



ISO/IEC JTC1 SC36

N0175

ISO/IEC JTC1 SC36
Information Technology for Learning, Education, and Training

Title:

IEEE P1484.2.1/D8, PAPI Learner – Core Features

Source:

IEEE LTSC

Project:

All

Document Type:

Working Document

Status:

Also known as SC36/WG3/N0015. Contribution from IEEE LTSC for SC36/WG3. See SC36/N0174 for more information.

Date:

2002-02-02

Action ID:

FYI

Distribution:

P, O, & L Members, WG Conveners

This page is intentionally blank.

IEEE P1484.2.1/D8, 2001-11-25 Draft Standard for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Core Features

Sponsored by the Learning Technology Standards Committee
of the IEEE Computer Society

Copyright © 2001 by the Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue
New York, NY 10016-5997, USA
All rights reserved.

This document is an unapproved draft of a proposed IEEE-SA Standard -- USE AT YOUR OWN RISK. As such, this document is subject to change. Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only. Prior to submitting this document to another standard development organization for standardization activities, permission must first be obtained from the Manager, Standards Licensing and Contracts, IEEE Standards Activities Department. Other entities seeking permission to reproduce portions of this document must obtain the appropriate license from the Manager, Standards Licensing and Contracts, IEEE Standards Activities Department. The IEEE is the sole entity that may authorize the use of IEEE owned trademarks, certification marks, or other designations that may indicate compliance with the materials contained herein.

IEEE Standards Activities Department
Standards Licensing and Contracts
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA

[Note: Information about IEEE LTSC P1484.2.1 can be found at:

<http://ieeeltsc.org/wg2>

This document is also available at:

<http://edutool.com/papi>

This note will be removed upon reaching the final draft of this IEEE document.]

Introduction

(This introduction is not part of IEEE P1484.2.1, Public and Private Information (PAPI) for Learners — Core Features.)

** TO BE SUPPLIED **

At the time this Standard was completed, the working group had the following membership:

Mike Collett, *Chair*

Frank Linton and Brad Goodman, *Co-Chairs (1996-1999)*

Frank Farance, *Technical Editor*

aaa	Josh Tonkel
Carlos. C Amaro	Brendon Towle
Thor Anderson	John Tyler
Debbie Brown	Tom Wason
Peter Brusilovsky	Eamonn Webster
Richard Burke	Ian Wright
Brant Cheikes	Bill Young
Philip Dodds	zzz
Mike Fore	To be supplied
Paul Foster	
Vladimir Goodkodsky	
Martha Gray	
Wayne Hodgins	
Roger Lange	
Ian Kegel	
Cindy Mazow	
William A. McDonald	
Bill Melton	
Yves Nicol	
Claude Ostyn	
Bruce Peoples	
Tom Probert	
Dan Rehak	
Kevin Riley	
Steve Ritter	
Robby Robson	
Randy Saunders	
Jim Schoening	
Paul Siegel	
Kathy Sinitza	
Gayle Stroup	

The following persons were on the balloting committee: (To be provided by IEEE editor at time of publication.)

