapping>
</hc:CapabilityMap>
```

The two `<Mapping>` elements in this example are nearly identical, except that they refer to different resources (`Resource_1` and `Resource_2`).

F.3.4.4 Defining a capability that consumes more than one resource

Sometimes it is necessary to define a capability that uses more than one resource or that prevents the use of some resource while it uses others. Such a case can occur, for example, if a power supply is able to output low power to multiple ports but can supply high power only to a single port. In such an instrument, the high-power resource would prevent any other resources from being used.

This case is handled by using multiple mappings of the resource and capability **within the same `<Mapping>` element,** but with one resource having no capability. This step is accomplished by adding multiple `<Map>` elements to a single `<Mapping>` element. In one `<Map>` element, the capability is mapped to one of the resources. In the second `<Map>` element, the extra resource is not mapped to anything. For example, a capability named `<SignalA>` can be mapped to two different resources `<Resource_1>` and `<Resource_2>` as follows:

```
<hc:CapabilityMap>
  <hc:Mapping>
```

```
    <hc:Map>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalA"]/
          hc:Interface/c:Ports/c:Port[@name="OutA"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
    </hc:Map>
    <hc:Map>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
    </hc:Map>
  </hc:Mapping>
</hc:CapabilityMap>
```

By definition, this mapping is interpreted to mean that if <SignalA> is in use, then it requires the use of both <Resource1> and <Resource2>.

F.3.4.5 Defining a capability that always uses the ports of two different resources at the same time

By examining these examples, it is clearly possible to put more than one resource in a single <Mapping> <Map>, as in this example:

```
<hc:CapabilityMap>
  <hc:Mapping>
    <hc:Map>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalA"]/
          hc:Interface/c:Ports/c:Port[@name="OutA"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
    </hc:Map>
  </hc:Mapping>
</hc:CapabilityMap>
```

This example is interpreted to mean that if either <Resource_1> or <Resource_2> is supplying the capability “SignalA”, then that same signal is on the ports of both resources. This configuration can be useful in some circumstances to describe triggering capabilities, for example.

This case is similar to the discussion in F.3.4.4, but it is distinguished in the fact that this case defines a signal that locks the output of two resources together so that the signals on each resources port are always identical. In F.3.4.4, the defined signal consumes two resources, but the signals on the resources’ ports do not have to be identical.

F.3.4.6 Defining a resource that can do more than one thing at a time

Some instruments are capable of doing more than one thing at the same time even though they have only a single port. For instance, a power supply may be capable of supplying a voltage at a port while simultaneously measuring the current that flows through the port. This case requires a special type of resource definition (a resource that can do more than one thing at a time).

To define this type of resource, simply add more than one capability to the resource inside of a single <Mapping> <Map>:

```
<hc:CapabilityMap>
  <hc:Mapping>
    <hc:Map>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="sinewave"]/
          hc:Interface/c:Ports/c:Port[@name="sineWave"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="measRMS"]/
          hc:Interface/c:Ports/c:Port[@name="Input"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
    </hc:Map>
  </hc:Mapping>
</hc:CapabilityMap>
```

This example is interpreted to mean that the resource <Resource_1> has both capabilities “sinewave” and “measRMS” at the same time.

There are no restrictions about the signals that can be defined in this type of capability mapping; therefore, some additional interpretations are necessary:

a) If two signal definitions in a single <Map> are both sources of the same type (e.g., Voltage), then the signals are added together at the resource port (not multiplied or shorted out). For example, this is a way to add a dc voltage offset to an ac signal.

b) If two signal definitions in a single <Map> are both sources of different types (e.g., Power and Impedance), then the signals’ characteristics at the resource port are both simultaneously available. An example may be 50 Ω load on a RF output. (Mixing signals of different types cannot defy the laws of physics.)

c) If two signal definitions in a single <Map> are both measurements, then both measurements are independently executed at the same time.

F.3.5 Wire the resource interfaces to the instrument interfaces

The final step that is needed to define instrument capabilities is to define the way that resources are connected to the physical instrument ports. Remember that resources are logical quantities and do not necessarily have a direct correspondence to physical hardware. In other words, more than one resources port may be (logically) connected to a single physical instrument port, although in this case the use of switches (described in F.3.5.1) is necessary. Many resource ports will have a simple one-to-one mapping to instrument ports.

This mapping is accomplished with an <NetworkList> element. <NetworkList> utilizes the <Network> type to define connections between resource ports and instrument ports. In essence, the <NetworkList> element defines a circuit topology. To wire up a resource to an instrument port, simply define a network node that contains references to the ports of both elements.

A simple instrument with one resource and one port would have a <NetworkList> entry similar to this example:

```
<hc:NetworkList>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="Output"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
</hc:NetworkList>
```

For multiple resources and multiple ports, additional <Network> entries must be added. For instance, an instrument with four physical ports and two resources, each with two ports, would have a <NetworkList> similar to this example:

```
<hc:NetworkList>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="Output1"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>

  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="Output2"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]
        /hc:Interface/c:Ports/c:Port[@name="P2"]
      </hc:Path>
    </hc:Node>
  </hc:Network>

  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="Output3"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]
        /hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="Output4"]
      </hc:Path>
```

```
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
        hc:Interface/c:Ports/c:Port[@name="P2"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
</hc:NetworkList>
```

F.3.5.1 Using switches to specify signal routing options

Often instruments are capable of internally routing signals to different output ports. It is quite common, for example, for an instrument to have signal connectors on both the front and back of the instrument. These instruments usually allow the user to choose the port to which a signal is sent. In ATML terms, these instruments have a single resource that can be routed to different physical ports. The description of these instruments requires the use of a special <Switch> element, which is then referenced in the <NetworkList>. <Switch> elements are contained in a list that is enclosed by a <Switching> element.

The <Switch> element lists the number of ports needed for the switch and the connections that the switch can make between its ports.

NOTE—This element is a logical switch (it describes routing options in the instrument, but need not exactly reproduce the architecture of any physical switching hardware in the instrument). In fact, the <Switch> element will always have more ports than the physical switches in the instrument because the <Switch> element must include connections for the resources as well as the physical instrument ports.

To define a switch, the <Switch> element requires a list of ports and a list of the possible connections that the switch can make. The list of ports is a simple <Ports> element; the list of connections is called <Connections>. <Connections> includes a list of <RelaySettings>, where the <RelaySetting> includes a list of all the connections (called <RelayConnection>) that would be active in any particular switch state.

For example, consider a switch that has three ports. One port is to be connected to the instrument’s resource, and the other two are to be connected to two different physical ports. This instrument is able to switch the output of the resource to either the front of the unit or the back of the unit, but not both at the same time. The definition of this switch would be similar to this example:

```
<inst:Switching>
  <hc:Switch name="Front_Back_Switch">
    <hc:Interface>
      <c:Ports>
        <c:Port name="Port1" />
        <c:Port name="Port2" />
        <c:Port name="Resource" />
      </c:Ports>
    </hc:Interface>
    <hc:Connections>
      <hc:RelaySetting name="Front">
        <hc:RelayConnection from="Port1" to="Resource" />
      </hc:RelaySetting>
      <hc:RelaySetting name="Back">
        <hc:RelayConnection from="Port2" to="Resource" />
      </hc:RelaySetting>
    </hc:Connections>
  </hc:Switch>
</inst:Switching>
```

F.3.6 Considerations for mapping signals to ports in complex instruments

Instruments may have the ability to generate (or measure) many different signals, and they may also have many ports to which these signals may be routed. Some instruments have multiple output ports, each of which can be the source of different signals. But the internal operation of instruments can be quite complex, and the mapping of signals to ports can likewise be complicated.

Sometimes an instrument can generate multiple signals but not all at the same time. Or, there may be other restrictions, such as with an instrument that is able to produce certain signals at one port and other signals at a second port, but cannot route these same signals to the opposite port. The description of these instruments must be able to handle these types of situations. This mapping is accomplished through the use of separate definitions for capabilities, resources, and physical ports along with the ability to add logical switching to the `<NetworkList>` element.

F.3.6.1 Signal-to-port mapping requirements

In more detail, the ability to describe signal-to-port mapping must have the following characteristics:

a) Must be easy to create.
   1) The mapping must be easy to understand.
   2) The mapping should be compact, i.e., it cannot create excessively large XML instance documents.
   3) Simple instruments should be simple to describe.
   4) Complex instruments may be difficult to describe (but obviously do not have to be).
b) Must be able to describe that a signal can go to one and only one port.
c) Must be able to describe that a signal can go to some port(s) on an instrument but not others.
d) Must be able to describe that a signal monopolizes a port, i.e., no other signal can be routed to that port.
e) Must be able to describe that a signal can be routed to any of a number of ports, but only one at a time.
f) Must be able to describe that a signal can be routed to any of a number of ports, all at the same time.
   1) The number of ports to which the signal can be routed may be less than the total number of ports on the instrument. For example, consider a signal that can be routed to any three of five possible ports.
g) Must be able to handle the case where generation of **Signal A** prevents the generation of **Signal B** even if **Signal B** could otherwise be generated.
h) Must be able to handle the case where generation of **Signal A** prevents **Signal B** from being routed to some ports even though it can still be generated and routed to the remaining ports.
i) Must be able to handle all combinations of each requirement, all on the same instrument.

F.3.7 Specifying software-specific capabilities using the instrument instance XML schema

The true capabilities of an instrument cannot be specified by considering the hardware alone. If the instrument is used with a software driver, then the instrument's capabilities are actually defined by a combination of the hardware's capabilities and the drivers. Some drivers may not expose all of the functionality of the instrument, while other drivers may actually include signal processing functions that extend the capabilities of the hardware.

Defining these instrument/driver combinations requires the use of ATML Instrument Instance XML schemas. The ATML Instrument Instance generally includes information about a certain **specific** instrument, such as the serial number or calibration history. In addition, the ATML Instrument Instance describes embedded capabilities.

To utilize the ATML Instrument Instance XML schema to define hardware and/or software capabilities, use the following technique:

a) Resource definitions and resource-to-port mappings go into ATML Instrument Description.

b) Resource-to-capability mappings go into Instrument Instance.

By separating data in this way, the ATML Instrument Instance can map more (or fewer) capabilities onto the instrument's resources to match the capabilities that are exposed by the driver. An instrument in a different test system, using a different driver, will naturally be matched with a different ATML Instrument Instance that can be tuned to the capabilities of its own driver.

F.3.8 A simple example

Consider the case of a simple two-channel signal source that can generate only sine wave signals. This instrument is essentially the same as two signal sources in a single physical package. The instrument can generate any two signals at the same time. The ports cannot be swapped, i.e., the signals from each of the two internal generators are hardwired to specific ports.

In this example, only a single signal type can be created with two resources and two ports. Descriptions of all of these entities are contained in the ATML Instrument Description, parts of which could contain the following:

```
<!-- Physical instrument ports “1” and “2” !-->
<hc:Interface>
  <c:Ports>
    <c:Port name="1" />
    <c:Port name="2" />
  </c:Ports>
</hc:Interface>

<!-- Each resource is hard-wired to one of the physical instrument ports !-->
<hc:NetworkList>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="1"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>

  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="2"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
</hc:NetworkList>

<inst:Resources>
  <hc:Resource name="Resource_1">
```

```
    <hc:Interface>
      <c:Ports>
        <c:Port name="P1"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
  <hc:Resource name="Resource_2">
    <hc:Interface>
      <c:Ports>
        <c:Port name="P1"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
</inst:Resources>

<inst:Capabilities>

  <!-- One capability, called “sinewave” !-->
  <hc:Capability name="sinewave">
    <hc:Interface>
      <c:Ports>
        <c:Port name="Out" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="sinewaveSignal" Out="Out">
        <std:Sinusoid
          name="sineWave"
          frequency="10kHz range 1kHz to 10MHz errlmt 0.1Hz res 1Hz"
          amplitude="trms 1V range 1uV to 1V errlmt 0.1% range 1V to 10V errlmt 1%"/>
        </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>

  <!-- Describe the relationship between resources and capabilities.
     In this case both resources can supply the same signal. !-->
  <hc:CapabilityMap>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="sinewave"]/
            hc:Interface/c:Ports/c:Port[@name="Out"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
            hc:Interface/c:Ports/c:Port[@name="P1"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>

    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="sinewave"]/
            hc:Interface/c:Ports/c:Port[@name="Out"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
            hc:Interface/c:Ports/c:Port[@name="P1"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
  </hc:CapabilityMap>
</inst:Capabilities>
```

F.3.9 Satisfying requirements use cases

F.3.9.1 through F.3.9.7 examine the key use cases for signal-to-port mapping and describe how to handle each of them with ATML Instrument Description (or ATML Instrument Instance).

F.3.9.1 Describing a signal that can go to one and only one port

If a signal can be routed to only one port, it should be assigned to only one resource and that resource should be mapped to only one instrument port.

Example:

```
<!-- Name the physical instrument port "1" !-->
<hc:Interface>
  <c:Ports>
    <c:Port name="1" />
  </c:Ports>
</hc:Interface>

<!-- Each resource is hard-wired to one of the physical instrument ports !-->
<hc:NetworkList>
  <hc:Network>
    <hc:Node>
      <hc:Path>
      /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="1"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
</hc:NetworkList>

<!-- Describe the resource !-->
<inst:Resources>
  <hc:Resource name="Resource_1">
    <hc:Interface>
      <c:Ports>
        <c:Port name="P1"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
</inst:Resources>

<inst:Capabilities>
  <hc:Capability name="sinewave">
    <hc:Interface>
      <c:Ports>
        <c:Port name="sineWave" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="sinewaveSignal" Out="sineWave">
        <std:Sinusoid
          name="sineWave"
          frequency="10kHz range 1kHz to 10MHz errlmt 0.1Hz res 1Hz"
          amplitude="trms 1V range 1uV to 1V errlmt 0.1% range 1V to 10V errlmt 1%"/>
        </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>

  <!-- One signal maps to one resource. !-->
  <hc:CapabilityMap>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="sinewave"]/
            hc:Interface/c:Ports/c:Port[@name="sineWave"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
            hc:Interface/c:Ports/c:Port[@name="P1"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
```

```
  </hc:CapabilityMap>
</inst:Capabilities>
```

If the instrument contains any other capabilities that can be routed to other ports, then they should be assigned to a different resource. That new resource can then be mapped to other ports.

F.3.9.2 Describing a signal that can go to some port(s) on an instrument but not others

A signal can be routed only to the instrument port(s) to which its associated resource is wired. If a signal cannot be directed to a port, then it should not be assigned.

F.3.9.3 Describing a signal that monopolizes a port

To have exclusive use of a port, a signal should be assigned to a resource that is connected to the desired port, and no other resources should be connected to that port. If the instrument has other capabilities that can utilize other ports, then other resources must be created.

Likewise, a resource can monopolize a port if it is the only resource that is connected to the port (including any possible alternate signal routes through switches). In this case, the resource monopolizes the port, but multiple different signals can be routed to the port if the resource is capable of generating them.

F.3.9.4 Describing a signal that can be routed to any of a number of ports, one at a time

Routing signals to various ports requires the use of a <Switching> element in the ATML Instrument Description, as described in F.3.5.1. The example in F.3.5.1 is a good example of this type of connection.

F.3.9.5 Describing a signal that can be routed to any of a number of ports, all at once

If an instrument has highly flexible signal routing that allows a signal to be routed to any of several ports, including the ability to route the signal to more than one port at the same time, the use of <Switching> elements is required. When defining the switch, all of the possible connections must be listed in the <Connections> setting.

This XML snippet shows how to configure a switch that can route a signal from a resource to either of two ports or to both ports at the same time. This example is identical to the example in F.3.5.1 except that it includes a <RelaySetting> entry that adds the ability to send the signal to both ports at once.

```
<inst:Switching>
  <hc:Switch name="Front_Back_Switch">
    <hc:Interface>
      <c:Ports>
        <c:Port name="Port1" />
        <c:Port name="Port2" />
        <c:Port name="Resource" />
      </c:Ports>
    </hc:Interface>
    <hc:Connections>
      <hc:RelaySetting name="Front">
        <hc:RelayConnection from="Port1" to="Resource" />
      </hc:RelaySetting>
      <hc:RelaySetting name="Back">
        <hc:RelayConnection from="Port2" to="Resource" />
      </hc:RelaySetting>
      <hc:RelaySetting name="Both">
        <hc:RelayConnection from="Port1" to="Resource" />
        <hc:RelayConnection from="Port2" to="Resource" />
      </hc:RelaySetting>
    </hc:Connections>
  </hc:Switch>
```

F.3.9.6 Describing how the generation of SignalA prevents the generation of SignalB, even if SignalB could otherwise be generated

If signals in an instrument are mutually exclusive, then they should simply be assigned to the same resource. This case is one of the primary intended uses of `<Resources>`. `<Resources>` defines what an instrument can do, and it also defines what an instrument cannot do. In this context, a resource may be able to supply `<signalA>` and `<signalB>`, but if the `<signalA>` is chosen, it implies `<signalB>` is not available.

A more complicated situation may exist if `<signalA>` prevents the generation of `<signalB>`, but a third signal (`<signalC>`) does not. `<signalB>` and `<signalC>` can be generated at the same time and sent to different instrument ports. For this example, assume that `<signalA>` and `<signalC>` is also mutually exclusive, i.e., `<signalA>` uses up all of the instrument's inherent capabilities.

In this case, `<signalB>` and `<signalC>` are both assigned to two different resources, and those resources are connected to the two instrument ports. `<signalA>` is assigned to **both** resources.

NOTE—This case will be true even if `<signalA>` really requires only a single instrument port. When `<signalA>` is generated, its use will then allocate both resources and prevent either `<signalB>` or `<signalC>` from being generated even if a physical instrument port is available.

An XML snippet of this case is illustrated by the following example:

```
<hc:Interface>
  <c:Ports>
    <c:Port name="1" />
    <c:Port name="2" />
  </c:Ports>
</hc:Interface>

<hc:NetworkList>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="1"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="2"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
</hc:NetworkList>

<inst:Resources>
  <hc:Resource name="Resource_1">
    <hc:Interface>
      <c:Ports>
        <c:Port name="P1"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
  <hc:Resource name="Resource_2">
    <hc:Interface>
      <c:Ports>
```

```
        <c:Port name="P1"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
</inst:Resources>

<inst:Capabilities>

  <hc:Capability name="SignalA">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutA" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="sinewaveSignal" Out="OutA">
        <std:Sinusoid
          name="sineWave"
          frequency="10kHz range 1kHz to 10MHz errlmt 0.1Hz res 1Hz"
          amplitude="trms 1V range 1uV to 1V errlmt 0.1% range 1V to 10V errlmt 1%"/>
        </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
  <hc:Capability name="SignalB">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutB" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="sinewaveSignal" Out="OutB">
        <std:Sinusoid
          name="sineWave"
          frequency="10kHz range 1kHz to 10MHz errlmt 0.1Hz res 1Hz"
          amplitude="trms 1V range 1uV to 1V errlmt 0.1% range 1V to 10V errlmt 1%"/>
        </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
  <hc:Capability name="SignalC">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutC" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="sinewaveSignal" Out="OutC">
        <std:Sinusoid
          name="sineWave"
          frequency="10kHz range 1kHz to 10MHz errlmt 0.1Hz res 1Hz"
          amplitude="trms 1V range 1uV to 1V errlmt 0.1% range 1V to 10V errlmt 1%"/>
        </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>

  <!-- One signal maps to one resource. !-->
  <hc:CapabilityMap>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalB"]/
            hc:Interface/c:Ports/c:Port[@name="OutB"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
            hc:Interface/c:Ports/c:Port[@name="P1"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalB"]/
            hc:Interface/c:Ports/c:Port[@name="OutB"]
          </hc:Path>
        </hc:Node>
```

```
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
            hc:Interface/c:Ports/c:Port[@name="P1"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalC"]/
            hc:Interface/c:Ports/c:Port[@name="OutC"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
            hc:Interface/c:Ports/c:Port[@name="P1"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalC"]/
            hc:Interface/c:Ports/c:Port[@name="OutC"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
            hc:Interface/c:Ports/c:Port[@name="P1"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalA"]/
            hc:Interface/c:Ports/c:Port[@name="OutA"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
            hc:Interface/c:Ports/c:Port[@name="P1"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalA"]/
            hc:Interface/c:Ports/c:Port[@name="OutA"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
            hc:Interface/c:Ports/c:Port[@name="P1"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
  </hc:CapabilityMap>
</inst:Capabilities>
```

An apparently even more complicated case exists if `<SignalA>` prevents the generation of `<SignalB>`, `<SignalB>` and `<SignalC>` can be generated at the same time, and `<SignalA>` and `<SignalC>` can also be generated at the same time. However, the use of resources actually makes this case easy to handle. This situation is handled via the following steps:

a) Define resources <Resource_1> and <Resource_2>.
b) Assign <SignalA> to <Resource_1>.
c) Assign <SignalB> to <Resource_1>.
d) Assign <SignalC> to <Resource_2>.
e) Connect both resources to both ports.

With this structure, if <SignalA> is being generated, then <SignalB> cannot be generated because its resource is already in use, while <SignalC> can be generated because it uses a different resource. Likewise, <SignalB> and <SignalC> can be generated at the same time because they use different resources. The XML snippet for this instrument would be similar to this example:

```
<hc:Interface>
  <c:Ports>
    <c:Port name="1" />
    <c:Port name="2" />
  </c:Ports>
</hc:Interface>

<hc:NetworkList>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="1"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="1"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="2"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="2"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
</hc:NetworkList>
```

```
<inst:Resources>
  <hc:Resource name="Resource_1">
    <hc:Interface>
      <c:Ports>
        <c:Port name="P1"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
  <hc:Resource name="Resource_2">
    <hc:Interface>
      <c:Ports>
        <c:Port name="P1"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
</inst:Resources>

<inst:Capabilities>

  <hc:Capability name="SignalA">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutA" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="sinewaveSignal" Out="OutA">
        <std:Sinusoid
          name="sineWave"
          frequency="10kHz range 1kHz to 10MHz errlmt 0.1Hz res 1Hz"
          amplitude="trms 1V range 1uV to 1V errlmt 0.1% range 1V to 10V errlmt 1%"/>
        </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
  <hc:Capability name="SignalB">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutB" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="sinewaveSignal" Out="OutB">
        <std:Sinusoid
          name="sineWave"
          frequency="10kHz range 1kHz to 10MHz errlmt 0.1Hz res 1Hz"
          amplitude="trms 1V range 1uV to 1V errlmt 0.1% range 1V to 10V errlmt 1%"/>
        </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
  <hc:Capability name="SignalC">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutC" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="sinewaveSignal" Out="OutC">
        <std:Sinusoid
          name="sineWave"
          frequency="10kHz range 1kHz to 10MHz errlmt 0.1Hz res 1Hz"
          amplitude="trms 1V range 1uV to 1V errlmt 0.1% range 1V to 10V errlmt 1%"/>
        </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>

  <!-- One signal maps to one resource. !-->
  <hc:CapabilityMap>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalB"]/
            hc:Interface/c:Ports/c:Port[@name="OutB"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
            hc:Interface/c:Ports/c:Port[@name="P1"]
          </hc:Path>
```