CONTENTS

1	Overview	9
1.1	Scope	10
1.2	Purpose.....	10
1.3	Out-of-Scope Features.....	11
1.4	Document organization (road map).....	11
2	Normative references	12
3	Definitions	13
3.1	Definitions incorporated via normative reference	13
3.2	accessor [+].....	13
3.3	aggregate (datatype, value) [*].....	13
3.4	API application (conformance paradigm) [+].....	13
3.5	API environment (conformance paradigm) [+]	13
3.6	application programming interface [*].....	13
3.7	application area.....	13
3.8	authentication [ITV] [+].....	13
3.9	authentication exchange [ITV] [+]	14
3.10	authentication information [ITV] [+]	14
3.11	binding [*]	14
3.12	characterstring [+]	14
3.13	child element (XML) [+]	14
3.14	client service (conformance paradigm) [+].....	14
3.15	coding [*].....	14
3.16	conditional data element [*]	14
3.17	confidentiality.....	15
3.18	consume (data) [+]	15
3.19	credentials [ITV]	16
3.20	data application (conformance paradigm) [+]	16
3.21	data element registry [ITV] [+]	17
3.22	data instance [*].....	17
3.23	data object [*].....	17
3.24	data reader [+]	17
3.25	data repository [+]	17
3.26	data set [*]	17
3.27	data structure [*].....	17
3.28	data writer [+].....	18
3.29	digital signature [ITV] [+]	18
3.30	distance (access, system) [+]	18
3.31	distributed (access, system) [+]	18
3.32	encoding [+]	18
3.33	extended data element [*].....	18
3.34	functional unit [+].....	18
3.35	generate (data) [+]	18
3.36	group.....	19
3.37	human identifier [+]	19
3.38	human information (learning technology) [*]	19
3.39	implementation behavior [*]	19
3.40	implementation-defined behavior/value [*].....	19

3.41	implementation value [+]	19
3.42	inbound security threat [+]	19
3.43	information type (PAPI Learner)	19
3.44	integrity (data) [ITV] [+]	20
3.45	internationalized string [+]	20
3.46	internationalized value [+]	20
3.47	interpret (data) [+]	20
3.48	learner [*]	21
3.49	learner entity [*]	21
3.50	learner identifier [+]	21
3.51	learner information [+]	21
3.52	learner performance granularity	22
3.53	learner profile	22
3.54	learning management system [*]	22
3.55	locale [+]	22
3.56	locale-specific behavior [+]	22
3.57	localized string [+]	23
3.58	localized value [+]	23
3.59	longevity (data element) [+]	23
3.60	mandatory data element [*]	24
3.61	multicultural string [+]	24
3.62	multicultural value [+]	24
3.63	nomadic (access, system)	25
3.64	obligation (data element)	25
3.65	obsolete data element [*]	26
3.66	optional data element [*]	26
3.67	outbound security threat	26
3.68	out-of-scope [*]	26
3.69	PAPI Learner	26
3.70	PAPI Learner data application	26
3.71	PAPI Learner extensions	26
3.72	PAPI Learner information	27
3.73	PAPI Learner information type	27
3.74	PAPI Learner record	27
3.75	PAPI Learner record reference	27
3.76	PAPI Learner system	27
3.77	PAPI server	27
3.78	parent element (XML) [+]	27
3.79	peer service (conformance paradigm) [+]	27
3.80	privacy (data) [ITV] [+]	27
3.81	produce (data)	28
3.82	protocol [+]	28
3.83	public and private information	28
3.84	repository [*]	28
3.85	reserved data element [*]	28
3.86	risk acceptance [ITV] [+]	28
3.87	risk analysis [ITV] [+]	28
3.88	risk assessment [ITV] [+]	28

3.89	role-based access control	28
3.90	root element (XML) [+]......	29
3.91	security administrator.....	29
3.92	security information (PAPI Learner).....	29
3.93	security perimeter [+]......	29
3.94	security perimeter integrity [+]......	29
3.95	security policy [ITV] [+]......	29
3.96	security strength [+].	30
3.97	semi-structured data	30
3.98	server service (conformance paradigm) [+]......	30
3.99	simple human identifier [*]	30
3.100	smallest permitted maximum [*]......	30
3.101	transformation phase (data)	30
3.102	undefined behavior/value [*]......	31
3.103	unspecified behavior/value [*]	31
3.104	user [+]......	31
3.105	wildcard pattern [+]......	31
3.106	Acronyms and abbreviations	31
4	Conformance	33
4.1	Conformance level	33
4.1.1	Subsets.....	33
4.1.2	Strictly conforming implementations	33
4.1.3	Conforming implementations	34
4.2	Conformance labels.....	34
4.3	Coding conformance	34
4.3.1	Data set conformance	34
4.3.2	Data instance conformance	35
4.4	API conformance.....	35
4.5	Protocol conformance	36
4.6	Data application conformance	36
4.6.1	Data repository.....	37
4.6.2	Data reader	38
4.6.3	Data writer.....	38
4.7	Registry conformance	38
5	Functionality.....	38
6	Conceptual model	39
6.1	Learner information types.....	39
6.2	Public and private information.....	41
6.3	Information types vs. data repositories.....	41
6.4	Common features, information types, and bindings.....	41
6.5	Metadata model	42
6.6	Data access model	43
6.7	Extending PAPI within an application area	44
6.8	Limiting PAPI extensions	45
6.9	Distance, distributed, and nomadic systems.....	45
6.10	Correlation of granularity levels.....	45
6.11	Security model.....	45
7	Semantics	47