```
    </hc:Node>
  </hc:Map>
</hc:Mapping>
<hc:Mapping>
  <hc:Map>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalB"]/
        hc:Interface/c:Ports/c:Port[@name="OutB"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Map>
</hc:Mapping>
<hc:Mapping>
  <hc:Map>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalC"]/
        hc:Interface/c:Ports/c:Port[@name="OutC"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Map>
</hc:Mapping>
<hc:Mapping>
  <hc:Map>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalC"]/
        hc:Interface/c:Ports/c:Port[@name="OutC"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Map>
</hc:Mapping>
<hc:Mapping>
  <hc:Map>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalA"]/
        hc:Interface/c:Ports/c:Port[@name="OutA"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Map>
  <hc:Map>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalA"]/
        hc:Interface/c:Ports/c:Port[@name="OutA"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Map>
</hc:Mapping>
```

```
  </hc:CapabilityMap>
</inst:Capabilities>
```

The <NetworkList> element is not shown here because an actual instrument may require signal routing and switching to be specified so that any of these signals can be routed to the correct port.

F.3.9.7 Describing how generation of SignalA prevents SignalB from being routed to some ports, even though SignalB can still be generated and routed to the remaining ports

This unusual and apparently complicated example is actually fairly easy to handle. Simply create two resources, e.g., <ResourceA> and <ResourceB>. Assign <SignalA> and <SignalB> to <ResourceA>. Assign <SignalB> to <ResourceB>. Connect <ResourceA> to some of the instrument ports, and <ResourceB> to the others. This setup will prevent <SignalB> from being sent to the ports that <SignalA> is using (because <ResourceA> will be in use), while still allowing <SignalB> to be sent to the remaining ports (using <ResourceB>).

For example, suppose an instrument has three ports, and <SignalA> blocks the use of <SignalB> on two of them. An XML snippet for this case would be similar to this example:

```
<hc:Interface>
  <c:Ports>
    <c:Port name="1" />
    <c:Port name="2" />
    <c:Port name="3" />
  </c:Ports>
</hc:Interface>

<hc:NetworkList>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="1"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="2"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="3"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
</hc:NetworkList>

<inst:Resources>
  <hc:Resource name="Resource_1">
```

```
    <hc:Interface>
      <c:Ports>
        <c:Port name="P1"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
  <hc:Resource name="Resource_2">
    <hc:Interface>
      <c:Ports>
        <c:Port name="P1"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
</inst:Resources>

<inst:Capabilities>

  <hc:Capability name="SignalA">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutA" />
      </c:Ports>
    </hc:Interface>
    <!-- 1641 Signal Description omitted for brevity -->
  </hc:Capability>
  <hc:Capability name="SignalB">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutB" />
      </c:Ports>
    </hc:Interface>
    <!-- 1641 Signal Description omitted for brevity -->
  </hc:Capability>
  <hc:Capability name="SignalC">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutC" />
      </c:Ports>
    </hc:Interface>
    <!-- 1641 Signal Description omitted for brevity -->
  </hc:Capability>

  <hc:CapabilityMap>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalA"]/
            hc:Interface/c:Ports/c:Port[@name="OutA"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
            hc:Interface/c:Ports/c:Port[@name="P1"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalB"]/
            hc:Interface/c:Ports/c:Port[@name="OutB"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /id:InstrumentDescription/id:Resources/hc:Resource[@name="Resource_1"]/
            hc:Interface/c:Ports/c:Port[@name="P1"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
```

```
          /id:InstrumentDescription/id:Capabilities/hc:Capability[@name="SignalB"]/
          hc:Interface/c:Ports/c:Port[@name="OutB"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_2"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
    </hc:Map>
  </hc:Mapping>
  <hc:Mapping>
    <hc:Map>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SignalC"]/
          hc:Interface/c:Ports/c:Port[@name="OutC"]
        </hc:Path>
      </hc:Node>
      <hc:Node>
        <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
          hc:Interface/c:Ports/c:Port[@name="P1"]
        </hc:Path>
      </hc:Node>
    </hc:Map>
  </hc:Mapping>
 </hc:CapabilityMap>
</inst:Capabilities>
```

F.3.10 More complex examples

F.3 through F.3.9 illustrated ways in which the ATML Instrument Description XML schemas can be utilized to describe instrument capabilities and route those capabilities to ports. F.3.10.1 through F.3.10.3 illustrate some more use cases and describe how some real-life problems are to be solved.

F.3.10.1 Timing examples (synchronous, asynchronous, parallel)

Some TPs have strict requirements regarding timing of signals or require multichannel synchronous signals. In this context, the following definitions are used:

Synchronous: Two events are said to be synchronous if they are related by some defined time interval.

For the purpose of the definition:

a) A time interval of 0 s is a defined time interval.

b) A time interval can have uncertainty such as 4 ns ± 1 ns.

c) A time interval can be open ended, e.g., greater than 2 ms, less than 20 ms.

d) A pair of synchronous analog signals has a fixed phase relationship (which is the same as a fixed time delay between the signals). If the signals are in phase, then the time interval is 0.

Asynchronous: Two events are said to be asynchronous if they are not related by any defined time interval. A shorter version is that the events are unrelated.

Asynchronous Testing: Performing multiple tests that have no time dependency.

Synchronous Testing: Performing multiple tests where some event within a test is synchronous with another event within another test.

Parallel Testing: Performing multiple tests over a time interval where the testing is regarded as concurrent.

Timing issues can be handled in two different ways. First, IEEE Std 1641 [B29] can easily be utilized to specify timing capabilities, both in terms of time intervals between events and synchronicity between multiple channels. In many cases, this approach will be the easiest way to specify timing.

Secondly, the ATML Instrument Description XML schema includes elements that can be utilized to define information about the external triggering capabilities of the instruments that it describes. In some cases, it will be possible to use that information to determine whether a given TP's timing requirements can be met.

Some specific examples of these cases are described in F.3.10.1.1 through F.3.10.1.6.

F.3.10.1.1 A resource outputs multiple channels so that the signals or events on the two channels are synchronized

Synchronous signals are commonly required in test systems. Some examples include the following:

a) An RF synthesizer that outputs both the RF modulated signal plus the low-frequency modulating signal.
b) An ARB that outputs a waveform trace plus a synchronized event (on a TTL line) when the trace starts.
c) Three-phase power supply where each output is phase locked with the others.
d) I and Q channels.
e) A two-channel oscilloscope.
f) A two-channel synchronous receiver such as a network analyzer.

Synchronous signals are easily described utilizing IEEE Std 1641 [B29]. Within ATML Instrument Descriptions, this mapping would result in a capability description that has more than one port. These ports would be mapped to a multiport resource whose ports are in turn mapped to multiple physical instrument ports. Signal synchronicity is defined by the IEEE 1641 signal definition, and the additional ATML Instrument Description information maps the signals onto the instrument's output ports.

F.3.10.1.2 A resource outputs multiple channels so that the signals or events on the two channels are asynchronous

To describe asynchronous signals, nothing special has to be done. Signals are asynchronous by default. The separate signals are simply assigned to different resources, and those resources are connected to different signal ports.

F.3.10.1.3 Using resources to describe asynchronous testing

Some instrument functions cannot be executed in parallel, but neither their order nor their timing is especially important. For instance, suppose both the voltage and current are to be measured at some point in a circuit using a simple DMM. For this test, it does not matter which parameter is measured first, but the information must allow the system to determine that current cannot be measured while voltage is measured, even if there are two resources on the instrument.

This example is particularly complex because there are other situations where the same instrument can measure voltage and resistance at the same time, namely, when the instrument is measuring voltage at one point and resistance at some other (unrelated) point.

Once again, this case is easily handled with the use of `<Resource>`. However, the particular example discussed here also requires some knowledge of the effects of an instrument on a circuit—knowledge that is **not** captured by ATML Instrument Descriptions.

This example can be considered in two different configurations. The instrument may have only a single port that measures either voltage or current; or the instrument may have two ports, one of which measures voltage and the other, current.

F.3.10.1.3.1.1 Single-port instrument

For single-port instruments, the ATML Instrument Description document would include these items:

a) Two capabilities descriptions: one for voltage measurements and the other for resistance measurements
b) One resource, to which both capabilities are assigned
c) One port, which is connected to the single resource

The use of a single resource prevents both capabilities from being used at the same time.

F.3.10.1.3.1.2 Two-port instrument

For two-port instruments, the ATML Instrument Description document would include these items:

a) Two capabilities descriptions: one for voltage measurements and the other for resistance measurements
b) Two resources, one of which is assigned to voltage measurements and the other to resistance measurements
c) Two ports, each of which is connected to one of the two resources

This configuration allows voltage and resistance measurements to be made in parallel, if desired. It does not require these measurements to be made at the same time; it merely enables the capability.

The information contained in the ATML Instrument Description document does not restrict the use of the instrument from measuring voltage and resistance at the same time and at the same point in the UUT. If the instrument in question functions like a common DMM, resistance measurements require that a voltage be applied to the UUT. This step would obviously affect the voltage measurement. Even a rudimentary knowledge of circuit behavior makes it obvious that executing these two tests on the same point and at the same time is not a good idea. However, ATML does not specify the principles of analog circuit behavior. It is the responsibility of the test designer to ensure that the tests do not interfere with one another.

F.3.10.1.4 Using a resource to describe synchronous testing

To define an instrument that can generate (or measure) two analog signals with a consistent phase relationship, an IEEE 1641-based signal description should be utilized.

To illustrate, a very simple capability definition for a two-channel oscilloscope would be as follows:

```
<inst:Capabilities>
  <hc:Capability name="TwoChannels">
    <hc:Interface>
      <c:Ports>
        <c:Port name="Trace1" />
        <c:Port name="Trace2" />
        <c:Port name="Ch1" />
        <c:Port name="Ch2" />
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="Osc" In="Ch1 Ch2" Out="Trace1 Trace2">
        <std:Instantaneous name="Trace1" In="Ch1"/>
        <std:Instantaneous name="Trace2" In="Ch2"/>
      </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
</inst:Capabilities>
```

This definition would be used to describe measurements that use both channels at the same time. Measurements that use only one channel would be described separately.

If the two channels are to be triggered synchronously, then addition information in the IEEE 1641-based description is needed that describes the trigger line and its functionality:

```
<std:Signal name="Osc" In="Ch1 Ch2" Out="Trace1 Trace2">
  <std:Instantaneous name="TrigInit" samples="0" In="Ch1" condition="GT" nominal="2.5V" />
  <std:EventedEvent name="Trig" In="TrigInit" />
  <std:Instantaneous name="Trace1" In="Ch1" Sync="Trig" />
  <std:Instantaneous name="Trace2" In="Ch2" Sync="Trig" />
</std:Signal>
```

In this case, only the IEEE 1641-based description is changed. The remainder of the `<Capabilities>` section in this example would remain the same.

F.3.10.1.5 Using resources to describe synchronous fixed-time events

Another type of synchronous signal involves the creation (or measurement) of two events over a predetermined fixed time interval. As in the F.3.10.1.4 example, this case is best handled by describing the desired functionality by an IEEE 1641-based signal description.

For example, a waveform that outputs an event (a trigger signal) after some time delay could be described utilizing IEEE Std 1641 [B29] similar to this example:

```
<std:Signal name="MyEventMaker" Out="Events Trace">
  <std:PulsedEvent name="Events" pulses="(0,(Trace.period),0)"/>
  <std:WaveformStep name="Trace" samplingInterval="1us range 1ns to 1s" />
</std:Signal>
```

This signal would be assigned to a two-port resource (one for the waveform and the other for the generated event).

F.3.10.1.6 Using resources to describe multiple signals running in parallel

Running parallel tests, or executing multiple simultaneous signals, is very simple if the instrument has multiple physical ports. Simply define multiple resources in the ATML Instrument Description document, and connect the resources to the different ports.

A more interesting example arises when considering an instrument that can accept a signal through a single port and then make multiple simultaneous measurements. For instance, an oscilloscope may be able to measure both overshoot and rise time on a single trace at the same time. This capability cannot be described by creating two resources that are connected to the same port because the use of one resource would allocate the port and prevent the other resource from using the same port. Instead, a switch element must be used. The switch simply routes the signal from the port to both of the resources. The XML snippet for this case might be something like this example:

```
<hc:Interface>
  <c:Ports>
    <c:Port name="1"/>
  </c:Ports>
</hc:Interface>

<hc:NetworkList>
  <hc:Network>
    <hc:Node>
      <hc:Path>
      /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="1"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Switching/hc:Switch[@name="LogicalFunctionRouting
        "]/
        hc:Interface/c:Ports/c:Port[@name="Port1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Switching/hc:Switch[@name="LogicalFunctionRouting
        "]/
        hc:Interface/c:Ports/c:Port[@name="Resource1"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="RiseTimeResource"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Switching/hc:Switch[@name="LogicalFunctionRouting
        "]
        /hc:Interface/c:Ports/c:Port[@name="Resource2"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="OvershootResource"]/
        hc:Interface/c:Ports/c:Port[@name="P1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
</hc:NetworkList>

<inst:Resources>
  <hc:Resource name="RiseTimeResource">
    <hc:Interface>
      <c:Ports>
        <c:Port name="P1"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
  <hc:Resource name="OvershootResource">
    <hc:Interface>
      <c:Ports>
        <c:Port name="P1"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
</inst:Resources>

<inst:Switching>
  <hc:Switch name="LogicalFunctionRouting">
    <hc:Interface>
      <c:Ports>
        <c:Port name="Port1"/>
        <c:Port name="Resource1"/>
```

```
        <c:Port name="Resource2"/>
      </c:Ports>
    </hc:Interface>
    <hc:Connections>
      <hc:RelaySetting name="TheOnlyConnection">
        <hc:RelayConnection to="Resource1" from="Port1"/>
        <hc:RelayConnection to="Resource2" from="Port1"/>
      </hc:RelaySetting>
    </hc:Connections>
  </hc:Switch>
</inst:Switching>

<inst:Capabilities>
  <hc:Capability name="RiseTimeMeas">
    <hc:Interface>
      <c:Ports>
        <c:Port name="Out"/>
      </c:Ports>
    </hc:Interface>
    <!-- 1641 Signal Description omitted for brevity -->
  </hc:Capability>
  <hc:Capability name="OvershootMeas">
    <hc:Interface>
      <c:Ports>
        <c:Port name="Out"/>
      </c:Ports>
    </hc:Interface>
    <!-- 1641 Signal Description omitted for brevity -->
  </hc:Capability>
  <hc:CapabilityMap>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="RiseTimeMea
            s"]/
            hc:Interface/c:Ports/c:Port[@name="Out"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="RiseTimeResource
            "]/
            hc:Interface/c:Ports/c:Port[@name="P1"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="OvershootMeas
          "]/
          hc:Interface/c:Ports/c:Port[@name="Out"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="OvershootResourc
            e"]/
            hc:Interface/c:Ports/c:Port[@name="P1"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
  </hc:CapabilityMap>
</inst:Capabilities>
```

NOTE—This example shows how resource definitions do not always correspond to physical hardware in the instrument. In this case, the instrument may be made up of a digitizer, some memory, and a signal processor. The two measurements would both be extracted from the sampled data by the signal processor. The resources that are defined in the previous example would both roughly correspond to the signal processor, but there is only a single processor, and there can be multiple resources.

As an alternative approach, it is always possible to define multiple outputs utilizing the IEEE 1641 signal-based capability description. In that case, the capability description would include the entire logical signal routing necessary to define both measurements.

F.3.10.2 RF synthesizer use case

Consider the case of an RF synthesizer with multiple capabilities [i.e., amplitude modulation (AM), frequency modulation (FM), and phase modulation (PM)], which are mutually exclusive. The synthesizer also has two low-frequency function generators that are capable of generating a sine wave, a square wave, or a triangle wave. The outputs of these low-frequency function generators can be summed (if desired) and used as input to the high-frequency AM/FM/PM modulator. Figure F.1 depicts a block diagram for this synthesizer.

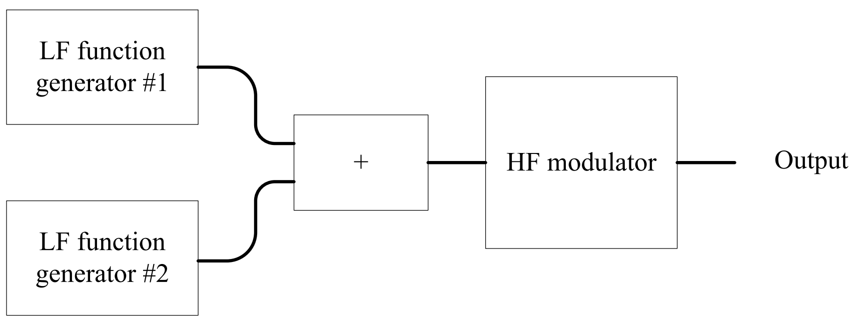


Figure F.1—Synthesizer block diagram

This synthesizer has only a single physical output port:

```
<hc:Interface>
  <c:Ports>
    <c:Port name="1"/>
  </c:Ports>
</hc:Interface>
```

There is more than one way to describe the available signals and resources in this instrument. It is possible to utilize the IEEE 1641 signal description constructs to completely describe the capabilities of the synthesizer. This approach would result in an ATML Instrument Description document that includes just one resource and one capability with a complicated signal description, similar to the XML snippet shown here:

```
<Signal Out="RFOut" >
  <rc:OneOf name="RFOut" In="AM FM PM"/>
  <AM name="AM" Carrier="carrier" In="LF" …/>
  <Sinusoid name="carrier" … />
  <FM name="FM" In="LF" …/>
  <PM name="PM" In="LF" …/>
  <rc:OneOf name="LF" In="Adder LF1"/>
  <Sum name="Adder" In="LF1 LF2"/>
  <rc:OneOf name="LF1" In="LF1SineWave LF1SquareWave LF1TriangleWave"/>
  <Sinusoid name="LF1OutSine" … />
  <SquareWave name="LF1OutSquare" … />
  <Triangle name="LF1OutTriangle" … />
  <rc:OneOf name="LF2" In="LF2SineWave LF2SquareWave LF2TriangleWave"/>
  <Sinusoid name="LF2OutSine" … />
  <SquareWave name="LF2OutSquare" … />
  <Triangle name="LF2OutTriangle" … />
```

```
</Signal>
```

Figure F.2 contains a block diagram representation of this example:

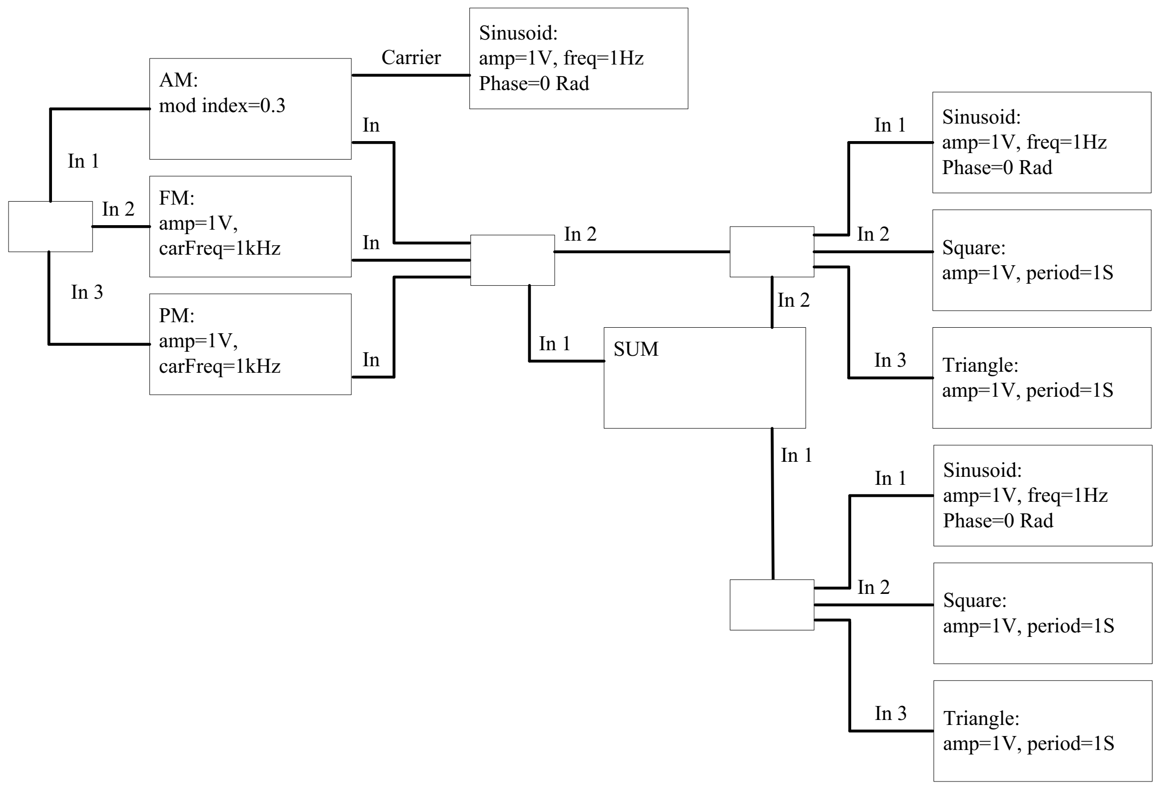


Figure F.2—One resource and one capability with a complicated signal description

This approach is not enlightening for purposes of this annex. Instead, the instrument will be described in terms of its internal capabilities at the ATML Instrument Description level.

To generate the ATML Instrument Description signal descriptions, assume that the two low-frequency generators in this instrument are identical. Each of them can generate three different types of signals. The high-frequency modulator can likewise generate three types of signals (one for each modulation type). Therefore, six different signal types must be defined. The low-frequency signals have only an output port; the high-frequency signals have both an input and an output.

In addition, the instrument has an additional capability: It can add the two low-frequency signals together. This capability can be described through the use of another signal capability with two inputs and a single output. The IEEE 1641-based description of this adder would include parameters that can specify whether the output of the adder contains only one signal or the sum of both inputs.

A listing of these capability definitions would look similar to the following (for brevity, the details of the IEEE 1641 signal descriptions are omitted here):

```
<inst:Capabilities>
  <hc:Capability name="SquareWave">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutSquare"/>
      </c:Ports>
    </hc:Interface>
```

```
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="SquareWave" Out="OutSquare">
        <std:Square name="OutSquare"/>
        <!-- 1641 Signal Description omitted for brevity -->
      </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
  <hc:Capability name="SineWave">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutSine"/>
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="SineWave" Out="OutSine">
        <std:Sinusoidal name="OutSine"/>
        <!-- 1641 Signal Description omitted for brevity -->
      </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
  <hc:Capability name="TriangleWave">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutTriangle"/>
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="TriangleWave" Out="OutTriangle">
        <std:Triange name="OutTriangle"/>
        <!-- 1641 Signal Description omitted for brevity -->
      </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
  <hc:Capability name="AM">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutAM"/>
        <c:Port name="InAM"/>
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="AM" In="InAM" Out="OutAM">
        <std:AM name="OutAM" In="InAM" Carrier="carrier"/>
        <std:Sinusoidal name="carrier"
          frequency="9kHz range 9kHz to 2.4GHz errlmt 1Hz"
          amplitude="-137dBm range -137dbM to +25dBm errlmt 2dB"/>
          <!-- 1641 Signal Description omitted for brevity -->
      </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
  <hc:Capability name="FM">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutFM"/>
        <c:Port name="InFM"/>
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="FM" In="InFM" Out="OutFM">
        <std:FM name="OutFM" In="InFM"
          carrierFrequency="9kHz range 9kHz to 2.4GHz errlmt 1Hz"
          freqDeviation="0Hz range 0Hz to 100kHz errlmt 5%"
          amplitude="-137dBm range -137dbM to +25dBm errlmt 2dB"/>
          <!-- 1641 Signal Description omitted for brevity -->
      </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
  <hc:Capability name="PM">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutPM"/>
        <c:Port name="InPM"/>
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="PM" In="InPM" Out="OutPM">
        <std:PM name="OutPM" In="InPM"
          carrierFrequency="9kHz range 9kHz to 2.4GHz errlmt 1Hz"
            phaseDeviation="0 rad range 0 rad to 10 rad errlmt 0.1%"
            amplitude="-137dBm range -137dbM to +25dBm errlmt 2dB"/>
          <!-- 1641 Signal Description omitted for brevity -->
```

```
      </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
  <hc:Capability name="Adder">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutAdder"/>
        <c:Port name="InAdder1"/>
        <c:Port name="InAdder2"/>
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal In="InAdder1 InAdder2" Out="OutAdder">
        <std:Sum name="OutAdder" In="InAdder1 InAdder2"/>
      </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
</inst:Capabilities>
```

Clearly, there are several internal resources needed in this example. Each low-frequency function generator corresponds with a resource, as does the high-frequency modulator. The adder represents a final resource.