7.1	General data operations	48
7.2	Application-specific data operations	48
7.3	Data compatibility	49
7.4	Foundational datatypes.....	49
7.4.1	ssd_record(semi-structured data)	50
7.4.2	papi_learner_context_label_type	51
7.4.3	arraylist.....	51
7.4.4	locstring_type.....	51
7.4.5	mcstring_array	52
7.4.6	locvalue_type	52
7.4.7	mcvalue_array.....	53
7.4.8	papi_learner_nvp_bucket_type	53
7.4.9	papi_learner_identifier_type	54
7.4.10	papi_learner_hid_type	55
7.4.11	papi_learner_identifier_type_type	55
7.4.12	papi_learner_data_certification_type.....	56
8	Bindings and encodings	57
9	Annex A: Bibliography (informative).....	58
10	Annex B: Conformance labels (normative)	59
10.1	Syntax for human-readable conformance labels	59
10.2	Identification of subsets	59
10.3	Identification of bindings.....	59
10.4	Implementation varieties.....	60
10.5	Coding for automated interpretation	62
11	Annex C: ISO/IEC 11404 data model summary (informative).....	63
12	Annex D: XML coding binding (normative)	65
12.1	XML-related terminology.....	65
12.2	Generating and producing XML	65
12.3	Consuming and interpreting XML.....	68
12.4	Representation of basic data types	69
12.4.1	Characters and character strings.....	69
12.4.2	Integers	70
12.4.3	Real numbers	70
12.4.4	Date and time values.....	70
12.4.5	Void types	72
12.5	Encoding of character representations	73
12.6	Handling exceptions and extensions	73
12.6.1	Implementation-defined behavior	73
12.6.2	Unspecified behavior.....	73
12.6.3	Undefined behavior	73
13	Annex E: DNVP coding binding (normative).....	74
13.1	Dotted Name-Value Pairs (DNVPs)	74
13.1.1	Basic lexical elements	74
13.1.2	Field name and field value	76
13.1.3	Newline processing.....	77
13.1.4	Syntax summary	77
13.2	Generating and producing dotted name-value pairs.....	77
13.3	Consuming and interpreting dotted name-value pairs.....	79

13.4	Representation of basic data types	80
13.4.1	Characters and character strings.....	80
13.4.2	Integers	80
13.4.3	Real numbers	81
13.4.4	Date and time values.....	81
13.4.5	Void types	81
13.5	Encoding of character representations	81
13.6	Handling exceptions and extensions	81
13.6.1	Implementation-defined behavior.....	81
13.6.2	Unspecified behavior	82
13.6.3	Undefined behavior	82
14	Annex F: ASN.1 coding binding (normative)	83
15	Annex G: MDAS API binding (normative).....	84
16	Annex H: Document development (informative).....	85
16.1	Revision history.....	85
16.2	Release notes for this document.....	85
16.3	Resolved issues	85
16.4	Open issues.....	86
16.5	Comments on this document.....	86

1 Overview

Abstract

The Public and Private Information (PAPI) for Learners (PAPI Learner) Standard is a data interchange specification that describes learner information for communication among cooperating systems ("cooperation" may be achieved by conformance to the PAPI Learner Standard and, possibly, other specifications). The data is exchanged: (1) via external specification, i.e., only PAPI Learner coding bindings are used while some other data communication method is mutually agreed upon by data exchange participants, (2) via control transfer mechanism to facilitate data interchange, e.g., PAPI Learner API bindings, or (3) via data and control transfer mechanisms, e.g., PAPI Learner protocol bindings.