```
<inst:Resources>
  <hc:Resource name="LF1">
    <hc:Interface>
      <c:Ports>
        <c:Port name="LF1Out"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
  <hc:Resource name="LF2">
    <hc:Interface>
      <c:Ports>
        <c:Port name="LF2Out"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
  <hc:Resource name="Modulator">
    <hc:Interface>
      <c:Ports>
        <c:Port name="ModOut"/>
        <c:Port name="ModIn"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
  <hc:Resource name="Adder">
    <hc:Interface>
      <c:Ports>
        <c:Port name="AdderIn1"/>
        <c:Port name="AdderIn2"/>
        <c:Port name="AdderOut"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
</inst:Resources>
```

Mapping capabilities to resources is fairly straightforward. Each low-frequency function generator can create the same three signals. The adder has only a single capability assigned to it, while the modulator has three:

```
  <hc:CapabilityMap>
    <!-- First low-frequency function generator -->
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SquareWave"
            ]/
            hc:Interface/c:Ports/c:Port[@name="OutSquare"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="LF1"]/
            hc:Interface/c:Ports/c:Port[@name="LF1Out"]
          </hc:Path>
        </hc:Node>
```

```
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SineWave"]/
            hc:Interface/c:Ports/c:Port[@name="OutSine"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="LF1"]/
            hc:Interface/c:Ports/c:Port[@name="LF1Out"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="TriangleWav
            e"]/
            hc:Interface/c:Ports/c:Port[@name="OutTriangle"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="LF1"]/
            hc:Interface/c:Ports/c:Port[@name="LF1Out"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>

    <!-- Second low-frequency function generator -->
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SquareWave"
            ]/
            hc:Interface/c:Ports/c:Port[@name="OutSquare"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="LF2"]/
            hc:Interface/c:Ports/c:Port[@name="LF2Out"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="SineWave"]/
            hc:Interface/c:Ports/c:Port[@name="OutSine"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="LF2"]/
            hc:Interface/c:Ports/c:Port[@name="LF2Out"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="TriangleWav
            e"]/
            hc:Interface/c:Ports/c:Port[@name="OutTriangle"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
```

```
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="LF2"]/
        hc:Interface/c:Ports/c:Port[@name="LF2Out"]
      </hc:Path>
    </hc:Node>
  </hc:Map>
</hc:Mapping>

<!-- High-frequency modulator -->
<hc:Mapping>
  <hc:Map>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="AM"]/
        hc:Interface/c:Ports/c:Port[@name="OutAM"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Modulator"]/
        hc:Interface/c:Ports/c:Port[@name="ModOut"]
      </hc:Path>
    </hc:Node>
  </hc:Map>
</hc:Mapping>
<hc:Mapping>
  <hc:Map>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="AM"]/
        hc:Interface/c:Ports/c:Port[@name="InAM"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Modulator"]/
        hc:Interface/c:Ports/c:Port[@name="ModIn"]
      </hc:Path>
    </hc:Node>
  </hc:Map>
</hc:Mapping>
<hc:Mapping>
  <hc:Map>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="FM"]/
        hc:Interface/c:Ports/c:Port[@name="OutFM"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Modulator"]/
        hc:Interface/c:Ports/c:Port[@name="ModOut"]
      </hc:Path>
    </hc:Node>
  </hc:Map>
</hc:Mapping>
<hc:Mapping>
  <hc:Map>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="FM"]/
        hc:Interface/c:Ports/c:Port[@name="InFM"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Modulator"]/
        hc:Interface/c:Ports/c:Port[@name="ModIn"]
      </hc:Path>
    </hc:Node>
  </hc:Map>
</hc:Mapping>

<hc:Mapping>
  <hc:Map>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="PM"]/
        hc:Interface/c:Ports/c:Port[@name="OutPM"]
      </hc:Path>
    </hc:Node>
```

```
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Modulator"]/
            hc:Interface/c:Ports/c:Port[@name="ModOut"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="PM"]/
            hc:Interface/c:Ports/c:Port[@name="InPM"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Modulator"]/
            hc:Interface/c:Ports/c:Port[@name="ModIn"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>

    <!-- Adder -->
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="Adder"]/
            hc:Interface/c:Ports/c:Port[@name="InAdder1"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Adder"]/
            hc:Interface/c:Ports/c:Port[@name="AdderIn1"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="Adder"]/
            hc:Interface/c:Ports/c:Port[@name="InAdder2"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Adder"]/
            hc:Interface/c:Ports/c:Port[@name="AdderIn2"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="Adder"]/
            hc:Interface/c:Ports/c:Port[@name="OutAdder"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Adder"]/
            hc:Interface/c:Ports/c:Port[@name="AdderOut"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
  </hc:CapabilityMap>
```

The final step in this example is to define the connections between the resources and the physical instrument port. The two low-frequency function generators are connected to the inputs of the adder; the adder's output is connected to the modulators input; and the modulator's output is connected to the physical instrument port.

```
<hc:NetworkList>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="1"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Modulator"]/
        hc:Interface/c:Ports/c:Port[@name="ModOut"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
  <hc:Network>
    <hc:Node>
      <hc:Path>
          /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Adder"]/
          hc:Interface/c:Ports/c:Port[@name="AdderOut"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Modulator"]/
        hc:Interface/c:Ports/c:Port[@name="ModIn"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Adder"]/
        hc:Interface/c:Ports/c:Port[@name="AdderIn1"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="LF1"]/
        hc:Interface/c:Ports/c:Port[@name="LF1Out"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
  <hc:Network>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Adder"]/
        hc:Interface/c:Ports/c:Port[@name="AdderIn2"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
        /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="LF2"]/
        hc:Interface/c:Ports/c:Port[@name="LF2Out"]
      </hc:Path>
    </hc:Node>
  </hc:Network>
</hc:NetworkList>
```

An alternative approach, which may be more useful for some applications, would follow these steps:

a) Define ATML Instrument Description-based signal descriptions for every signal that the combination of low-frequency function generators plus the adder could produce. Since each of these generators can produce three different signals, this step would result in a list of six possible signals: (sine + sine), (sine + square), (sine + triangle), (square + square), (square + triangle), and (triangle + triangle).

b) Define a single resource that encompasses the functions of both function generators and the adder. Assign all six signals to that resource.

c) Define a second resource for the high-frequency modulator and assign its three signal descriptions to it, as in the F.3.10.2 example.

d) Connect the two resources together, and connect the modulator's output to the instrument's physical port.

This alternative approach uses more signal definitions but fewer resources and less complicated network connections. The choice is at the discretion of the developer.

F.3.10.3 Signal source with frequency-dependent power output use case

Sometimes, an instrument's capability will depend on some intrinsic parameter of the capability itself. This situation is illustrated by the case of a signal source whose output power capability is a function of the frequency of the signal.

Consider a source that can supply 1 V of signal amplitude from 1 kHz to 1 GHz but can supply only 0.1 V at frequencies from 1 GHz to 10 GHz. While this source can be completely described utilizing an IEEE 1641 signal description and a single resource, it may be more useful to use ATML Instrument Description resource definitions for this purpose. This description is easily accomplished through the use of multiple signal definitions: one for the lower-frequency signal capabilities and one for the higher frequencies, as shown by this XML snippet:

```
<inst:Capabilities>
  <hc:Capability name="LowFreqSignal">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutA"/>
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="lowFreqSignal" Out="OutA">
        <std:Sinusoid name="OutA"
          frequency="10kHz range 1kHz to 1GHz errlmt 0.1Hz res 1Hz"
          amplitude="trms 1V range 1uV to 1V errlmt 0.1%"/>
      </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>

  <hc:Capability name="HighFreqSignal">
    <hc:Interface>
      <c:Ports>
        <c:Port name="OutB"/>
      </c:Ports>
    </hc:Interface>
    <hc:SignalDescription xmlns:tsf="STDTSF">
      <std:Signal name="highFreqSignal" Out="OutB">
        <std:Sinusoid name="OutB"
          frequency="1GHz range 1GHz to 10GHz errlmt 0.1Hz res 1Hz"
          amplitude="trms 0.1V range 1uV to 0.1V errlmt 0.1%"/>
      </std:Signal>
    </hc:SignalDescription>
  </hc:Capability>
</inst:Capabilities>
```

This simple source would then be described using a single resource that is capable of generating either signal:

```
<inst:Resources>
  <hc:Resource name="Resource_1">
    <hc:Interface>
      <c:Ports>
        <c:Port name="Output"/>
      </c:Ports>
    </hc:Interface>
  </hc:Resource>
</inst:Resources>

  <hc:CapabilityMap>
    <hc:Mapping>
      <hc:Map>
```

```
        <hc:Node>
          <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="LowFreqSignal
          "]/
          hc:Interface/c:Ports/c:Port[@name="OutA"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
            hc:Interface/c:Ports/c:Port[@name="Output"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
    <hc:Mapping>
      <hc:Map>
        <hc:Node>
          <hc:Path>
          /inst:InstrumentDescription/inst:Capabilities/hc:Capability[@name="HighFreqSigna
          l"]/
          hc:Interface/c:Ports/c:Port[@name="OutB"]
          </hc:Path>
        </hc:Node>
        <hc:Node>
          <hc:Path>
            /inst:InstrumentDescription/inst:Resources/hc:Resource[@name="Resource_1"]/
            hc:Interface/c:Ports/c:Port[@name="Output"]
          </hc:Path>
        </hc:Node>
      </hc:Map>
    </hc:Mapping>
  </hc:CapabilityMap>
```

The resource would then be mapped to the instrument’s physical outputs, as previously shown.

If an instrument has a more complex signal description (for instance, if the power output source in the example were to be a continuous function of frequency), then the signal limits must be described utilizing IEEE 1641 signal definitions. IEEE Std 1641 [B29] allows for parameters to be described as functions of other parameters. This capability makes it possible to define very complicated interrelationships between signal parameters that cannot be described by an ATML Instrument Description.

F.4 Describing ATS capabilities

The ATML Test Equipment XML schema is included by both the ATML Test Station and ATML Test Adapter Description XML schemas. Test station and test adapter capabilities are not as easy to describe as it would appear on the surface.

In some cases, the capability of an entire test scenario (e.g., ATS and UUT) is defined by the union of the capabilities of the instruments in the test system, the signal path characteristics of all of the signal paths in the test system that connect the instruments together, and to the UUT via a test adapter, and the capabilities of the test system software.

In other cases, the capability of a test system must be defined wholly or partially independently of the capabilities of the instruments. For example, it is common for ATE station developers to intentionally restrict the type of tests that can be executed on a given ATE station.

Furthermore, an ITA can either add capabilities to the system or restrict its capabilities.

The ATML Test Station Description and ATML Test Adapter Description XML schemas support the description of capabilities for all of these scenarios. F.4.1 through F.4.6 describe test station and test adapter capability descriptions in more detail.

F.4.1 Signal path characteristics

If many of the capabilities of an ATS are described by the capabilities of the instruments, the remaining hardware-based capabilities of an ATS (the signal path characteristics) can be specified utilizing the ATML Test Station and ATML Test Adapter XML schemas (each includes the ATML Test Equipment XML schema).

To define signal path characteristics, the ATML Test Equipment XML schema provides a `<Paths>` element. The `<Paths>` element is a list of `<Path>` elements, each of which can be utilized in a ATML Test Station Description document or a ATML Test Adapter Description document to describe the characteristics of a signal path.

A `<Path>` element is conceptually a little like an instrument, only less complicated. It contains only two bits of information:

a) **Ports**: Each `<Path>` element has two `<PathNode>` elements, one for each end of the signal path. These `<PathNode>` elements are referenced in a `<NetworkList>` element to identify connections.

b) **Signal characteristics**: Each `<Path>` element includes information that specifies the electrical characteristics of the path. In other words, for practical purposes, the `<Path>` element specifies loss as a function of frequency, although it can contain information that is more specific. The `<Path>` element provides for several different methods of specifying signal path characteristics.

   NOTE—The characteristics of a signal path can also include the effects of filters or amplifiers, if desired. These devices would simply be characterized as a part of the overall path.

F.4.2 Available signal paths

The ATML Test Equipment XML schema provides a `<NetworkList>` element, which can be utilized by both the ATML Test Station Description and ATML Test Adapter Description documents, to define all of the connections within the ATS. This `<NetworkList>` element references all of the physical ports of each instrument (including switch elements), all ports on the interface connectors, and all of the connections that are made between them. Every instrument port is attached to a `<Path>` element's port (or directly to another instrument's port) via a `<Node>` element in the `<NetworkList>`.

NOTE—The `<NetworkList>` element also exists in the ATML Instrument Description XML schema and is used there in the same manor. See F.3.10.2 for an example.

In a `<NetworkList>`, not all ports have to be connected. In fact, there will typically be at least one or two ports that are not connected to anything. These are the ports where the test adapter, test cabling, or UUT will be connected.

F.4.3 Referring to instrument ports

In order to complete the description of an ATS, it is necessary to refer to instrument ports. These instrument ports are naturally defined in the ATML Instrument Description documents; in other words, they exist in different physical files from the ATML Test Station Description document or ATML Test Adapter Description document.

To accomplish this reference, the `<Node>` element (used in `<Network>`) contains an optional `documentId` attribute. To reference an element in an external file, this attribute must contain the GUID of an external document. If the attribute does not exist, then the `<Node>` element refers to the active XML instance document.

For example, a <Node> element that refers to an external document would contain a GUID in this fashion:

```
<hc:Node>
  <hc:Path documentId="{D9B428F7-E1BF-470f-8210-7DF444F8B7DA}">
    /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="1"]
  </hc:Path>
</hc:Node>
```

This example would refer to the port `@name="1"` in the instrument description document that is specified by the given `documentId` attribute.

The GUID that is specified in the `documentId` attribute is contained in the XML header of the referred-to file. It is up to the system integrator to properly manage these files so that all necessary information is available.

F.4.4 Test scenario example

As a simple example, consider a ATS made up of two instruments that (when connected together properly by Cable #1) create a signal, a third instrument that measures a UUT's response, UUT test cables (Cable #2 and Cable #3), and the UUT itself. A block diagram of this system is shown in Figure F.3.

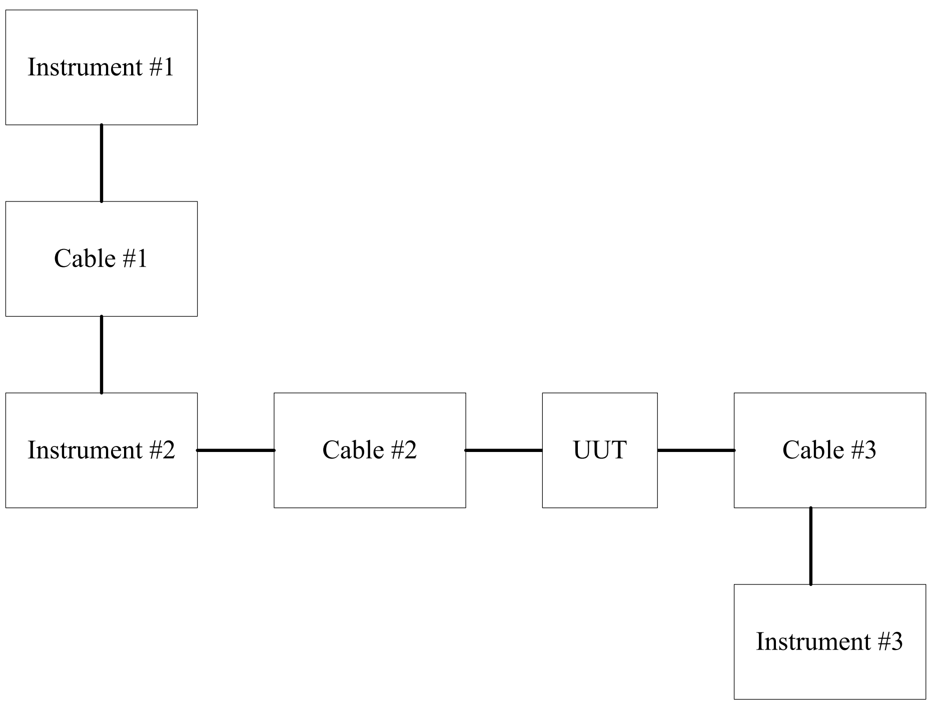


Figure F.3—Test scenario example

For brevity, the detailed descriptions of the instruments and cables will not be listed here. Likewise, the GUIDs for the external instrument description files will be abbreviated as `Instrument1FileGUID`, `Instrument2FileGUID`, and `Instrument3FileGUID`. However, port names are needed. Port names need not be unique across different instruments and cables; therefore, for this example, all ports will be named `Port1` or `Port2`.

NOTE—All cables have two or more ports. Instruments can have any number of ports, but in this example Instrument #2 has two ports while Instrument #1 and Instrument #3 only have a single port each.

The <NetworkList> element for this system would be as follows:

```
<hc:Interface>
  <c:Ports>
    <c:Port name="Port1"/>
    <c:Port name="Port2"/>
  </c:Ports>
</hc:Interface>

<hc:NetworkList>

  <hc:Network>
    <hc:Node>
      <hc:Path documentId="{Instrument1FileGUID}">
      /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="Port1"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path documentId="{Instrument2FileGUID}">
      /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="Port1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>

  <hc:Network>
    <hc:Node>
      <hc:Path documentId="{Instrument2FileGUID}">
      /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="Port2"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
      /tsd:TestStationDescription/hc:Interface/c:Ports/c:Port[@name="Port1"]
      </hc:Path>
    </hc:Node>
  </hc:Network>

  <hc:Network>
    <hc:Node>
      <hc:Path documentId="{Instrument3FileGUID}">
      /inst:InstrumentDescription/hc:Interface/c:Ports/c:Port[@name="Port1"]
      </hc:Path>
    </hc:Node>
    <hc:Node>
      <hc:Path>
      /tsd:TestStationDescription/hc:Interface/c:Ports/c:Port[@name="Port2"]
      </hc:Path>
    </hc:Node>
  </hc:Network>

</hc:NetworkList>
```

NOTE—The free ends of **Cable #2** and **Cable #3** (see Figure F.3), which define the points for UUT (or test adapter) connection, are not included in this <NetworkList>.

F.4.5 SI in test systems

The ATML Test Equipment XML schema provides for a list of components that are included in either the test station and/or test adapter. This list is separate from the <NetworkList> element that defines all of the system interconnections. Among other things, the component list allows the inclusion of SI in the ATS.

Because SI usually makes use of multiple hardware modules, a simple <NetworkList> element is not sufficient to specify the entire set of capabilities that an ATS might have. The SI capabilities rely on software that is loaded on the ATS's controller(s), and it is not explicitly included in the <NetworkList> element. However, SI is required to have ATML Instrument Description documents associated with it. Inclusion in the ATML Test Station and/or ATML Test Adapter document component list allows SI capabilities to be added to the ATS.

F.4.6 System capability specifications

The ATML Test Equipment XML schema provides a `<Capabilities>` element that is structurally identical to the ATML Instrument Description XML schema element described in F.3. The ATML Test Equipment XML schema's `<Capabilities>` element can be utilized to modify the totality of the individual instruments' capabilities to reflect the actual capabilities of the ATS. For example:

— If some instrument capabilities are intentionally prohibited from use, they can be removed from the systems capabilities.

— If the system has capabilities that are not present in any individual instrument (which is often true when system-level software adds measurement capabilities not present in any of the instruments alone), these capabilities can be added.

The rules for utilizing the ATML Test Equipment XML schema's `<Capabilities>` element are straightforward:

a) If the capabilities of the ATS are identical to the union of all the capabilities of the instruments, then the ATML Test Station and/or ATML Test Adapter documents should not include a `<Capabilities>` element. In this case, the ATS's capabilities are automatically inherited from the instruments.

b) If a `<Capabilities>` element exists in either a ATML Test Station and/or ATML Test Adapter document, then the ATS's capabilities are completely described by the data specified by `<Capabilities>`.

   1) To include all of the capabilities of an instrument, add a reference to that instrument's capabilities.

   2) To add additional capabilities, simply add them to the `<Capabilities>` element.

   3) To add some of an instrument's capabilities but not others, edit the ATML Instrument Instance document to remove the capabilities that are to be hidden.

Syntactically, ATML Test Station and ATML Test Adapter `<Capabilities>` are assembled identically to capabilities in an ATML Instrument Description, including the use of IEEE 1641-based capability definitions, user-defined resources, and ports. In this context, resources will frequently correspond to instruments but are not required to. The system integrator is free to describe test station (or test adapter) capabilities in whatever manner best suits the problem at hand.

F.5 Capability information in ATML Test Description

Specific test description test requirements are described using IEEE 1641 signals within the `ActionType/Behavior` section of an ATML Test Description.

Test requirements for any signal, e.g., sensor or measurement, can be specified as described in F.5.4 and F.5.5.

F.5.1 Overview

The following categories of capability information are supported by the ATML Test Description XML schema:

a) Signal type (example AC_SIGNAL)

b) Signal role (source, sensor, monitor, or load)

c) Signal attributes that are controlled, measured, and monitored

d) Range, resolution, and accuracy for signal attributes

e) Signal timing and synchronization

f) Signal connectivity to the pins of the UUT

This information can be specified at several levels within a ATML Test Description document, as follows:

— **Performance Characteristics**. These optional characteristics describe the performance envelope of the UUT. The information is typically generated from UUT design data. It may be used by test designers to establish the level of performance to be verified through testing. It may also be used to derive the limits not to exceed during testing in order to avoid damage to the UUT.

— **Aggregated Requirements**. These optional requirements apply to the overall ATML Test Description document. They indicate the minimum level of capability that must be provided by the ATS in order to perform **all** the tests and test groups specified as entry points in the ATML Test Description document. The aggregated requirements may be derived from the requirements of individual tests and test actions, taking into account the parallelism of various signals. They can be used to verify that a particular ATS is able to run the TPS described by the ATML Test Description document or to design an ATS that is capable of executing a given set of TPSs.

NOTE—Aggregated requirements for power signals are typically specified in test requirements documents.

— **Test Requirements**. These requirements apply to individual tests. They can be specified only for tests whose parameters and test results are described utilizing IEEE 1641 signal descriptions.

— **Test Action Requirements**. These requirements apply to individual test actions.

F.5.2 through F.5.5.3 describe in detail the capability information that can be specified at each of these levels.

F.5.2 Performance characteristics

Location: `TestDescription/PerformanceCharacteristics`

Capability information:

a) For UUT inputs[16]

  1) Nominal value

  2) Range

  3) Tolerance (i.e., accuracy[17])

  4) UUT ports

b) For UUT outputs

  1) Nominal value

  2) Range

[16] See XML schema annotations for detailed descriptions of signals and capability attributes referenced in this document.
[17] The terminology used in the XML schema originates in the main use cases of the XML schema (in this case, representing test requirements documents). In some cases, this terminology leads to inconsistencies between various sections of the XML schema (e.g., tolerance, accuracy, and error limit). To compensate, this document provides the equivalent signal terminology.

3) Accuracy
4) UUT ports

c) For UUT controls
1) Range
2) UUT ports

F.5.3 Aggregated requirements

F.5.3.1 and F.5.3.2 describe requirements for electrical signals.

F.5.3.1 Requirements for power signals

Location: `TestDescription/UUT/TestData/PowerRequirements`

Capability information:

a) For sources of dc signal
   1) Voltage
      i) Value and polarity
      ii) Tolerance (i.e., accuracy)
   2) Current—maximum
   3) Ripple—nominal, minimum, and maximum values
   4) Connection to UUT ports
b) For sources of ac signal
   1) Voltage
      i) Value
      ii) Tolerance (i.e., accuracy)
   2) Frequency
      i) Value
      ii) Tolerance (i.e., accuracy)
   3) Current—maximum
   4) Phase
   5) Phase reference
   6) Connection to UUT ports

F.5.3.2 Requirements for other signals

Location: `TestDescription/SignalRequirements`

Capability information:

a) Signal role (one of Source, Sensor, Monitor, or Load)

b) Signal type (as an IEEE 1641 TSF class, example: AC_SIGNAL)

c) For signal attributes (i.e., IEEE 1641 ac_ampl)

   1) Signal characteristic role (one of IEEE 1641 Control, Limit, Measurement, or Capability)

   2) Value

   3) Range

   4) Resolution

   5) Accuracy

F.5.4 Test requirements

Signal definitions define complete test requirements as a single signal definition. The complete signal definitions are described under the `ActionType/Behavior/IeeeStd1641` element. These signals can utilize external attribute values through input parameters (`InValues`) and results referenced through (`OutValues`).

F.5.4.1 and F.5.4.2 describe requirements for the stimulus and measurement signal specifications.

F.5.4.1 Stimulus requirements

Location: `Action/Behavior/IeeeStd1641`

Capability information (specified using IEEE 1641 signal descriptions):

a) Signal type (as an IEEE 1641 BSC signal description or as TSF class)

b) For signal attributes

   1) Value

   2) Range

   3) Error limit (i.e., accuracy)

c) Connection to UUT ports

Example 1 (IEEE 1641 BSC): Apply a BSC-defined `AC_Signal1` at `J1-1` and `J1-2` with specified accuracy for signal attribute frequency.