An important feature of the PAPI Learner Standard is the logical division, separate security, and separate administration of several types of learner information (also known, collectively, as "learner records"). These types of learner information are also known as "profile information" and "learner profiles". The PAPI Learner Standard may be integrated with other systems, protocols, formats, and technologies. The PAPI Learner Standard is organized as the following Parts:

- IEEE 1484.2.1, "*Standard for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Core Features*": The provisions that are common to other parts of this Standard, e.g., the main data model and references to other standards.
- IEEE 1484.2.2, "*Guide for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Rationale*": An explanation of important decisions during the development of this Standard.
- IEEE 1484.2.3, "*Guide for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Learner Information Security Issues*": Information and recommendations on important security issues for implementations.
- IEEE 1484.2.4, "*Guide for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Examples and Illustrations*": Information for implementers.
- IEEE 1484.2.5, "*Standard for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Registration Authority Process*": How data elements, value spaces, coding schemes, code sets, etc. are registered.
- IEEE 1484.2.6, "*Standard for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Data Element Registry*": The registry of data elements, value spaces, coding schemes, code sets, etc..
- IEEE 1484.2.21, "*Standard for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Learner Contact Information*", e.g., name, postal address, telephone number, etc..
- IEEE 1484.2.22, "*Standard for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Learner Relations Information*", e.g., classmates, teammates, mentors, etc..
- IEEE 1484.2.23, "*Standard for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Learner Security Information*", e.g., public keys, private keys, credentials, etc..

- IEEE 1484.2.24, "*Standard for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Learner Preference Information*", e.g., as useful and unusable I/O devices, learning styles, physical limitations, etc..
- IEEE 1484.2.25, "*Standard for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Learner Performance Information*", e.g., grades, interim reports, log books, etc..
- IEEE 1484.2.26, "*Standard for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Learner Portfolio Information*", e.g., accomplishments and works, etc..

NOTE — The phrase "this Part" or "this Part of this Standard" self-refers to an individual part of the PAPI Learner Standard. The phrase "this Standard" or "the PAPI Learner Standard" refers to the whole collection of parts.

1.1 Scope

The PAPI Learner Standard is a multi-part standard that specifies the semantics and syntax of learner information. Learner information is information associated learners and used by learning technology systems. Learner information may be created, stored, retrieved, used, etc., by learning technology systems, individuals (e.g., teachers, learners, etc.), and other entities.

The PAPI Learner Standard defines and/or references elements for recording descriptive information about: knowledge acquisition, skills, abilities, personal contact information, learner relationships, security parameters, learner preferences and styles, learner performance, learner-created portfolios, and similar types of information. This Standard permits different views of the learner information (perspectives: learner, teacher, parent, school, employer, etc.) and substantially addresses issues of privacy and security.

This Part only specifies the common core features: conformance wording, functionality, conceptual model, core datatypes, and generic bindings.

1.2 Purpose

The purpose of this Standard is:

- To enable learners (students or knowledge workers) of any age, background, location, means, or school/work situation to create and build a personal learner information repository, based on standards, which they can utilize throughout their education, learning experiences, and work life.
- To promote data portability of learner information.
- To provide a framework for data security, data privacy, and data integrity.
- To enable learning content and learning management systems to provide more personalized and effective learning experiences.

The purpose of this Part is to provide common features that may be included, via normative referencing, in other parts of this Standard.