```
<td:Behavior>
  <td:IeeeStd1641>
    <std:Signal Out="Conn1">
      <std:Constant name="DC_Offset" amplitude="0"/>
      <std:Sinusoid name="AC_Component"
        amplitude="1.0V" frequency="1000Hz errlmt +-0.01%"/>
      <std:Sum name="AC_Signal1" In="DC_Offset AC_Component"/>
      <std:TwoWire name="Conn1" hi="J1-1" lo="J1-2" In="AC_Signal1"/>
    </std:Signal>
  </td:IeeeStd1641>
</td:Behavior>
```

Example 2 (IEEE 1641 TSF): Apply an `AC_Signal2` at `J1-3` and `J1-4` with specified accuracy for TSF class attribute frequency.

```
<td:Behavior>
  <td:IeeeStd1641>
```

```
      <std:Signal Out="Conn2">
        <tsf716:AC_SIGNAL name="AC_Signal2" ac_ampl="1.0V"
        dc_offset="0.0V" freq="1000Hz errlmt +- 0.01%"/>
        <std:TwoWire name="Conn2" hi="J1-3" lo="J1-4" In="AC_Signal2"/>
      </std:Signal>
  </td:IeeeStd1641>
</td:Behavior>
```

F.5.4.2 Measurement requirements

Location: `Action/Behavior/Ieee1641`

Capability information (specified using IEEE 1641 signal descriptions):

a) Signal/measurement type (as IEEE 1641 BSC signal description, TSF class, or generic measurement)

b) For measured signal attributes

   1) Range

   2) Error limit (i.e., accuracy)

c) For other signal attributes (specifying measurement conditions)

   1) Value

   2) Range

   3) Error limit (i.e., accuracy)

d) Connection to UUT ports

Example 1 (IEEE 1641 BSC): Measure the instantaneous voltage (`Inst1`) at `J2-1` and `J2-2`; the nominal value of the voltage is known.

```
<td:Behavior>
  <td:IeeeStd1641>
      <std:Signal Out="Inst1">
        <std:TwoWire name="Conn4" hi="J2-1" lo="J2-2"/>
        <std:Instantaneous name="Inst1" In="Conn3"
          type="Voltage" samples="1" nominal="5.0V"/>
      </std:Signal>
  </td:IeeeStd1641>
</td:Behavior>
```

Example 2 (IEEE 1641 TSF): Measure the AC amplitude (`ac_ampl`) of `AC_Signal4` at `J2-3` and `J2-4` with specified accuracy; the nominal value and range of `ac_ampl` are known; the nominal value of `freq` is known.

```
<td:Behavior>
  <td:IeeeStd1641>
      <std:Signal Out="Meas4">
        <std:TwoWire name="Conn5" hi="J2-3" lo="J2-4"/>
        <tsf716:AC_SIGNAL name="AC_Signal4"
          ac_ampl="trms 2V range 0V to 5V errlmt +- 0.01V" freq="1000Hz"/>
        <std:Measure name="Meas4" As="AC_Signal4"
          attribute="ac_ampl" Conn="Conn4"/>
      </std:Signal>
  </td:IeeeStd1641>
</td:Behavior>
```

F.5.5 Test action requirements

An alternative to specifying test requirements to complete signal definitions is to break the test requirement into a series of signal stimulus and measurement actions.

F.5.5.1 and F.5.5.2 describe requirements for the stimulus and measurement test actions.

F.5.5.1 Stimulus requirements

Locations:

```
Action/Behavior/Operations/Operation[xsi:type="td:OperationSetup"]/Source
Action/Behavior/Operations/Operation[xsi:type="td:OperationConnection"]/Signal
```

Capability information (specified using IEEE 1641 signal descriptions):

a) Signal type (in IEEE 1641 Operation, as a BSC signal description or as TSF class)
b) For signal attributes
   1) Value
   2) Range
   3) Error limit (i.e., accuracy)
c) Connection to UUT ports (in IEEE 1641 OperationConnect, as Connection signal description)

Example (IEEE 1641 TSF): Apply a DC_SIGNAL (`sig1`) at `J1-10` and `J1-30` with a specified accuracy for the IEEE 1641 TSF class attribute `dc_ampl`.

```
<td:Action xsi:type="Test">
  <td:Behavior>
    <td:Operations>
      <td:Operation xsi:type="OperationSetup">
        <td:Source>
          <std:Signal Out="sig1">
            <tsf716:DC_SIGNAL name="sig1" dc_ampl="28.0 V errlmt +- 1.0%"/>
          </std:Signal>
        </td:Source>
      </td:Operation>
    </td:Operations>
  </td:Behavior>
</td:Action>

<Action xsi:type="Test">
  <td:Behavior>
    <td:Operations>
      <td:Operation xsi:type="OperationConnect">
        <td:Signal Out="Conn1">
          <std:TwoWire name="conn1" hi="J1-10" lo="J1-30"/>
        </std:Signal>
      </td:Operation>
    </td:OPerations>
  </td:Behavior>
</td:Action>
```

F.5.5.2 Measurement requirements

Locations:

```
Action/Behaviour/Operations/Operation[xsi:type="td:OperationSetup"]/Sensor
Action/Behaviour/Operations/Operation[xsi:type="td:OperationConnection"]/Signal
```

Capability information (specified using IEEE 1641 signal descriptions):

a) Signal/measurement type (in IEEE 1641 OperationSetup, as a BSC signal description, TSF class, or generic measurement)

b) For measured signal attributes
   1) Range
   2) Error limit (i.e., accuracy)
c) For other signal attributes (specifying measurement conditions)
   1) Value
   2) Range
   3) Error limit (i.e., accuracy)
d) Connection to UUT ports (in IEEE 1641 OperationConnect, as Connection signal description)

F.5.5.3 Timing and synchronization requirements

Locations:

```
Action/Behaviour/Operations/Operation[xsi:type="td:OperationSetup"]/Monitor
Action/Behaviour/Operations/Operation[xsi:type="td:OperationConnection"]/Signal
```

The execution of test actions can be controlled through events, as summarized in the following:

a) Signal actions can be synchronized with events or gated by events.
b) Events are generated by Monitor signals in the following circumstances:
   1) When the value of an attribute of a monitored signal crosses a given threshold.
   2) With a periodic repetition rate, after a configurable initial delay. The total duration or number of repetitions can be specified.

The event mechanism can be used to specify the following requirements:

a) Signal **timing** requirements
   1) The start time and duration of individual signals
   2) Configurable time delays between signals
b) Signal **synchronization** requirements
   1) A signal action is performed after a specific condition occurs in another signal.

Annex G

(informative)

IEEE download Web site material associated with this document

This document includes supporting material required to maintain and/or develop the ATML framework as well as maintain the ATML family of standards.

This material is published by the IEEE in association with this document and presented in a machine friendly format. This material has digital rights management restricted use.

The ATML family of standards utilizes this download Web site to allow easy accessibility to these documents' XML schemas and associated material referenced within this document (e.g., examples or committee drafts).

For an explanation and the location of the IEEE download Web site and its structure (as it pertains to the ATML family of standards), see Clause 9.

Table G.1 describes the material available on the IEEE download Web site in association with this document.

Table G.1—This document's IEEE download Web site contents

File	Description
Common.xsd	The ATML common element schema defined in B.1
HardwareCommon.xsd	The ATML common element schema defined in B.2
TestEquipment.xsd	The ATML common element schema defined in B.3
Capabilities.xsd	The ATML common schema defined in C.1
WireLists.xsd	The ATML common schema defined in C.2
Extension Mechanism Example.zip	Example ATML extensions compliant with the extension mechanism defined in Clause 10.
TestAdapterDescriptionDemoV15.xml	I.3 ATML Test Adapter XML document
TestConfigDemoV4.xml	I.3 ATML Test Configuration document
TestStationDescriptionDemoV12.xml	I.3 ATML Test Station XML document
UUTDescriptionDemoV12.xml	I.3 ATML UUT Description document
AWG1_demo.xml	I.3 ATML Instrument Instance document
DCPS1_demo.xml	I.3 ATML Instrument Instance document
DMM1_demo.xml	I.3 ATML Instrument Instance document
Readme.txt	User information pertaining to the files posted, related files, and their usage

Annex H

(informative)

ATS architectures

H.1 ATS architectures utilization of published standards

ATS architectures (as described in Clause 4) may include architectural elements (e.g., additional subdomains) not supported by the ATML framework (the specific functionality ATML addresses is described in Clause 5).

The ATML framework is, therefore, a subdomain of an ATS architecture, as depicted in Figure H.1.

Figure H.1 in its entirety reflects subdomains that can be developed based upon IEEE SCC20-related standards, while Figure H.2 reflects a brief listing of other standards (either from standards organizations or consortiums) that also could be utilized in support of additional subdomains. Collectively, the subdomains of Figure H.1 and Figure H.2 may be incorporated into an ATS architecture.

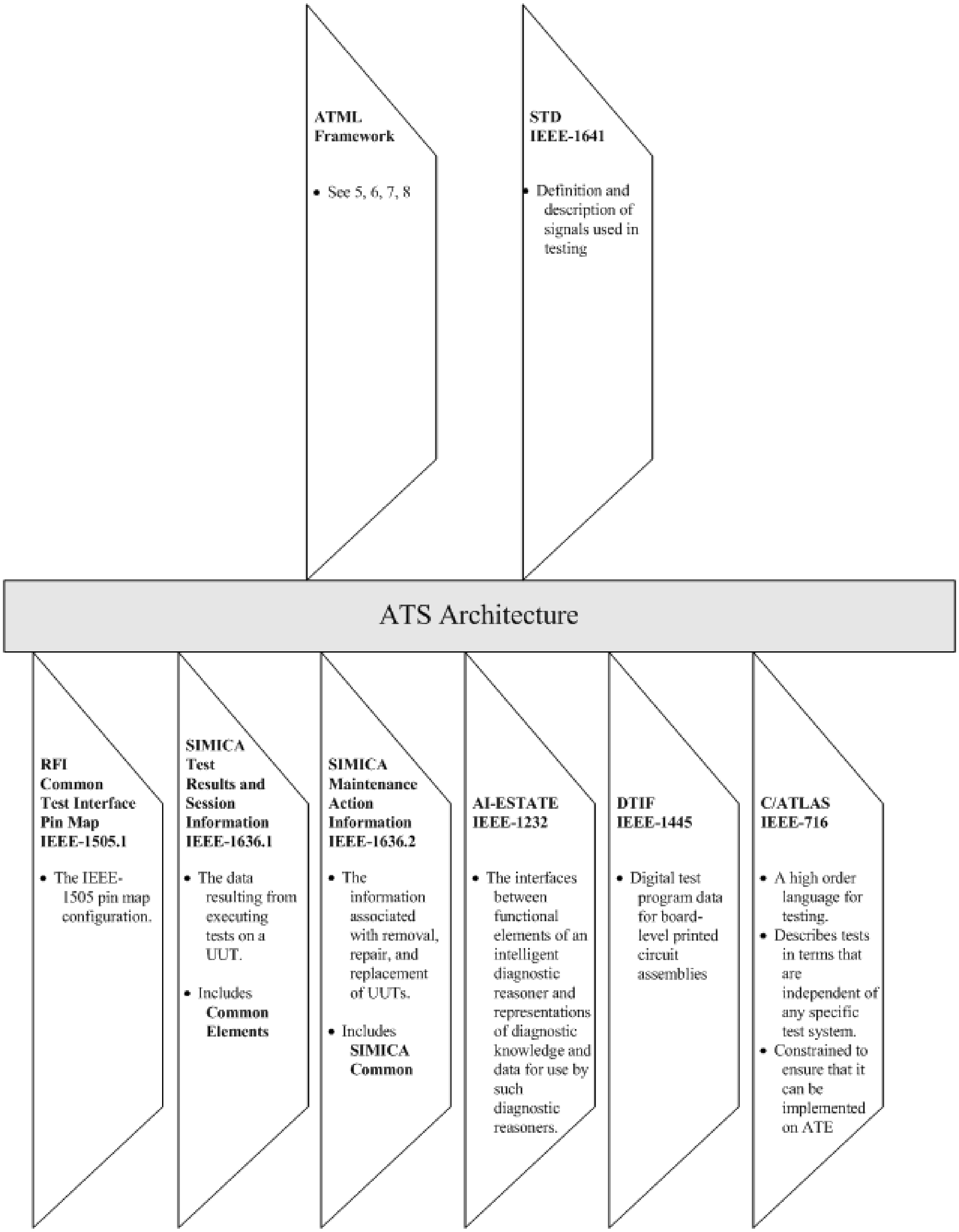


Figure H.1—IEEE SCC20-related standards

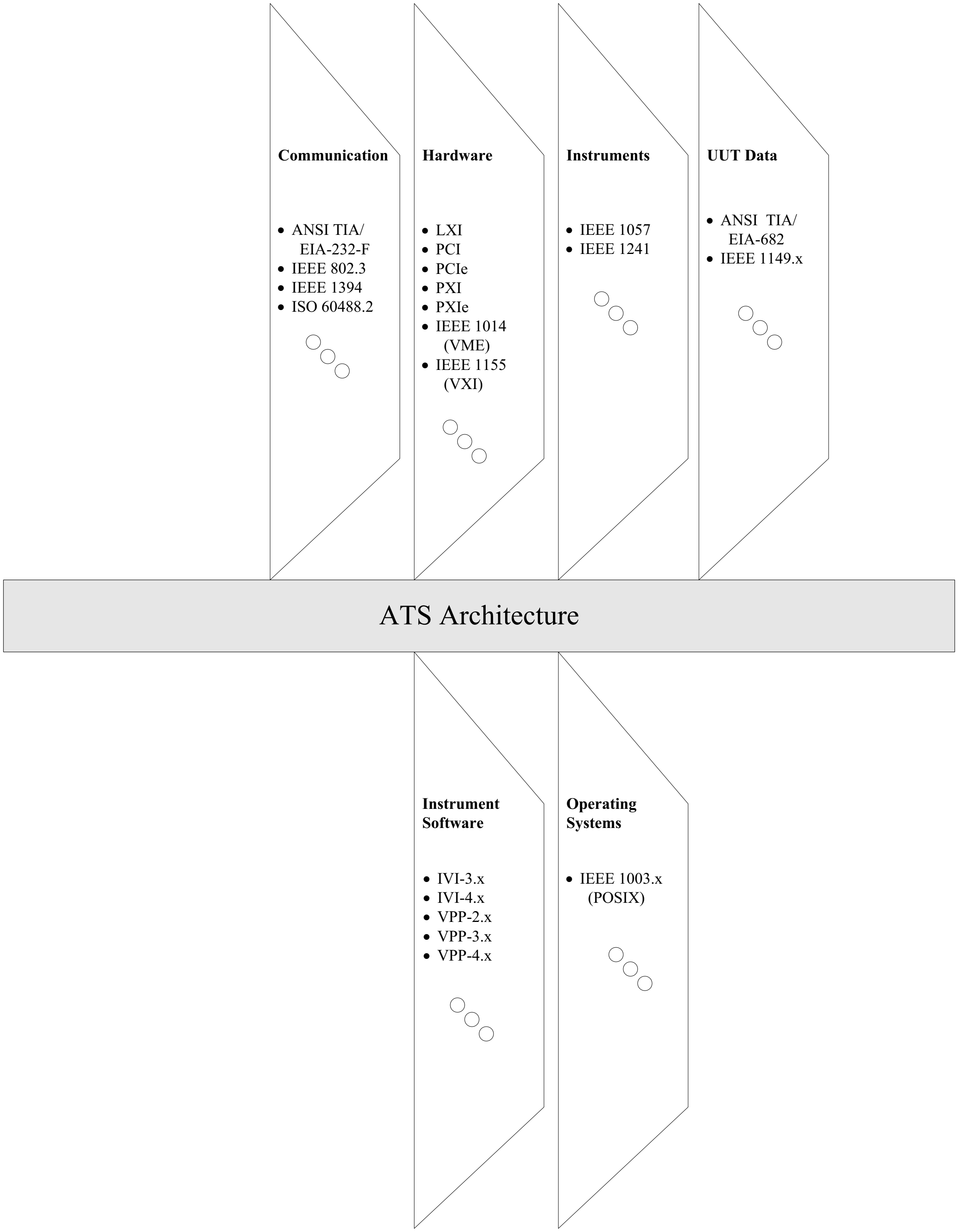


Figure H.2—ATS-related standards

H.2 ATS architectural relationships to IEEE SCC20-based standards

H.2.1 Related software standards

ATS architectures may utilize (or interface with) other software standards (as described in H.1, Figure H.1, and Figure H.2). These standards may, for example, be for an application that requires the use of diagnostic reasoners and reasoner models, ATLAS or digital-based TPSs, logging of the results of testing, and/or the collection of the actions taken during the maintenance of a UUT.

Should an ATS architecture utilize (or interface with) diagnostic reasoners and reasoner models, the application should be supported through an implementation of IEEE Std 1232™ [B20].

Should an ATS architecture utilize (or interface with) ATLAS TPSs, the application should be supported through an implementation of IEEE Std 716™-1995 [B12].

Should an ATS architecture utilize (or interface with) digital TPSs, the application should be supported through an implementation of IEEE Std 1445™ [B23].

Should an ATS architecture utilize (or interface with) the logging of the results of testing, the application should be supported through an implementation of IEEE Std 1636.1™-2007 [B27]. Should an ATS architecture utilize an ATML framework and IEEE Std 1636.1, the guidelines outlined in H.3.1 should be followed.

Should an ATS architecture utilize (or interface with) the logging of maintenance actions, the application should be supported through an implementation of IEEE Std 1636.2™-2010 [B28]. Should an ATS architecture utilize an ATML framework and IEEE Std 1636.2-2010, the guidelines outlined in H.3.2 should be followed.

H.2.2 Related hardware standards

ATS architectures may utilize ATS-related hardware standards. These standards may, for example, be for cases where the ATE incorporates the IEEE 1505™ [B24] receiver fixture interface and the IEEE 1505.1™ [B25] pin map as the ATE station's electrical I/O.

In this case, the instrument's station-level characteristics defined in IEEE Std 1505.1 should be a superset of the ATML Instrument Description and/or Test Station Description capabilities definition (e.g., ATML Instrument Description and/or ATML Test Station Description document contents shall be achievable by the IEEE 1505.1 interface pin map signal characteristics definition).

H.3 ATS architectural ATML subdomain relationship to SIMICA standards

H.3.1 Relationship to test results and session information (IEEE Std 1636.1-2007 [B27])

An ATS architecture may require a means to store the results of performing test(s) on a UUT.

Within the context of ATML, a test is any procedure for evaluating or quantifying the operation of a UUT. This test shall be an implementation of an ATML Test Description document. This implementation defines the test method that shall be utilized in order to execute the test. This test method is a definitive procedure that produces an SIMICA TestResult document.

The SIMICA TestResult XML instance document contents can be either binary or continuous (a measured or calculated value) in nature.

Within the context of ATML, the SIMICA TestResults XML schema provides a standard format for exchanging and storing the measured values, pass/fail results, and accompanying data (including test operator, station information, and environmental conditions) **associated with the test method implemented for the ATML Test Description** document-defined test(s).

An ATS architecture **needs to maintain a direct correlation between the ATML test and the SIMICA TestResult**. An example of this direct correlation is depicted by Figure H.3.

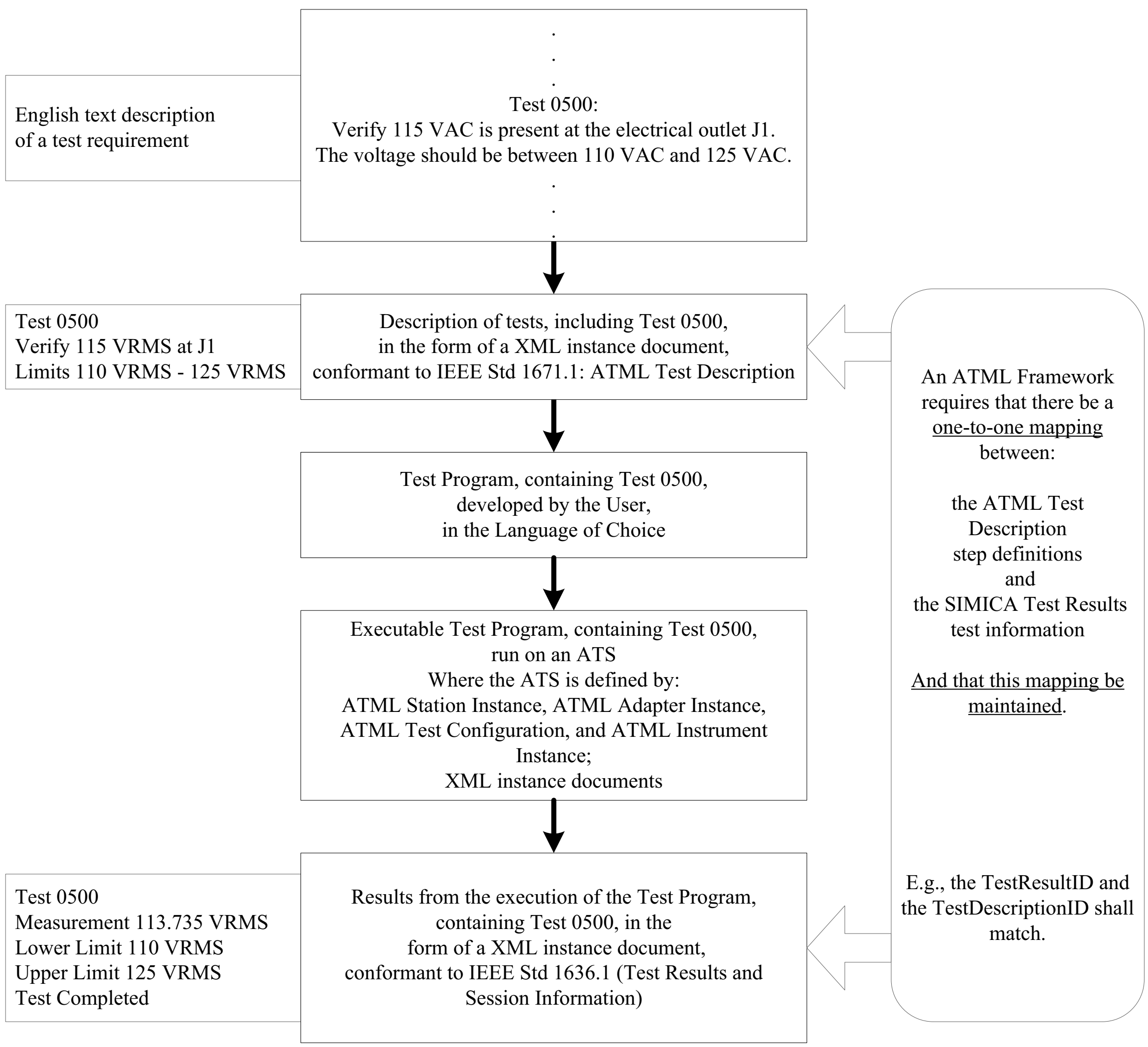


Figure H.3—Direct relationship between test results and test descriptions

Maintaining this direct correlation between the test, the test method, and the test results is vital to all interested parties (e.g., engineering, contracts) to both understand and agree upon the following:

— The methods of making measurements

— The method of obtaining the data

H.3.2 Relationship to maintenance action information (IEEE Std 1636.2™-2010 [B28])

An ATS architecture may require a means to store information relating to maintenance performed on a particular UUT.

Within the context of ATML, the collection of information associated with the actions required for the maintenance of a UUT at a specific level of maintenance or at a repair facility is historical information that should be part of the UUT description for that serial number of UUT. Therefore, this information should be included within an ATML UUTInstance (IEEE 1671.3) document.

Additionally, within the context of ATML, these maintenance action(s) should be performed only as a result of the execution of any test(s) that indicate a maintenance action needs to be taken. Therefore, there shall be a relationship between the recorded test results and the maintenance performed (recorded in a SIMICA MaintenanceActionInformation document).

The SIMICA MaintenanceActionInformation XML schema provides a standard format for exchanging and storing the information associated with the actions taken to perform maintenance **associated with the test results recorded** and to **provide history information to an ATML UUTInstance document**.

An ATS architecture **needs to maintain a direct correlation between the maintenance actions performed and the UUT test results and to support the augmentation of an ATML UUTInstance document with the maintenance action data**. An example of this direct correlation is depicted by Figure H.4.

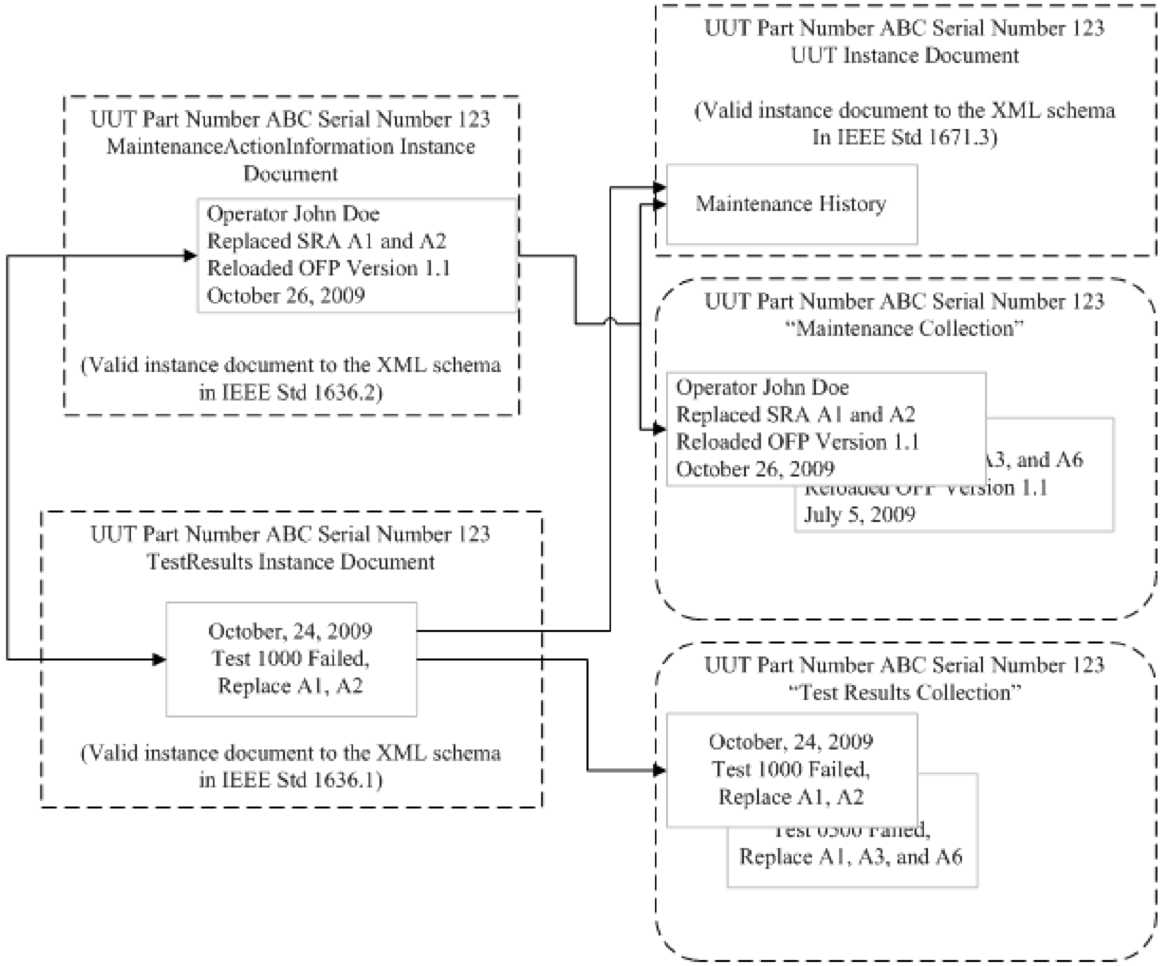


Figure H.4—Relationships—test results, maintenance information, and UUT descriptions

Annex I

(informative)

Architecture examples

I.1 Instruments

The following has been created from a real-life scenario to demonstrate the operational benefits of utilizing ATML Instrument Descriptions. The scenario of this example is that of representing an instrument's capabilities.

Typically, instrument vendors utilize datasheets (paper and/or electronic formats) as the representation of instrumentation capabilities. This document usually contains unique (to the particular vendor) descriptions of the instrument. This document may or may not be a complete description of all hardware and software capabilities.

ATE system developers, utilizing this nonstandard information, make interpretations of each of the instrument's capabilities. Each interpretation is typically manipulated to fit within the constraints of the desired ATE system description.

ATML improves upon the interpretation process by eliminating it and then using and maintaining the instrument data in the form of an ATML Instrument Description, which could be provided by the instrument manufacturer.

Figure I.1 illustrates using ATML Instrument Descriptions in support of an instrument database, rather than requiring a system integrator to interpret data sheets and populate an instrument database.

ATML Instrument Descriptions, implemented properly, can make interpreting instrument datasheets by system integrators and maintainers a thing of the past.

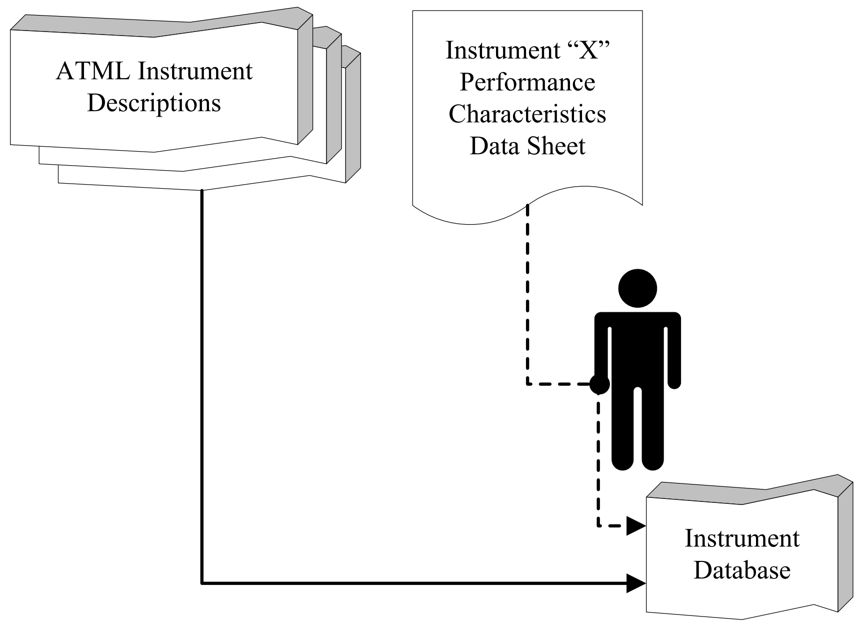


Figure I.1—Example ATML instrument usage/benefits

I.2 Test descriptions

The following has been created from a real-life scenario to demonstrate the operational benefits of utilizing ATML Test Descriptions. The scenario of this example is that of moving (either by rehosting or porting) a TPS from one ATE family to another.

Typically, TPSs are developed and integrated with a particular ATE. What this methodology introduces is station dependencies integrated within the TPS. These station dependencies can be categorized as follows:

a) **TPS instrument programming** is a historical programming technique that evolved from the manual testing methodology and is widely utilized today. The result is that a test is written around a particular instrument, its features, and the instrument's operation within that ATE. The problem with this methodology is that it is nearly impossible to replace the instrument without some level of modification to the written tests (up to and including a total rewrite).

b) **Instrument hardware electrical characteristics** are electrical characteristics that must be taken into account when interfacing instruments with the UUT. For example, impedance, drive capability, resolution, and accuracy are capabilities typically found on instrument data sheets. However, there are characteristics that are quite often overlooked, and not documented, such as timing (e.g., 3 µs after a trigger, the waveform is present) or firmware impacts (e.g., the instrument resets to 0 V between steps of a ramped waveform output). These overlooked characteristics quite often have to be reverse engineered for each instrument and ATE configuration.

c) **Instrument programming interfaces** include the instrument's setup time and the time to command the instrument. These times affect the overall TPS execution times as well as individual test timing. Regardless of the interfaces used within the ATE, TPSs were developed with this specific timing integrated within. This timing represents not only the actual hardware interface, but also the controller execution times and software overhead.

It is important to recognize that at some point in time, the tests that were developed and integrated on a specific ATE were implementing a test strategy for the particular UUT.

The philosophy of ATML Test Description is to support the reuse and maintenance of the test requirements for the particular UUT. No longer is the reuse of the implemented tests (and addressing the issues discussed above) necessary. When a new ATE is required, rather than reverse-engineering the existing TP, a new TP is developed (in the programming language of choice), is targeted at the new ATE (and its capabilities), and utilizes the test strategy captured by an ATML Test Description.

ATML tool developers have tool sets that can extract test requirement information from existing ATLAS-based TPs (for use when the test strategy information may no longer be available) as well as support the development of TP source code from Test Description.

Figure I.2 illustrates using the original test requirement (which would be an ATML Test Description) to develop implementations (each aimed at the capabilities of the target ATE) rather than interpreting a specific implementation (Implementation #1) to adapt the new ATE to the old code (Implementation #2). ATML Test Descriptions, implemented properly, make adapting legacy TPS code, without going back to the original test requirement, a thing of the past.

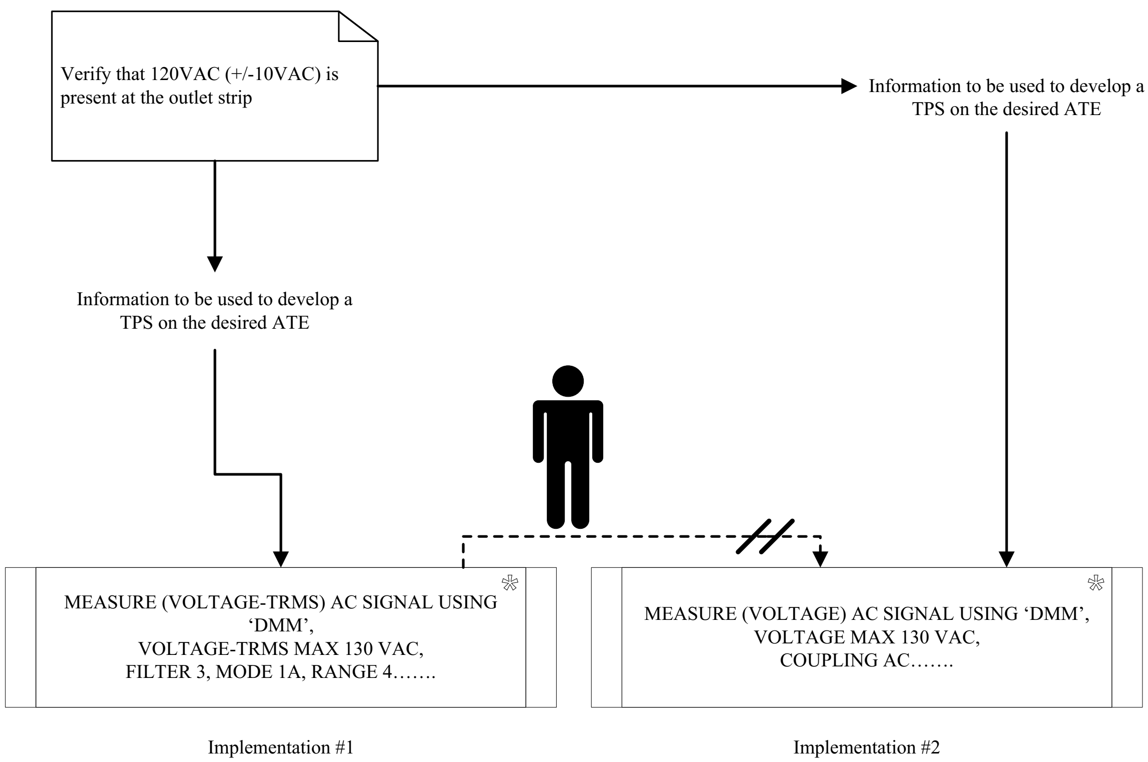


Figure I.2—Example ATML Test Description usage/benefits

I.3 Complete testing scenario

I.3.1 Introduction and objectives

The following example is derived from ATML demonstrations. These demonstrations successfully showed the operational benefits of utilizing ATML standards in a real testing scenario. Using ATML family XML instance documents and COTS software products, a UUT was successfully tested and diagnosed and subsequently rehosted across multiple ATE platforms.

This example outlines a scenario where three system-level use cases (see I.3.1.1) are developed utilizing a majority of the ATML family of standards (see I.3.1.2) and describes both the hardware and software design and development tasks involved with implementing the testing scenario (see I.3.2 through I.3.3).

I.3.1.1 System-level use cases

The example system-level use cases are to

a) Provide all the necessary test information, in a reusable format, to take a Test Description in the form of UUT Test Requirements and convert it into a TP (in an ATE-oriented language) running on a fielded ATE system.

b) Use ATML family component standards to compare requirements and capabilities as well as calculate switching paths.

c) Collect test results and use them in various use cases such as for statistical analysis and display to the operator.

I.3.1.2 Utilized standards

This example is based around the ATML family of standards. Specifically, the following IEEE standards are included in this example:

a)	IEEE Std 1671	ATML Overview and Architecture
b)	IEEE Std 1671.1 [B31]	ATML: Test Descriptions
c)	IEEE Std 1671.2 [B32]	ATML: Instrument Descriptions
d)	IEEE Std 1671.3 [B33]	ATML: UUT Description
e)	IEEE Std 1671.4 [B34]	ATML: Test Configurations
f)	IEEE Std 1671.5 [B35]	ATML: Test Adapter Descriptions
g)	IEEE Std 1671.6 [B36]	ATML: Test Station Descriptions
h)	IEEE Std 1636.1-2007 [B27]	SIMICA: Test Results and Session Information
i)	IEEE Std 1641 [B29]	Signal and Test Definition (STD)

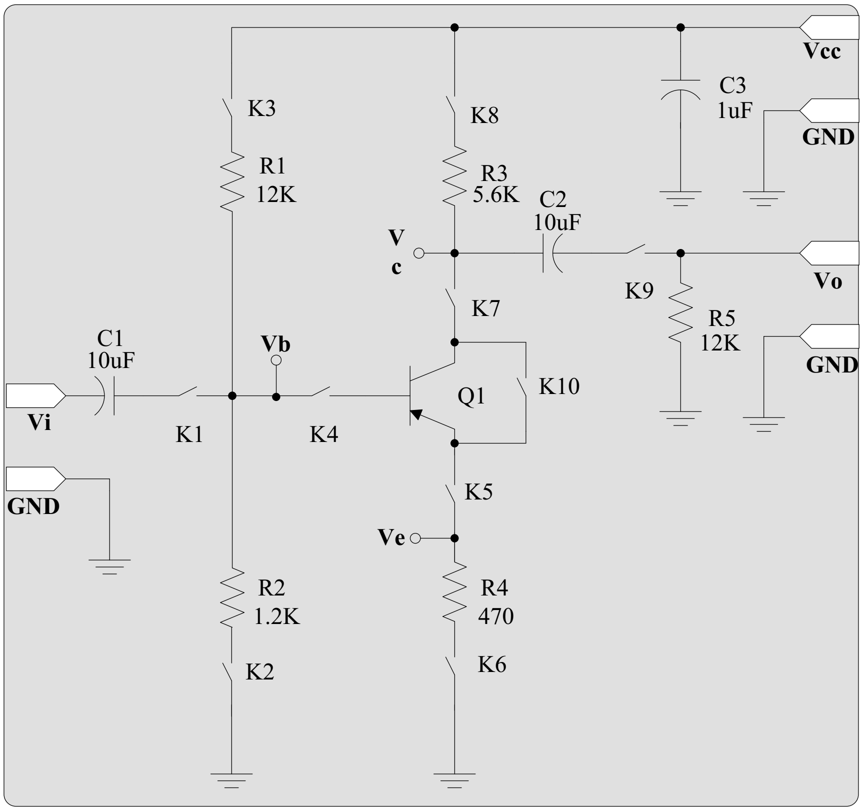
I.3.2 Hardware test environment and UUT design

The hardware elements designed, chosen, and utilized (UUT, ATE, ITA) for this example are described in I.3.2.1 through I.3.2.3.

I.3.2.1 UUT hardware and test strategy

For the purpose of this example, a simple low-frequency analog UUT is designed and developed. This dedicated UUT allows both parametric testing and diagnostics through fault simulation, allows for the generation of an ATML UUT Description document, permits the test execution on a variety of ATE stations (UUT testing requires only commonly available stimulus and measurement resources), and allows faults to be inserted during execution of tests through the activation of switches incorporated into the UUT design.

The UUT schematic and connector pin definitions are depicted in Figure I.3. The UUT connector (and the mating connector that will be part of the ITA) itself is a commercially available D connector containing both low-frequency signal and coaxial contacts.



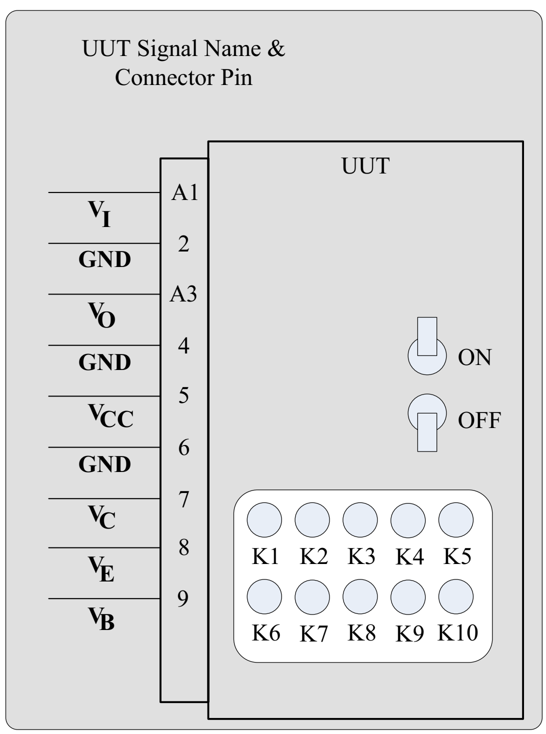


Figure I.3—UUT's schematic and UUT's connector pins

The testing and fault isolation strategy of this UUT is depicted in Figure I.4. This strategy is utilized for the ATML Test Description generation as well as during the ITA design (and thus the associated ATML Test Adapter Description generation).

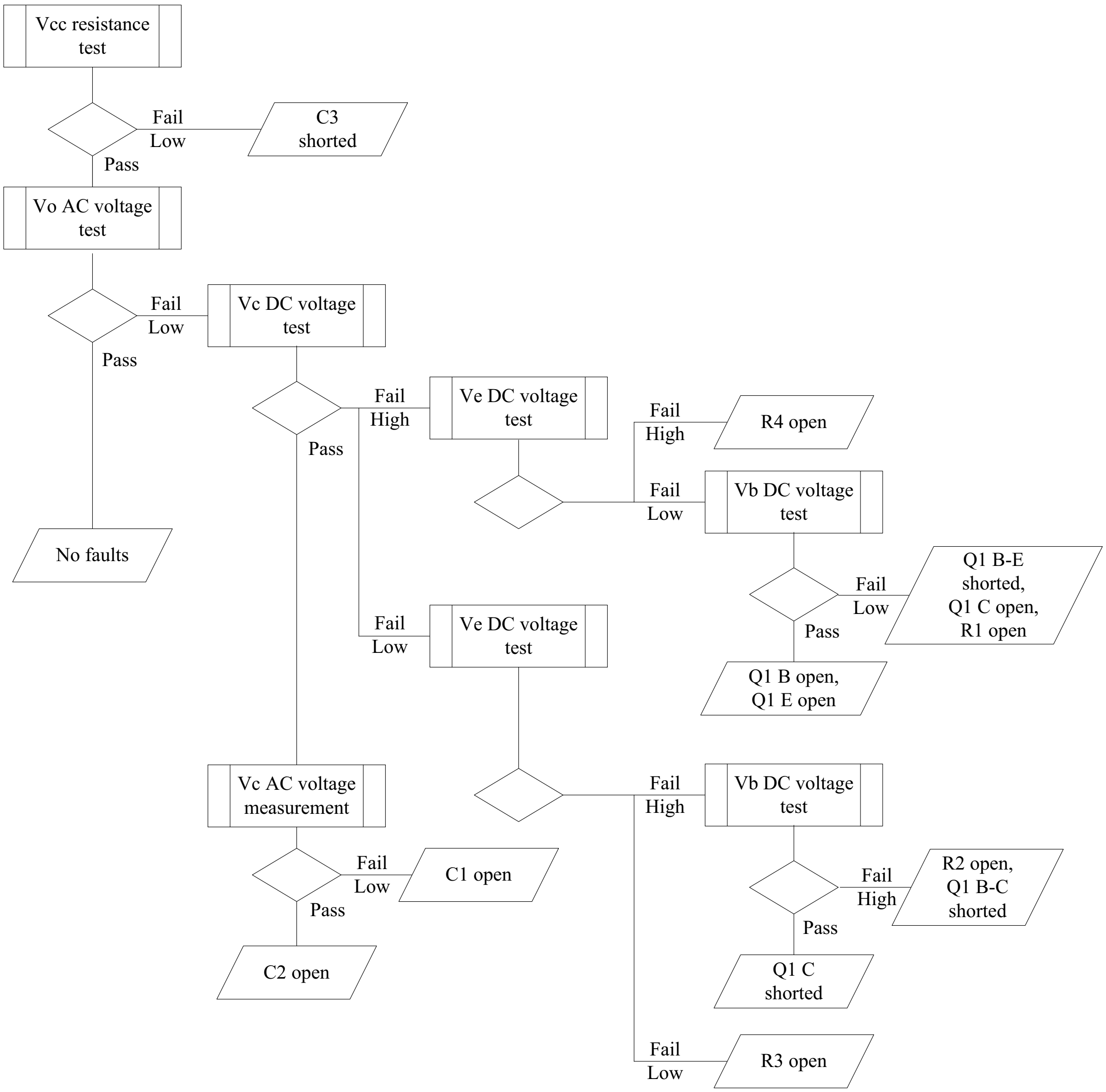


Figure I.4—UUT's testing strategy

The UUT is defined within an ATML UUT Description. This ATML UUT Description document is available at http://standards.ieee.org/downloads/1671/1671-2010.

NOTE—This UUT is also represented in Annex B of IEEE Std 1671.1-2009 (ATML Test Description); further detail on the UUT can be found in that annex.

I.3.2.2 ATE and test configuration

An ATE was chosen from the DOD approved family of ATE (see DOD ATS Master Plan [B8]), although any target platform test station could be used to support both development and the full demonstration which provides the station test resources and supports the types of tests (this assumes however that the target platforms electrical interface is identical, or that the ITA could either be mechanically and electrically adapted to the target platform or developed specifically to the target platform).

The necessary station test resources are those for:

a) UUT power.

b) Low frequency signal source.

c) Simple analog measurements.

d) Measurement analysis.

e) Switching.

These test resources as well as the station control is portion of the ATE is represented in Figure I.5.

The chosen ATE contains a well-documented station interface (both hardware and signal) pin-out definition, has a station control that is PC based, is running a widely used COTS operating system (allowing the ATML tools to be easily hosted and execute on the ATE itself), and contains the necessary test resources to provide the stimulus and measurement capabilities required to implement the TP that will be derived from the ATML Test Description (see I.3.2.1) in a ATE-oriented language. The stimulus and measurement resources are electrically accessible directly as feed-throughs at the station's interface receiver. This ATE additionally contains low-frequency, coaxial, and power switching, which permits the test assets to be routed to multiple locations as required by a testing implementation, within the ITA.

With a general understanding of the UUT described in I.3.2.1, and a general understanding of the chosen ATE, a ITA can then be designed and developed to interface the two (see Figure I.5 and I.3.2.3).

The utilized portions of the ATE are defined within an ATML Test Station Description document. This ATML Test Station Description document is available at http://standards.ieee.org/downloads/1671/1671-2010.

Each of the utilized instrument's functions is defined a within ATML Instrument Description documents. Each of these ATML Instrument Descriptions documents is available at http://standards.ieee.org/downloads/1671/1671-2010.

The configuration depicted by Figure I.5 (and the software installed on the station controller, e.g., operating system, ATML tools, and COTS software such as the ATE control software, ATE support software, and ATE system software) is defined within an ATML Test Configuration document. The ATML Test Configuration document is available at http://standards.ieee.org/downloads/1671/1671-2010.

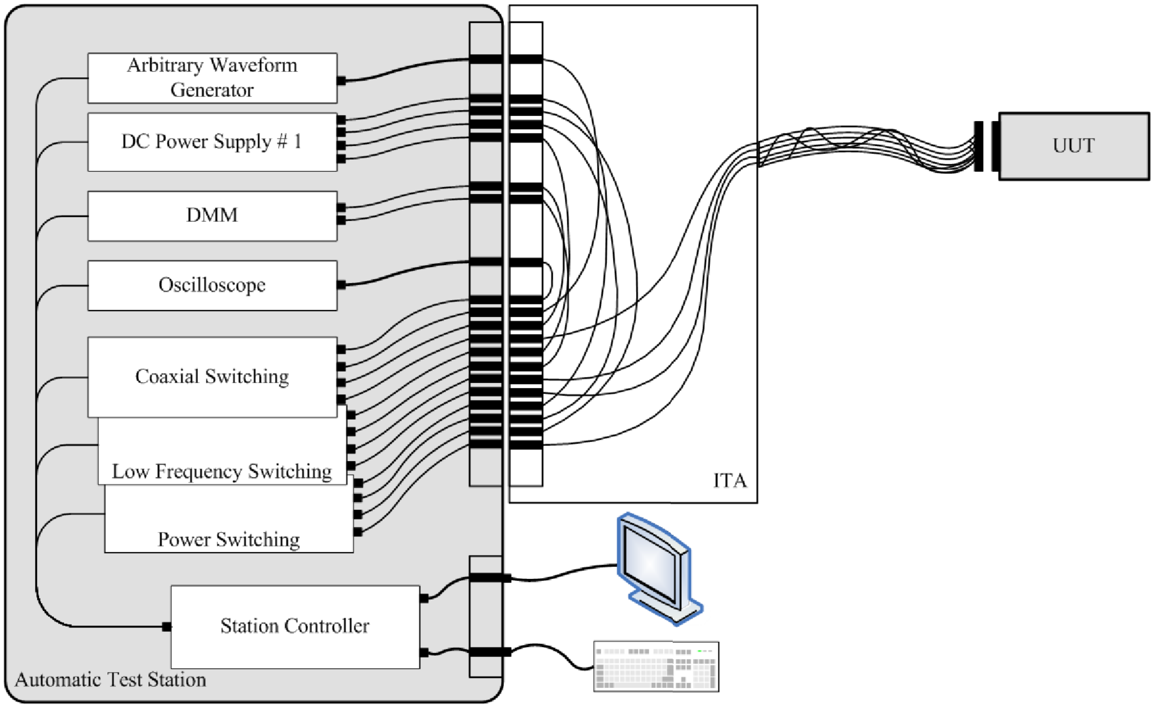


Figure I.5—UUT's hardware testing configuration

I.3.2.3 Interface test adapter (ITA)

For the purpose of this example, an ITA is designed and developed to electrically interface the UUT (see I.3.2.1) to the chosen ATE (see I.3.2.2).

Utilizing the ATE's stimulus, measurement, and switching capabilities on one hand and the UUT's connector/mating connector pin-out and test strategy on the other hand, an ITA concept can be developed. Each of the tests in the strategy is manually mapped to test resources (through the utilization of switch paths, which allow for multiple resources to be routed to the same UUT pin or for resources to be totally disconnected, for example). The completion of each of the test strategy mappings to ATE resources, with each mapping overlaid on each other, results in the ITA interconnections depicted by Figure I.6.

NOTE—In developing the ITA concept, the designer has mentally assigned resources and utilized switch paths to route signals to/from the UUT connector pins so that they know that each test within the testing strategy can be electrically connected. This knowledge is obviously not known by the ATE station software unless there is a method of providing the information to the software. Thus can be seen the importance of IEEE Std 1671.5 [B35] and the contents of the associated XML instance document representing the ITA.

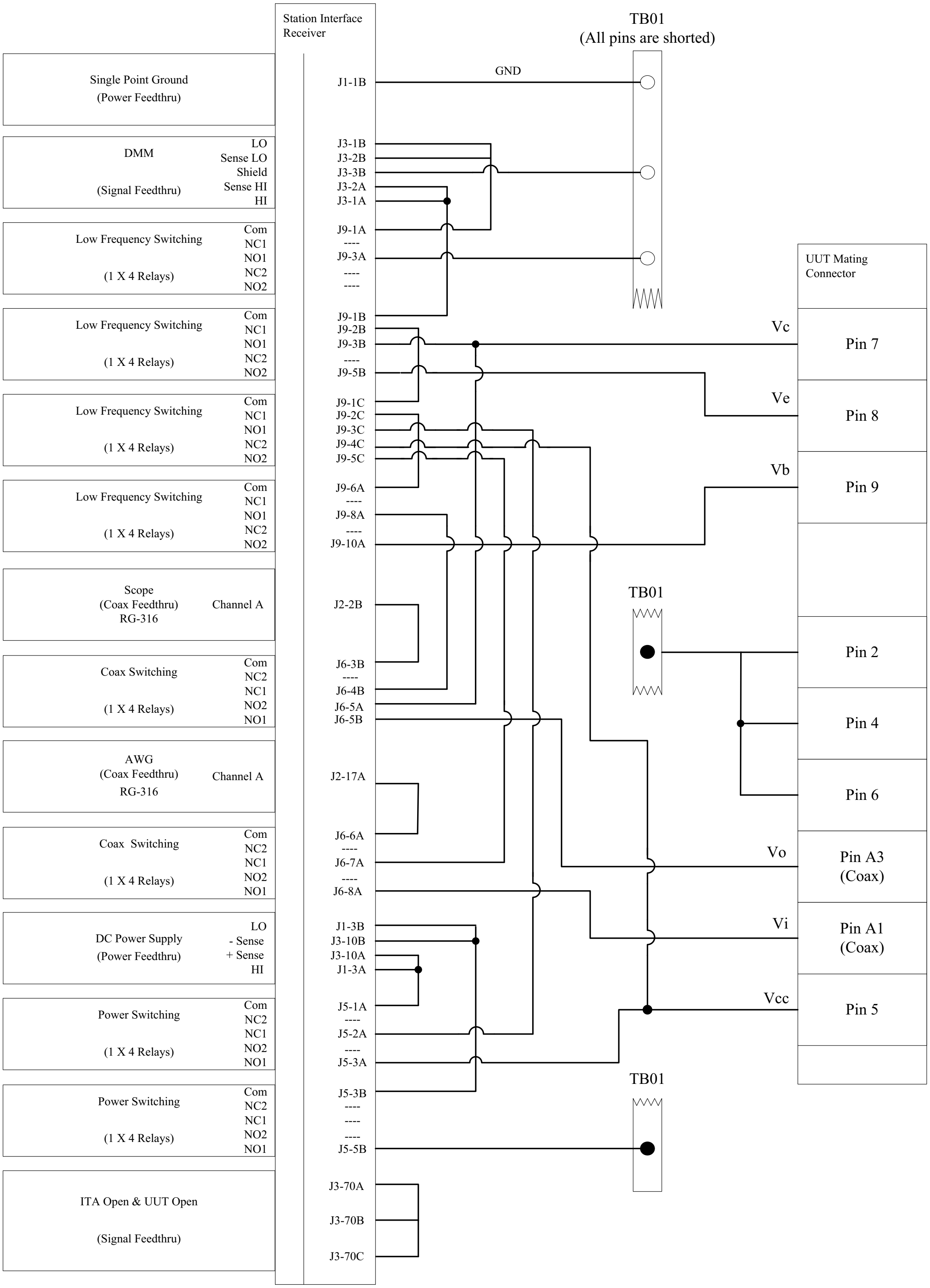


Figure I.6—ITA interconnections—ATE resources to UUT mating connector pins

The ITA is defined within an ATML Test Adapter Description document. This ATML Test Adapter Description document is available at http://standards.ieee.org/downloads/1671/1671-2010.

I.3.3 Software test environment design

The software environment that is utilized for the development and execution of the UUT's TP is depicted by Figure I.7. All nonshaded elements of Figure I.7 represent ATML family standards. Shaded elements represent tools, COTS software, and software outputs from the execution of tools.

I.3.3.1 through I.3.3.2 further detail the ATE support software, ATE system software, and the ATE control software. For the purpose of the example, the TPS will be a C/C++ program.

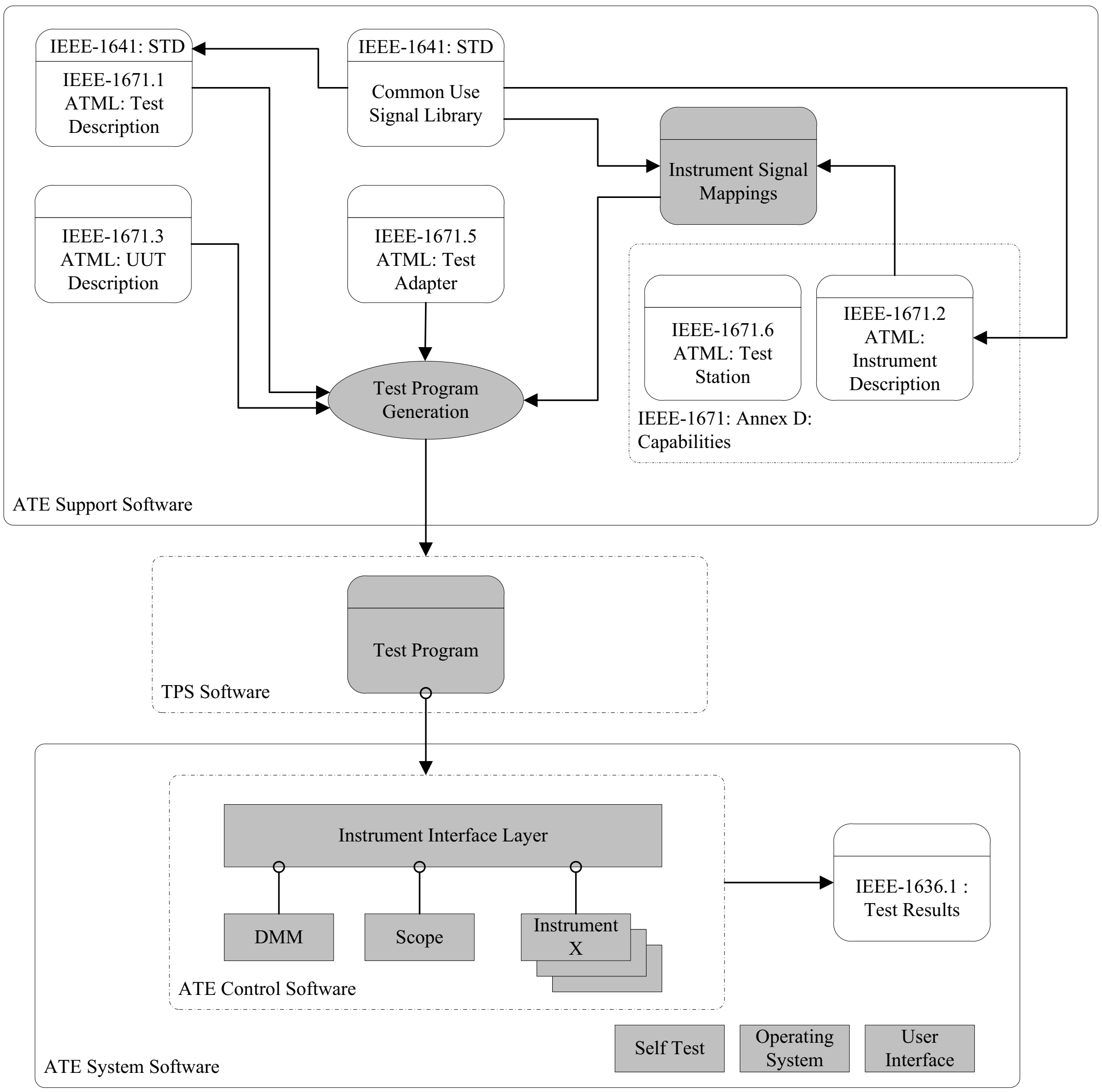


Figure I.7—UUT's software testing configuration

I.3.3.1 ATE support software

ATE support software consists of the software that aids in the preparation, analysis, and maintenance of UUT TPs. The ATE support software typically is available off station; however, there may be instances where the ATE support software is also available on the target ATE. Within the scope of this example, the location of the ATE support software is irrelevant.

In order to actually conduct tests on the UUT (see I.3.2.1) utilizing the hardware items (ATE and ITA) described in I.3.2, a UUT TP needs to be developed from the UUT testing strategy (see Figure I.4) so that it may be executed by the ATE station control software.

Historically, the elements of a particular ATE's support software incorporated interpretations by the ATE developers of such items as instrument's vendor data sheets/documentation, ATE hardware design material, etc. These interpretations are effectively turning one data format into a second (usually proprietary) format (e.g., an instrument's data sheet contents in PDF format put into a compiler's instrument database's unique file format). This process usually always loses something in the translation as information is lost, does not have a home, etc. Thus can be seen the importance of IEEE Std 1671.6 [B36] and IEEE Std 1671.2 [B32] and the contents of the associated XML instance documents that can be shared between vendors and ATE organizations; they eliminate any need to interpret or to utilize proprietary formats.

For this example, COTS software products (see I.3.3.1.1) that utilize/incorporate ATML XML instance document data and STD-based (IEEE Std 1641 [B29]) signal modeling concepts (see I.3.3.1.2) can be assembled to allow for the development of an ATE-oriented language TP for the UUT.

I.3.3.1.1 COTS software products

This example is reliant on the use of COTS tools already developed to create and consume the ATML information as part of the operational ATS. Prototype COTS tools are acceptable as long as documentation or support is available and the tools comply with the conformance section of this standard. ATML tools may consist of any application that produces, translates, or consumes the ATML format (e.g., editors, display tools, converters, database). In this technological area, it is expected that a wide range of diversity and innovation will occur within COTS products, while supporting the interchangeable ATML format.

The COTS software products included in the ATE support software are the following:

a) STD-based signal modeling tools
b) ATML test and instrument description tools
c) Test executive ATML importer tools
d) Switch path analysis tools
e) C/C++ language (see ISO/IEC 9899:1999 [B40]) compilers and linkers

I.3.3.1.2 ATML and STD signal modeling data

The ATML information utilized by these COTS software products includes the following:

a) STD-based signal model libraries
b) ATML test description
c) ATML instrument description
d) ATML test station description

e) ATML test adapter description

f) ATML UUT description

g) ATML capabilities

h) ATML test configurations

This support software architecture, in the form of an information flow diagram, is depicted by Figure I.8. Various tools (as depicted by the shaded oval shape) are utilizing various ATML and STD information/libraries (as depicted by the shaded shape).

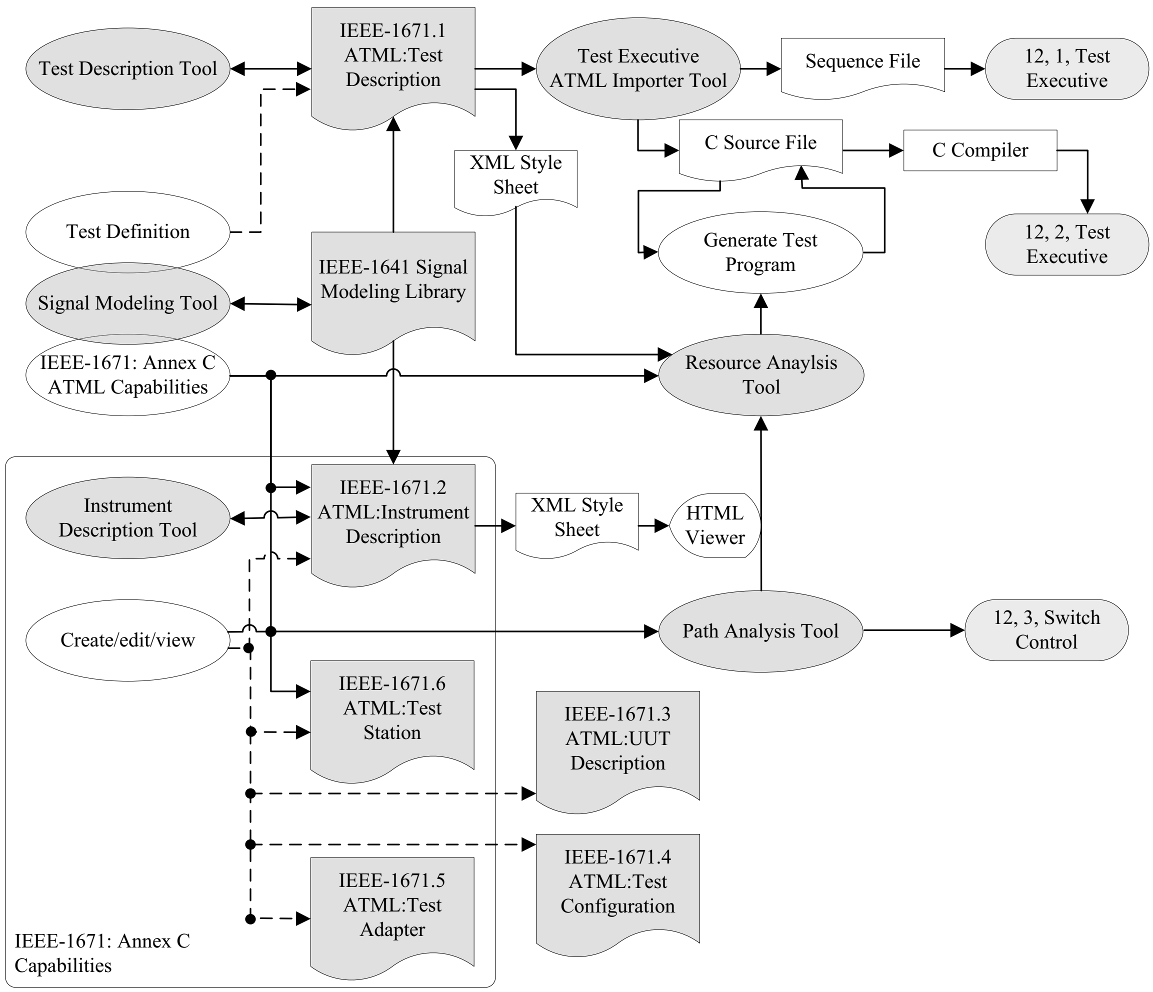


Figure I.8—Support software architecture

ATML Capabilities are used as the **glue** to allow the UUT's signal requirement to be mapped to the ATE test resource (ultimately to the ATE connector pins) that can provide the capability through the ATE and ITA to the required UUT connector pin. This mapping is depicted by Figure I.9.

The ATE support software provides for the creation, editing, and viewing of this **glue**.

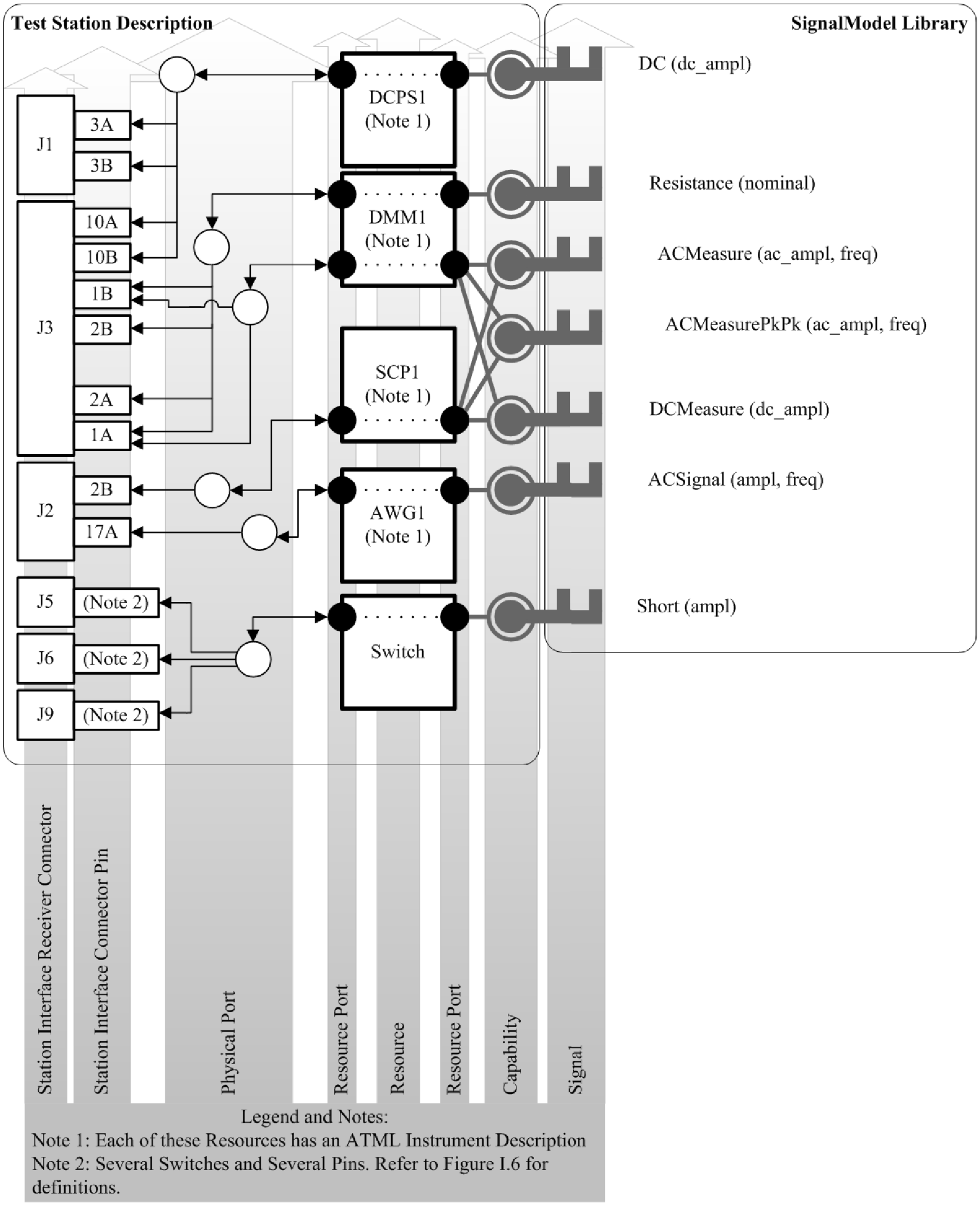


Figure I.9—Resource capabilities

I.3.3.1.3 Support software outputs

The outputs of the tools in Figure I.8 collectively result in the following, which will be utilized by the ATE control and system software (see I.3.3.2). These three outputs are the three inputs of Figure I.10:

a) A sequence file for use by the test executive

b) The generation of the TP, which is then compiled and linked with a C/C++ compiler (see ISO/IEC 9899:1999 [B40])

c) Switch control information

XML style sheets and HTML viewers display to the user the XML information.

I.3.3.2 ATE control software and ATE system software

ATE control software is used during the execution of a UUT TP to control the nontesting operations of the ATE. This software executes a test procedure, but does not contain any of the stimuli or measurement parameters used in testing the UUT.

ATE system software is the total software environment of the ATE including operating system, test executives, user interface, system self-test, and other software required to run UUT TPs. For the purposes of this example, the PC's operating system, ATE system self-test, and user interface will not be further described.

In order to run the tests on the UUT, utilizing the ATE and ITA, the UUT TP created via the ATE support software (see I.3.3.1) needs an environment from which to be executed.

Historically, the elements of a particular ATE's station control software interfaced with and produced data in proprietary formats (e.g., TP intermediate programming languages, test results). In other words, only the matching ATE support software could be utilized, and test results were represented/stored in a format unique to that ATE. ATML **permits** (as depicted in I.2) **the actual TP to be implemented in the language of choice**. Therefore, the complementary test execution environment for that language of choice must be an element of the ATE station control software. Thus can be seen the importance of the ATML reference to the IEEE Std 1636.1-2007 [B27] and the contents of the associated XML instance documents that can be shared between vendors and ATE organizations; they eliminate any need to utilize proprietary formats for the recording of UUT test results.