1.3 Out-of-Scope Features

The following features are *outside* the scope of this Standard:

- **Specific Extensions.** Not all human information about learners is specified by this Standard. Implementations may provide their own data extensions to this Standard, but these systems might change from "strictly conforming implementations" to "conforming implementations". *This Standard supports data extension capabilities that may be used by users, groups, vendors, institutions, industries, and others.*
- **Granularity.** This Standard does not specify the granularity of information. For example, in learning technology applications PAPI Learner records can have granularity ranging from professional certifications (e.g., years of learning), to semester grades (e.g., months of learning), to lesson scores (e.g., days of learning), to minute-by-minute data samples of learner progress (e.g., minutes of learning). *This Standard may be used for a wide range of data recording applications.*
- **Repository Design.** This Standard does not specify the implementation or administration of the records repositories but supports a wide variety of design and implementation possibilities. For example, all types of PAPI Learner information may be implemented as a single, combined repository or may be implemented as separate repositories. *The specific design of the repository (or repositories) is a "quality of implementation" feature and is outside the scope of this Standard. The choice and design of partitioning repositories into one or more administrative "realms" is a feature of the implementation and application, and is outside the scope of this Standard.*
- **Specific Security Technologies.** This Standard supports the incorporation of various security architectures, methods, and techniques that support a wide range of implementations and security policies. *However, no specific security technologies (e.g., 128-bit encryption) are mandated by this Standard.*

1.4 Document organization (road map)

This subclause is informative and not normative.

This Part consists of 8 clauses and 8 annexes.

The following is an overview of each section.

- **Clause 1 [Introduction]:** background information and a high-level summary of the features of this Standard.
- **Clause 2 [Normative References]:** normative wording that is incorporated by referring to other standards and specifications.
- **Clause 3 [Definitions]:** a list of terms and their definitions, and a list of abbreviations.
- **Clause 4 [Conformance]:** the technical requirements for claiming conformance to this Standard.
- **Clause 5 [Functionality]:** description of the purpose and use of PAPI Learner implementations.
- **Clause 6 [Conceptual Model]:** description of a "logical" (in contrast to "actual") implementation; the PAPI Learner information types: contact, relations, security, preference performance, portfolio.

- **Clause 7 [Semantics]:** the meaning of data interchange across implementations and bindings.
- **Clause 8 [Bindings and Encodings]:** the mapping of semantics to one or more codings, APIs, and protocols; and the representation and format of information as data formats, calling conventions, and communication layers.
- **Annex A [Bibliography, informative]:** references to related documentation.
- **Annex B [Conformance labels, normative]:** a list of conformance labels for implementation conformance statements (ICSs).
- **Annex C [ISO/IEC 11404 Data Model Summary, informative]:** a summary of the data model in ISO/IEC 11404 notation.
- **Annex D [XML Coding Binding, conditionally normative]:** the XML coding binding of PAPI Learner data interchange.
- **Annex E [DNVP Coding Binding, conditionally normative]:** the RFC 822-style (E-mail header) coding binding of PAPI Learner data interchange.
- **Annex F [ASN.1 Coding Binding, conditionally normative]:** the coding binding of PAPI Learner to Abstract Syntax Notation 1.
- **Annex G [MDAS API Binding, conditionally normative]:** the API binding of PAPI Learner to the Metadata Access Service.
- **Annex H [Document Development, informative]:** a revision history and list of all outstanding issues with respect to this document.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

- IEEE P1484.2.6/D8, Standard for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Data Element Registry
- IEEE P1484.3/D3, Standard for Learning Technology — Glossary
- IEEE P1484.12.1/D6, Standard for Learning Technology — Learning Object Metadata (LOM) — Data Model
- IEEE P1484.13/D3, Simple Human Identifiers
- IEEE P1484.14.1/D2, Guide for Learning Technology — Data Extension Techniques
- IEEE P1484.14.2/D2, Guide for Learning Technology — Rule-Based XML Bindings
- IEEE P1484.14.3/D2, Guide for Learning Technology — Rule-Based DNVP Bindings
- IEEE P1484.15/D3, Standard for Learning Technology — Data and Control Transfer Protocol
- ISO/IEC 2382, Information Technology — Vocabulary (multiple parts)
- ISO/IEC 11179, Information Technology — Data Management and Interchange — Metadata Registries (MDR)
- ISO/IEC 11404, Information Technology — Programming languages, their environments and system software interfaces — Language-independent datatypes
- ISO/IEC TR 14652, Information Technology — Specification method for cultural conventions

3 Definitions

3.1 Definitions incorporated via normative reference

The following terms and their definitions have been incorporated via the normative references in the following order (lowest precedence to highest precedence):

- ISO/IEC 2382, Information Technology — Vocabulary (multiple parts). Terms marked with an "[ITV]" symbol are adapted from this vocabulary standard.
- IEEE 1484.3, Glossary. Terms marked with a "[*]" symbol are defined in this glossary.

3.2 accessor [+]

A user, system, agent, or entity that attempts to access an asset within a security perimeter.

3.3 aggregate (datatype, value) [*]

A generated datatype each of whose values is made up of values of the component datatypes, in the sense that operations on all component values are meaningful. *See ISO/IEC 11404:1996.*

3.4 API application (conformance paradigm) [+]

An implementation that uses the services and resources of an API specification.

Example: Implementation P, a software program, conforms as an API application because it *uses* (in contrast to *implements*) the API according to the requirements of that API standard.

3.5 API environment (conformance paradigm) [+]

An implementation that implements the interface, services, resources, etc. of an API specification.

Example: Implementation L, a software library, conforms as an API environment because it *implements* (in contrast to *uses*) the interface, services, resources, etc. of the API according to the requirements of the API specification.

NOTE — Some API environments may be recursive in nature and, therefore, the implementation may be both an API environment and an API application.

3.6 application programming interface [*]

A boundary across which application software uses facilities of programming languages to invoke services.

3.7 application area

An industry or market segment for which a set of related applications are developed.

3.8 authentication [ITV] [+]

The act of verifying the claimed identity of an entity.

3.9 authentication exchange [ITV] [+]

A mechanism intended to ensure the identity of an entity by means of an information exchange.

3.10 authentication information [ITV] [+]

Information used to establish the validity of a claimed identity of an entity.

3.11 binding [*]

The result of applying or mapping one framework or specification to another.

3.12 characterstring [+]

A finite sequence of characters from the repertoire of a particular character set.

Examples: An ASCII characterstring; an ISO/IEC 10646-1 characterstring.

3.13 child element (XML) [+]

A non-root element **C** in an XML document for which there is one other element **P** in the document such that **C** is in the content of **P**, but **C** is not in the content of any other element that is in the content of **P**.

NOTE — **C** is referred to as a child of **P**.

3.14 client service (conformance paradigm) [+]

An implementation that implements the client portion of a client-server protocol specification.

Example: Implementation C, a software program, conforms as a client service because it implements the client portion of the protocol according to the requirements of that protocol's specification.

NOTE — Typically, client services make requests to server services and, possibly, peer services that act as servers.

3.15 coding [*]

(1) In information interchange, a formalized or structured representation of information. *See also:* **encoding**. (2) A process of representing information in some structure.

3.16 conditional data element [*]

A data element of a data structure that is defined and is included within an instance of the data structure only under certain conditions. *See also:* **data element obligation**; **extended data element**; **mandatory data element**; **optional data element**.

NOTE — The "conditional" nature of a data element is an obligation attribute.

3.17 confidentiality

The property of data that indicates the extent to which these data have not been made available or disclosed to unauthorized individuals, processes, or other entities.

NOTE 1 — Confidentiality is a security technique that minimizes outbound security threats to an acceptable level by permitting retrieval or read access to authorized entities, and prohibiting retrieval and read access to unauthorized entities.

NOTE 2 — Various security technologies may implement "confidentiality".

NOTE 3 — *See Also:* IEEE 1484.2.3, PAPI Learner Information Security Notes, for more details on confidentiality-related issues.