For this example, COTS software products (see I.3.3.2.1) that utilize/incorporate standards (see I.3.3.2.2) can be assembled to execute an ATE-oriented language TP for the UUT, create and view UUT test results, and provide an interface to reasoner-based test execution.

I.3.3.2.1 COTS software products

This example is reliant on the use of COTS tools already developed to create and consume the ATML information as part of the operational ATS. Prototype COTS tools are acceptable as long as documentation or support is available, and the tools comply with the conformance section of this standard. ATML tools may consist of any application that produces, translates, or consumes the ATML format (e.g., editors, display tools, converters, database). In this technological area, it is expected that a wide range of diversity and innovation will occur within COTS products, while supporting the interchangeable ATML format.

The COTS software products included in the ATE control software are the following:

a) Test executive

b) Switch path control

c) Instrument support handlers

d) Instrument drivers

e) Virtual instrument software architecture

f) Bayesian reasoner

g) Test results analyzers and viewers

I.3.3.2.2 AI-ESTATE models

The ATML information utilized by these COTS software products included an Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE) Bayesian reasoned model.

This station control software architecture, in the form of an information flow diagram, is depicted by Figure I.10.

As can be seen by Figure I.10, the graphic visualization and test result analyzer tools (as depicted by the shaded oval shape) are utilizing test results information (as depicted by the shaded shape).

I.3.3.2.3 Station control software inputs

The inputs of the tools in Figure I.10 collectively result from ATE support software outputs (see I.3.3.1). These three inputs are the three outputs of Figure I.10.

a) A sequence file for use by the test executive

b) The compiled and linked C/C++ language UUT TP

c) Switch control information

XML style sheets and HTML viewers display to the user the XML information.

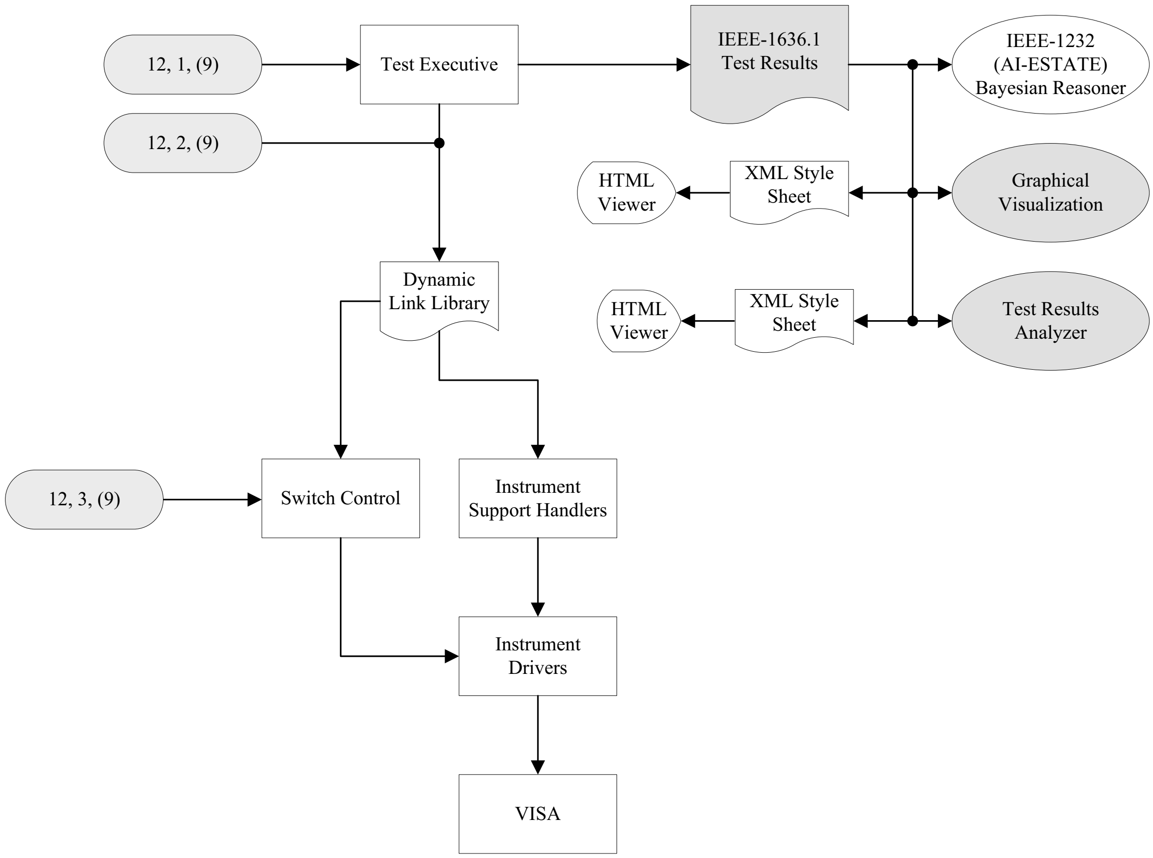


Figure I.10—Station control software architecture

I.4 Integrated ATML system

I.4.1 Introduction

This example has been derived from the AUTOTESTCON 2007 paper "A Proposed Comprehensive Architecture Utilizing the Automatic Test Equipment Markup Language" [B49].

In a typical ATS, a fixed set of equipment (e.g., signal generators, digitizers) is interconnected through one or more software-controlled switches. TPs are executed on one or more controlling computers, which send signals and commands to this equipment. These commands configure signal pathways for stimulus and response and set signal and power parameters for the equipment.

In an integrated test system fully utilizing the ATML framework, it is presumed that each instrument in the test system is to be described in an ATML InstrumentInstance document. Additionally, the test station will have a ATML TestStation document, and each UUT will have an associated ATML UUTInstance document.

In order to conduct tests, this notional system will use a ATML TestDescription document in combination with the other XML instance documents to create the runtime software necessary to exercise the test system and conduct tests. As testing proceeds, the system will produce SIMICA TestResults documents to capture the measurements and results produced by the system.

The advantages this system provides over existing, traditional test systems are many. Use of a test description document eliminates the need to specify in software the "how" of a test; it is sufficient to depict "what" is being tested. The UUT instance XML document describes the UUT's input ports, including power levels, voltages, and expected signals. Likewise, UUT output ports are specified along with the normally expected signals and upper and lower boundaries for each port. A test system software executive capable of ingesting these documents will match the "what is being tested" against the capabilities of the instruments installed in the station to determine whether the current station configuration is compatible with the requirements of the UUT. ATML InstrumentInstance documents will be provided by each instrument manufacturer to enable modular system architecture; in other words, the test system is not "locked in" to a single vendor for any instrument component. Likewise, the manufacturer of each UUT will be responsible for characterizing its UUT in a ATML UUTDescription document.

I.4.2 Integrated system

A system that maximizes use of these XML schemas could consist of several subsystems. This level of modularity is enabled through use of the well-defined data exchange format provided by XML schemas. A conceptual design for such a modular system is shown in Figure I.11. Equipment vendors would provide "description" documents appropriate to their products. A system integrator/developer would create the configuration and "instance" documents along with test description documents. All of these XML files would be available in a shared data system. The test executable creator component would retrieve necessary XML instance documents and generate executable instructions. During test execution, the test execution system would retrieve the execution instructions and pass them to the test station instruments. Measurement data would be returned as a SIMICA TestResults document for storage. These results could also be passed to a diagnostic system, which would return diagnoses for action by the operator. This modular design would allow the implementer to design each component as a "black box" with simple service-oriented interfaces.

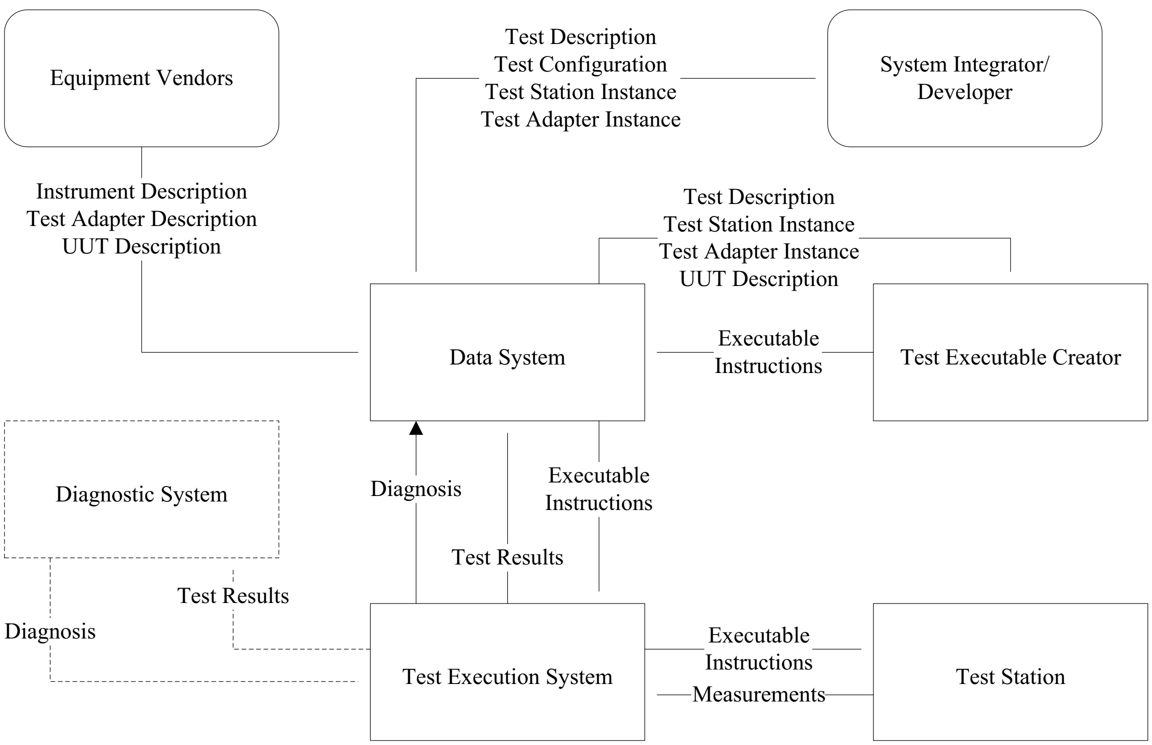


Figure I.11—Conceptual system block diagram

I.4.3 Test system implementation

Any test system requires a set of instruments capable of meeting the I/O requirements of the device(s) to be tested, i.e., the UUT.

In an ATML-oriented world:

— The original equipment manufacturers would document the UUT using the standard format and nomenclature of UUT description.

— TP engineers would document the UUT testing requirements using the standard format and nomenclature of test description.

— Instrument vendors would document the capabilities of their instruments using the standard format and nomenclature of instrument description.

A system integrator would then simply match all of the UUT requirements against the instrument capabilities in order to properly equip and configure a test system. The system integrator would then produce a test station document that would be used as the basis of a test station instance document. This instance document would probably be uniquely associated with a single UUT. If test adapter hardware is necessary to interface the UUT with the station, one or more test adapter instance documents may be produced to refine the information provided in the test adapter description document. In addition to instrument description documents, an instrument vendor may also provide software drivers for their equipment.

I.4.4 Test development

The schema for test description provides a fundamentally new approach to the development of automated tests. Typically, current systems utilize procedural languages that require a test developer to explicitly specify instruments, input signals (e.g., waveforms), signal switching paths and I/O ports on the UUT. Simply put, it is necessary to specify “how” to conduct a test. In an ATML environment, this level of detail is abstracted into a set of “what to test” statements. In essence, this approach reduces a set of instructions such as

— Prepare the signal generator.

— Load waveform x to signal generator.

— Switch signal generator output to station output port z.

— Associate port Z with UUT input port j.

— Prepare to measure on station port a.

— Associate station port a with UUT port k.

— GO.

to a much simpler set of instructions:

— Measure the output on UUT port k when UUT input port j is stimulated with waveform x.

Obviously, the above example is heavily paraphrased. However, the reduction in complexity by moving from “how to test” into a “what to test” orientation is valid. Recall that actual execution of tests on this conceptual ATML-oriented system will require transformation of such test description statements into actual executable instructions. This transformation process will use UUT description documents along with the instrument description and any necessary test adapter documents. The process will analyze the “what to test” elements in the test description document along with the input parameters specified in the UUT description document and determine which instrument(s) are necessary to generate these inputs. Finally, the process will create the necessary execution instructions for the selected test execution system. Unless there is a change to the operational characteristics of either the UUT or the instruments in the station, this process likely would be performed only once for each combination of UUT and test description.

I.4.5 Test execution

During test execution, the execution instructions generated will be used to establish the proper signal paths from the instrument(s) to the UUT. Since many instruments will be capable only of returning raw measured values, it will be necessary to convert these raw data values into engineering units. Taking the conversion one step further, the measurements will be formatted to the SIMICA TestResults XML schema, which provides details such as test identity, input parameters, test limits, test outcome, and, optionally, indicted components should the test be capable of identifying them. These SIMICA TestResults documents may be passed simultaneously to the data storage system for archival and to a diagnostic system for determination of recommended actions for the system operator.

Annex J

(informative)

UML models

J.1 Generic ATS testing of a UUT

This generic ATS UML model is provided in support of the material contained in 7.4.

Figure J.1 represents the relationship between various components involved with a generic ATS and a UUT, where the UUT is being tested with the ATS.

This model is not exhaustive; however, it provides sufficient detail to establish the definition of the nine ATML family component standards.

NOTE—Not every component shown in Figure J.1 is an ATML family component standard. For example, the **TestProgram** and **InstrumentSet** components are not ATML family component standards. The six ATML family component standards (see 6.1) were chosen as the interfaces that offer the greatest potential of reducing costs associated with rehosting or replacing ATS components.

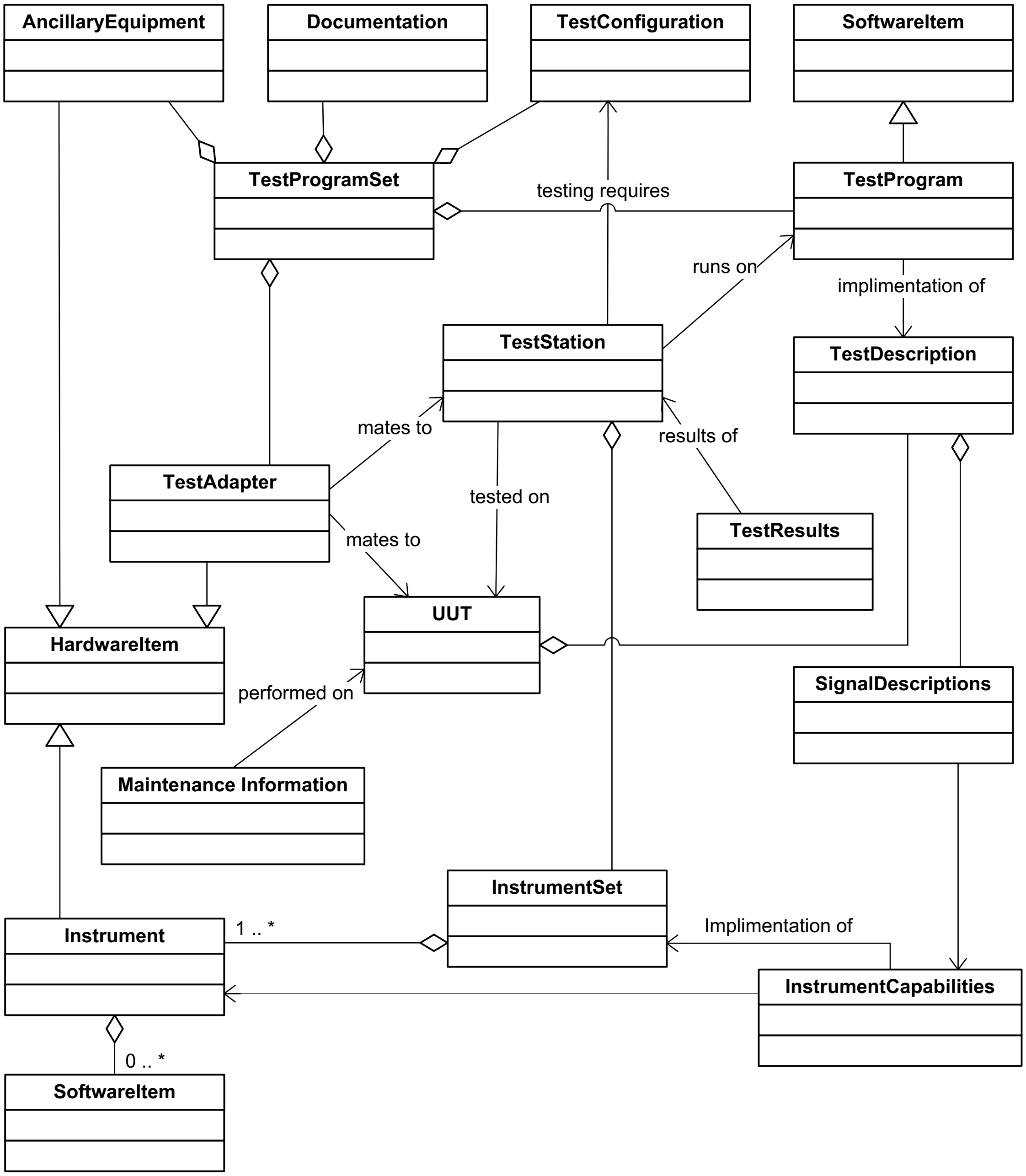


Figure J.1—Generic ATS and UUT various components

J.2 ATML XML schema relationships

This ATML XML schema relationship UML model is provided in support of the material contained in 7.4.

Figure J.2, Figure J.3, Figure J.4, and Figure J.5 collectively represent the relationship between the various ATML family XML schemas. Note that the XML schemas documented in Annex B (represented by the shading) are used by nearly all other ATML family schemas. In these figures, a line labeled **<<uses>>** indicates that the schema at the tail of the arrow uses or includes the schema at the head of the arrow. Similarly, the <<**refines**>> label indicates that the schema at the tail of an arrow is a more specific instantiation of the schema at the head of the arrow, and the <<**collects instances of**>> label indicates that the schema at the tail of an arrow is a collector of XML instance documents, which are valid against the schema at the head of the arrow.

Figure J.2 represents the relationship between the three common element schemas.

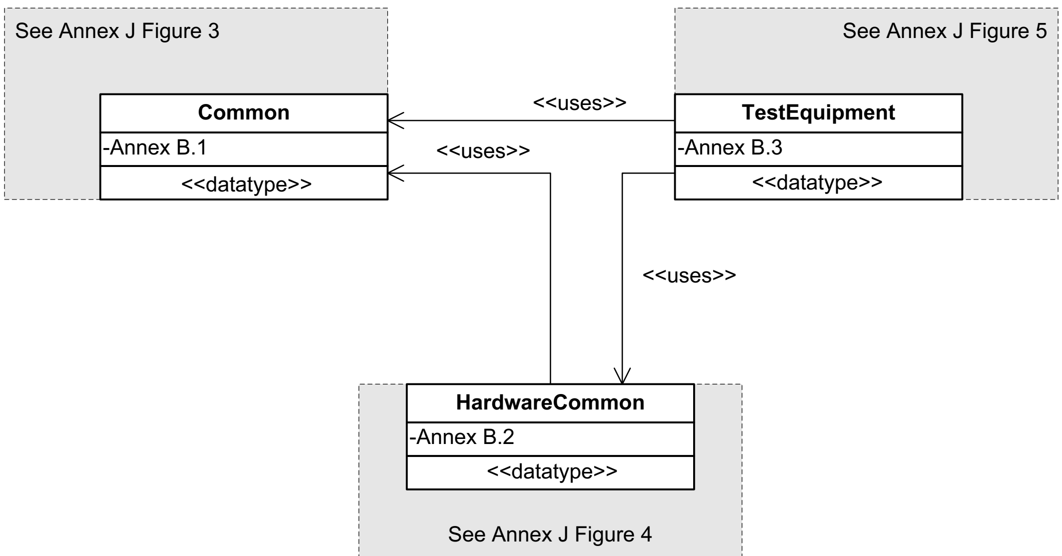


Figure J.2—XML common element schema relationships

Figure J.3 represents the ATML external interface, which includes only Common.xsd.

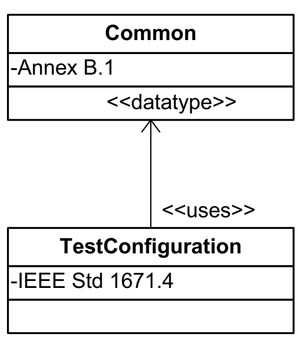


Figure J.3—XML schema relationships to Common

Figure J.4 represents the ATML external interface, which includes HardwareCommon.xsd.

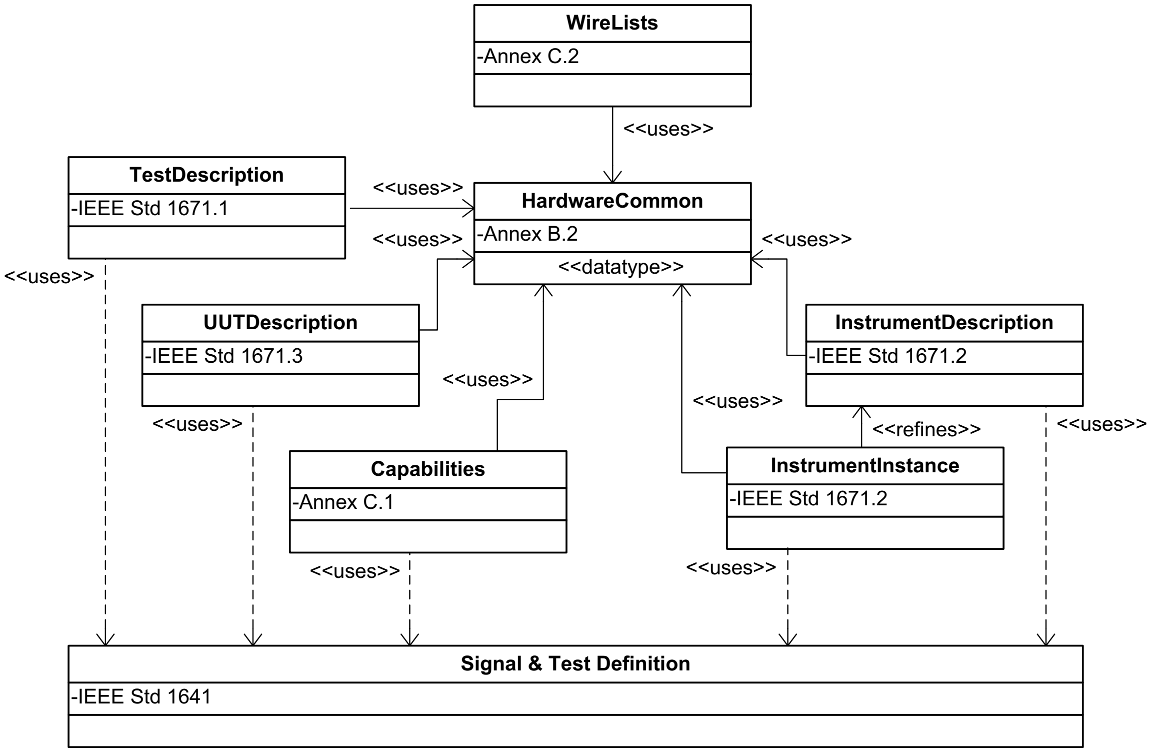


Figure J.4—XML schema relationships to HardwareCommon

Figure J.5 represents the ATML external interface, which includes TestEquipment.xsd.

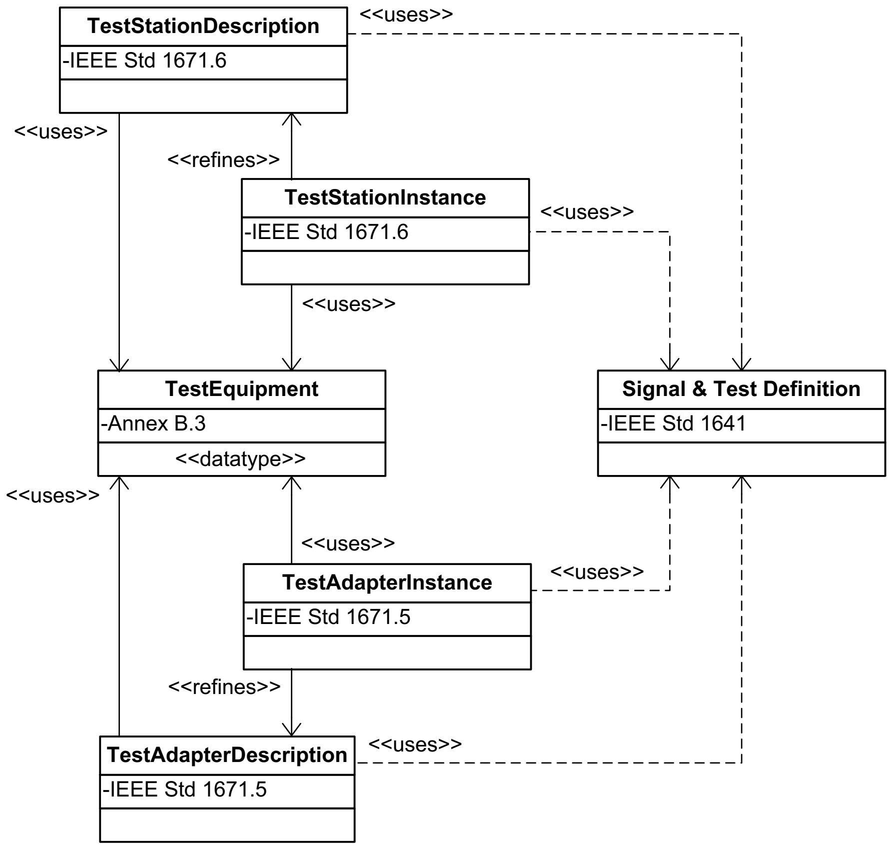


Figure J.5—XML schema relationships to TestEquipment

Annex K

(informative)

Glossary

Abbreviated Test Language for All Systems (ATLAS): A standard abbreviated English language used in the preparation (and documentation) of test procedures or test programs (TPs). The test procedures or TPs are implemented either manually or with automatic or semiautomatic test equipment.

adapter: A device or series of devices designed to provide a compatible connection between the test subject and the test equipment. *Synonyms*: interface device; interface test adapter (ITA); test adapter.

attribute: A documenting characteristic of an entity.

behavior: A formal representation of the characteristics that describe the operation, function, relationships, control, or static properties of a test entity.

class: A template for the creation of an object instance. The class defines the properties of an object.

classification: A grouping of objects on the basis of common characteristics.

context: Reflects the intended scope of a set of tests. Examples of context include manufacturing process test, maintenance test, design verification test, screening test, etc.

diagnosis: The conclusion(s) resulting from tasks, tests, observations, or other information.

diagnostic knowledge: Provides the information required to support the diagnostic process. This knowledge defines the relationships between possible test outcomes and anomalies that may cause these outcomes.

diagnostic process: A structured combination of tasks, tests, observations, and other information used to localize a fault or faults.

diagnostic procedures: *See*: diagnostic process.

framework: A collection of classes created specifically to serve the needs of an application area.

historical data: All relevant information available concerning the product, tests, and test equipment. This includes test observations (raw measurement data) derived test outcomes (i.e., LO, HI, GO), diagnostic conclusions derived from performing tests and the knowledge base, test subject mission and configuration history, test resources mission and history, etc.