3.18 consume (data) [+]

To read data and then process it to the extent that some lexical or coding boundaries are discovered. A data consumer performs a limited number of translation phases. *Other Forms:* **consume data**, **data consumer**, **data consumption**. *See Also:* **interpret** (data); **produce** (data).

NOTE — Data is consumed before it is interpreted.

Example 1: In the following character stream:

```
<R>
  <A>123.45</A>
  <B>PQR</B>
  <C X="Y">Z</C>
</R>
<R>
  <D>JKL</D>
  <E>
    <F>XXX</F>
    <G>YYY</G>
  </E>
</R>
```

a data consumer might recognize: there are two records, both with tags "R"; the first "R" record contains three records with tags "A", "B", "C"; and the second "R" record contains two records with tags "D" and "E". However, the data consumer: might not understand the meanings of tags (what does "..." mean?); might not validate the tags (is "<C>" permitted to have the attribute "X"?); might not validate the contents of the records (within record "A", is "123.45" a valid value?); or might limit the depth of its analysis (e.g., "R" is only explored one level deep to discover tags "D" and "E", but only a limited analysis of "E" is performed such as finding balanced tags within the contents of "E", but not analyzing or discovering tags "F" and "G"). Thus, a data consumer might only have a partial understanding of an information structure.

Example 2: Below is an API example that distinguishes data consumption from data interpretation in a case where extended data of the implementation is indirectly used, yet the implementation is strictly conforming.

```
//// This example shows two files:
```

Copyright © 2001 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

//// 1. "std_data.h": a header that includes an extended element
//// 2. A strictly conforming application that uses the header

////////////////////////////////////
//// The following is the include file "std_data.h"
struct std_data
{
    int std_element_1;    // Mandatory element.
    void *std_element_2; // Optional element.
    int ext_element_3;   // Extended element.
};

////////////////////////////////////
//// The strictly conforming application begins.
// Include the standard header (contents are listed above)

#include "std_data.h"
struct std_data x; // Declares "x" as standard data.
my_code()
{
    struct std_data y,z; // Declares "y" and "z".

    // Strictly conforming code, yet extended
    // element "ext_element_3" is copied.
    memcpy(&y,&x,sizeof x);

    // Assign string to "std_element_2".
    // Assign length to "std_element_1".
    y.std_element_2 = "hello there";
    y.std_element_1 = strlen(y.std_element_2);

    // Still strictly conforming code, yet extended
    // element "ext_element_3" is copied
    memcpy(&z,&y,sizeof y);
}

```

This example is strictly conforming because the implementation only interprets or generates elements from a standard set, i.e., `std_element_1` and `std_element_2`. The `memcpy` (copy object in memory) operations are the equivalent *consume* and *produce* operations in this hypothetical API binding, while the direct element accesses (e.g., `y.std_element_1`) are the *interpret* and *generation* operations for this hypothetical API binding.

3.19 credentials [ITV]

Data that are transferred to establish the claimed identity of an entity.

3.20 data application (conformance paradigm) [+]

An implementation of a functional unit interface of the data interoperability specification.

Examples: data repository, data reader, data writer.

NOTE 1 — In this context, a functional unit might also be called an "information technology system".

NOTE 2 — Given a data interoperability standard named D, a standard D *data application* is different from a standard D *application*. The latter is any information technology *implementation* that incorporates the standard D, while the former is a limited subset of information technology *systems*, e.g., a data repository, a data reader, a data writer.

3.21 data element registry [ITV] [+]

An information resource kept by a registration authority that describes the meaning and representational form of data elements, including registration identifiers, definitions, names, value spaces, meta-data and administrative attributes, etc..

3.22 data instance [*]

A data set rendered in some binding. *See also:* **data set**.

3.23 data object [*]

A unit of data processing within the conceptual model of accessing implementations' data. *See also:* **data element; data instance; data set; data structure**.

3.24 data reader [+]

A data application that operates *as if* it processes data in two phases:

- Phase #1: It consumes data, based on external data sources.
- Phase #2: It interprets data from Phase #1, which results in data sets that are internal to the data application.

NOTE — The "as if" rule implies that, conceptually, the data reader processes the information in two phases (consumption and interpretation). However, the design of implementations is not constrained and implementations may use any number of phases of data processing.

3.25 data repository [+]

(1) A collection of information.

(2) An implementation of a collection of information along with data access and control mechanisms, such as search, indexing, storage, retrieval and security.

3.26 data set [*]

An instance of an aggregate of zero or more data elements. *Syn:* **data structure**.

NOTE — A data set is binding-independent.

3.27 data structure [*]

(1) The datatype of an aggregate of zero or more data elements. (2) *See:* **data set**. Examples: a record; a set; a sequence; a list; an array.

NOTE — A data structure may be a data element of a higher-level data structure

3.28 data writer [+]

A data application that operates *as if* it processes data in two phases:

- Phase #1: It generates data from data sets that are internal to the data application.
- Phase #2: It produces data, based on the results from Phase #1, to external data sinks.

NOTE — The "as if" rule implies that, conceptually, the data writer processes the information in two phases (generation and production). However, the design of implementations is not constrained and implementations may use any number of phases of data processing.

3.29 digital signature [ITV] [+]

Data appended to a message, that allow the recipient of the message to verify the source of the message.

3.30 distance (access, system) [+]

Is constrained by bandwidth limitations or delays in communication systems such that applications are significantly affected.

3.31 distributed (access, system) [+]

Uses the internet or wide area networks as the primary means of communication among subsystems and related systems.

3.32 encoding [+]

The format and representation of data. *See also:* **coding**.

3.33 extended data element [*]

A data element of a data structure that is defined outside a standard and permitted within an instance of the data structure. Extended data elements may be used as allowed by data interchange participants and data interchange implementations. *See also:* **conditional data element; data element obligation; mandatory data element; optional data element**.

3.34 functional unit [+]

An entity of hardware or software, or both, capable of accomplishing a specified purpose.

NOTE — Information technology systems and their subsystems are examples of functional units.

3.35 generate (data) [+]

To transform data from its meaning to some form suitable for data interchange. *See Also:* **interpret (data); produce (data)**.

Example: To serialize a data structure according to a conceptual model without rendering the data in a specific coding or encoding.

3.36 group

A collection of users who share some common attributes.

NOTE 1 — The nature and definition of the common attributes is outside the scope of this Standard.

NOTE 2 — Groups may be created by administrators, users, or others.

NOTE 3 — Depending on administration policy, a user may belong to more than one group.

3.37 human identifier [+]

A designator associated with a human. *See Also:* **learner identifier**.

NOTE — Identifiers are specified by IEEE 1484.13 Simple Human Identifiers.

3.38 human information (learning technology) [*]

Information that is primarily associated with, tracked by, tracked for, and about humans in information technology systems and learning technology systems.

3.39 implementation behavior [*]

Observable actions or appearance of an implementation. *See also:* **implementation-defined behavior; undefined behavior; unspecified behavior**.

3.40 implementation-defined behavior/value [*]

Unspecified behavior or an unspecified value(s) for which each implementation documents how the choice among the available alternatives is made. Example: Permitting a maximum size, as measured in octets, of a coding. *See also:* **implementation behavior; undefined behavior; unspecified behavior**.

3.41 implementation value [+]

A quantifiable artifact associated with an implementation. *See Also:* **implementation behavior; implementation-defined behavior/value; undefined behavior/value; unspecified behavior/value**.

3.42 inbound security threat [+]

An external threat that breaches the security of a system and affects information inside the security perimeter. *See Also:* IEEE 1484.2.3, PAPI Learner Information Security Notes, for related information.

Examples: The data injected into a communications stream; changing information; destroying information.

3.43 information type (PAPI Learner)

A category of information within a particular application area that is associated with some subset of application use and/or administration.

Example: PAPI Learner information (an application area) contains six information types: (1) PAPI