instrument: A device whose purpose is usually the generation or measurement of a class of signal.

interface: A shared boundary that specifies the interconnection between two units or systems, hardware or software. In hardware, the specification includes the type, quantity, and function of the interconnection circuits and the type and form of signals to be interchanged via those circuits. In software, the specification includes the object type and, where necessary, the name or instance handle of specific objects copied or shared between the two systems.

interface test adapter (ITA): *See*: adapter.

maintenance: Activity intended to keep equipment (hardware) or programs (software) in satisfactory working condition, including replacements, adjustments, repairs, software/firmware updates, and program improvements. Maintenance can be preventative or corrective.

manual testing: Testing that requires a human to execute some or all of a test procedure.

mapping: Process of correspondence between the elements of one set and the elements of another set.

method: A property of a class that defines a specific behavior.

observation: The raw data acquired by executing a test procedure. It represents the observed characteristics of a specific signal (e.g., the voltage peak of a sinusoid wave form), the observed characteristics of the environment (e.g., the ambient temperature), or the derived value of product characteristics (e.g., the measured value of gain).

operation: An action defined by a procedure.

process: Sequence of operations performed in and by the equipment in which the variable is to be controlled.

product characteristic: An observable attribute of a product. This includes functional, physical, and performance characteristics (e.g., gain and bandwidth of an amplifier).

service: Operation or run-time call whose behavior and interface are standardized. *See also*: method.

software product: A complete set of computer programs, procedures, associated documentation and data designated for delivery to a user.

task: The smallest unit of work subject to management accountability. (e.g., a sequence of instructions treated as a basic unit of work by an operating system)

test: **(A)** An observed activity that may be caused to occur (e.g., stimulus-response) in order to obtain information about the behavior of a test subject. **(B)** A set of stimuli, either applied or known, combined with a set of observed responses and criteria for comparing these responses to a known standard.

NOTE—Definition (B) adapted from IEEE Std 1232 [B20].

test adapter: *See*: adapter.

test asset: An assemblage of instruments, interconnect devices, supporting software, and manual procedures that enable one or more test objectives to be achieved. *See also*: automatic test system.

test control: The functionality that directs and facilitates the execution of tests and the collection of data.

test method: A specification that defines the algorithm, procedures, and required controllable inputs and potential behavior (nominal and anomalous) of a test subject.

test object: Any object defined for use within the domain of test representing an encapsulated view of a test method with interfaces to a test system.

test outcome: A mapping from an observation to one of a set of discrete possibilities.

test procedure: The implementation of a test method.

test program (TP): A program specifically intended for the testing of a test subject.

test requirement: A specification of the test methods and test conditions needed to evaluate and diagnose a test subject.

test strategy: (A) The arrangement of specific tester types to achieve optimum throughput and diagnostic capability at the least possible cost given the fault spectrum, process yield, production rate, and product mix for a particular environment. **(B)** A selection of test methods to achieve some diagnostic test within execution time and test resource constraints.

NOTE—Definition (A) adapted from MIL-STD-1309D [B52].

test subject: The specific product design that is the focus of attention or target for the development of tests and diagnostics.

Annex L

(informative)

Bibliography

The following documents may apply to the application of this standard. For all undated entries, the latest edition of the referenced document (including any amendments or corrigenda) applies. For all dated entries, the referenced document (including any amendments or corrigenda) applies. For all IEEE project entries, the latest draft of the referenced document applies.

[B1] ANSI TIA/EIA-232-F, *Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange*.[18]

[B2] ANSI TIA/EIA-682, *Electronic Design Interchange Format*.

[B3] *ATML Overview*, ATML Focus Group.[19]

[B4] Conventional PCI 3.0.[20]

[B5] *DoD ATS Handbook, 2004*.[21]

[B6] *DoD ATS Framework related documents*.[22]

[B7] *DoD ATS Selection Process Guide, 2009*.[23]

[B8] *DoD Automatic Test Systems Executive Directorate, DoD Automatic Test Systems Master Plan*.[24]

[B9] Gorringe, C., Lopes, T., and Pleasant, D., "ATML Capabilities Explained," *Proceedings of the IEEE AUTOTESTCON 2007*, pp. 178–189, Sept. 2007.[25]

[B10] HTML 4.01 Specification. W3C Recommendation, 24 Dec. 1999.[26]

[B11] IEEE Std 260.1™, IEEE Standard Letter Symbols for Units of Measurement (SI Units, Customary Inch-Pound Units, and Certain Other Units). [27, 28]

[B12] IEEE Std 716™-1995, IEEE Standard Common/Abbreviated Test Language for All Systems.

[B13] IEEE Std 754™-1985, IEEE Standard for Binary Floating-Point Arithmetic.

[18] ANSI publications are available from the Sales Department, American National Standards Institute, 25 West 43rd Street, 4th floor, New York, NY 10036, USA (http://www.ansi.org/).
[19] This document is available from http://grouper.ieee.org/groups/scc20/tii/ATML/Working Groups/Management.
[20] This document is available from http://www.pcisig.com/specifications/conventional/pci_30/.
[21] This document is available from http://www.acq.osd.mil/ats/.
[22] This document is available from http://www.acq.osd.mil/ats/.
[23] This document is available from http://www.acq.osd.mil/ats/.
[24] This document is available from http://www.acq.osd.mil/ats/.
[25] This document is available from the IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854, USA.
[26] This document is available from http://www.w3.org/TR/html4/.
[27] IEEE publications are available from the Institute of Electrical and Electronics Engineers Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).
[28] The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

[B14] IEEE Std 802.3™-2005, IEEE Standard for information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.

[B15] IEEE Std 1003.1™, IEEE Standard Portable Operating System Interface (POSIX).

[B16] IEEE Std 1014™, IEEE Standard for a Versatile Backplane Bus: VMEbus.

[B17] IEEE Std 1057™, IEEE Standard for Digitizing Waveform Recorders.

[B18] IEEE Std 1149.1™, IEEE Standard Test Access Port and Boundry-Scan Architecture.

[B19] IEEE Std 1155™, IEEE Standard for VMEbus Extensions for Instrumentation: VXIbus.

[B20] IEEE Std 1232™, IEEE Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE).

[B21] IEEE Std 1241™, IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters.

[B22] IEEE Std 1394™, IEEE Standard for a High Performance Serial Bus.

[B23] IEEE Std 1445™, IEEE Standard for Digital Test Interchange Format (DTIF).

[B24] IEEE Std 1505™, IEEE Standard for Receiver Fixture Interface.

[B25] IEEE Std 1505.1™, IEEE Trial-Use Standard for the Common Test Interface Pin Map Configuration for High-Density, Single-Tier Electronics Test Requirements Utilizing IEEE Std 1505™.

[B26] IEEE Std 1636™, IEEE Trial-Use Standard for Software Interface for Maintenance Information Collection and Analysis (SIMICA).

[B27] IEEE Std 1636.1™-2007, IEEE Trial-Use Standard for Software Interface for Maintenance Information Collection and Analysis (SIMICA): Test Results and Session Information via the Extensible Markup Language (XML).

[B28] IEEE Std 1636.2™-2010, IEEE Trial-Use Standard for Software Interface for Maintenance Information Collection and Analysis (SIMICA): Exchanging Maintenance Action Information via the Extensible Markup Language (XML).

[B29] IEEE Std 1641™, IEEE Standard for Signal and Test Definition.

[B30] IEEE Std 1641.1™, IEEE Guide for the Use of IEEE Std 1641, Standard for Signal and Test Definition.

[B31] IEEE Std 1671.1™, IEEE Trial-Use Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML: Exchanging Test Descriptions.

[B32] IEEE Std 1671.2™, IEEE Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Information via XML, Exchanging Instrument Descriptions.

[B33] IEEE Std 1671.3™, IEEE Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Information via XML (eXtensible Markup Language): Exchanging UUT (Unit Under Test) Description Information.

[B34] IEEE Std 1671.4™, IEEE Trial-Use Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML, Exchanging Test Configuration Information.

[B35] IEEE Std 1671.5™, IEEE Trial-Use Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Information via XML, Exchanging Test Adapter Information.

[B36] IEEE Std 1671.6™, IEEE Trial-Use Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Information via XML, Exchanging Test Station Information.

[B37] ISO 8601:2004, Data elements and interchangeability formats — Information interchange — Representation of dates and times. [29]

[B38] ISO 60488.2:2004 (IEEE 488.2™), Standard digital interface for programmable instrumentation — Part 2: Codes, Formats, Protocols, and Common Commands.

[B39] ISO 8879:1986, Standard Generalized Markup Language (SGML).

[B40] ISO/IEC 9899:1999, Standard C Programming Language.

[B41] IVI Specifications.[30]

[B42] LXI Standard, LXI Consortium.[31]

[B43] *Namespaces in XML 1.0*. W3C Recommendation, 16 Aug. 2006.[32]

[B44] PCIe Base Specification 1.1.[33]

[B45] PXI-1, PXI Hardware Specification, Rev 2.2.[34]

[B46] PXI-2, PXI Software Specification, Rev 2.1.

[B47] PXI-5, PXI Express Hardware Specification.

[B48] PXI-6, PXI Express Software Specification.

[B49] Ralph, J., "A Proposed Comprehensive Architecture Utilizing the Automatic Test Markup Language," *Proceedings of the IEEE AUTOTESTCON 2007*, pp. 190–196, Sept. 2007.[35]

[B50] Universal Serial Bus, Revision 2.0 Specification.[36]

[B51] *URN Syntax*. Network Working Group, May 1997.[37]

[B52] U.S. Navy, *Definitions of Terms for Test, Measurement and Diagnostic Equipment*, MIL-STD-1309D, Naval Electronics Systems Command (ELEX-8111), Washington, D.C., 12 Feb. 1992.

[29] ISO publications are available from the ISO Central Secretariat, Case Postalė 56, 1 rue de Varembė, CH-1211, Genevė 20, Switzerland/.Suissė (http://www.iso.ch/). ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 25 West 43rd Street, 4th floor, New York, NY 10036, USA (http://www.ansi.org/).
[30] This document is available from http://www.ivifoundation.org/downloads/specifications.htm.
[31] This document is available from http://www.lxistandard.org/home/.
[32] This document is available from http://www.w3.org/TR/2006/REC-xml-names-20060816.
[33] This document is available from http://www.pcisig.com/specifications/ordering_information/.
[34] This document is available from http://www.pxisa.org/Specifications.html.
[35] This document is available from the IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854, USA.
[36] This document is available from http://www.usb.org/developers/docs/.
[37] This document is available from http://www.ietf.org/rfc/rfc2141.txt.

[B53] VXI-1, VXI System Specification, Revision 3.0, dated 24 Nov. 2003.[38]

[B54] VXI*plug&play* Specifications[39]

[B55] *Webster's New Collegiate Dictionary*. Springfield, MA: Merriam-Webster, Inc.

[B56] W3C Technical Reports and Publications.[40]

[B57] XHTML 1.1 *Module-based XHTML*. W3C Recommendation, 31 May 2001.[41]

[B58] *XML Schema Definition Language (XSD) 1.1 Part 1: Structures*, W3C Recommendation, 28 Oct. 2004.[42]

[B59] *XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*, W3C Recommendation, 28 Oct. 2004.[43]

[B60] *XML Schema Tutorial*.[44]

[38] This document is available from http://www.vxibus.org/.
[39] This document is available from http://www.ivifoundation.org/downloads/specifications.htm.
[40] This document is available from http://www.w3.org/TR/.
[41] This document is available from http://www.w3.org/TR/XHTML11.
[42] This document is available from http://www.w3.org/TR/2004/REC-xmlschema-1-20041028.
[43] This document is available from http://www.w3.org/TR/2004/REC-xmlschema-2-20041028.
[44] This document is available from http://www.xfront.com.

Annex M

(informative)

IEEE List of Participants

IEEE List of Participants

The following Test Information Integration (TII) Subcommittee members contributed to the development and preparation of this standard:

Chris Gorringe, *Co-Chair*
Teresa Lopes, *Co-Chair*

Anthony Alwardt
Malcolm Brown
Matt Conrish
Timothy W. Davis
Robert Fox
Jose Gonzalez
Michelle Harris
Ashley Hulme
Anand Jain
Ion Neag
Dan Pleasant
William Ross
Mike Seavey
Dave Sharone
Ronald Taylor

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

Stephen Allen
Anthony Alwardt
H. Stephen Berger
Malcom Brown
Keith Chow
David Droste
James Dumser
Sourav Dutta
Anthony Geneva
Chris Gorringe
Randall Groves
Werner Hoelzl
Ashley Hulme
Anand Jain
Adam Ley
Teresa Lopes
Robert Mcgarvey
Mukund Modi
Ion Neag
Leslie Orlidge
Gordon Robinson
David Rohacek
Bartien Sayogo
Mike Seavey
John Sheppard
Gil Shultz
Joseph Stanco
Walter Struppler
Ronald Taylor
Timothy Wilmering
Oren Yuen

When the IEEE-SA Standards Board approved this standard on 30 September 2010, it had the following membership:

Robert M. Grow, *Chair*
Richard H. Hulett, *Vice Chair*
Steve M. Mills, *Past Chair*
Judith Gorman, *Secretary*

Karen Bartleson
Victor Berman
Ted Burse
Clint Chaplin
Andy Drozd
Alexander Gelman
Jim Hughes
Young Kyun Kim
Joseph L. Koepfinger*
John Kulick
David J. Law
Hung Ling
Oleg Logvinov
Ted Olsen
Ronald C. Petersen
Thomas Prevost
Jon Walter Rosdahl
Sam Sciacca
Mike Seavey
Curtis Siller
Don Wright

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish Aggarwal, *NRC Representative*
Richard DeBlasio, *DOE Representative*
Michael Janezic, *NIST Representative*

Lisa Perry
IEEE Standards Program Manager, Document Development

Soo H. Kim
IEEE Standards Program Manager, Technical Program Development

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch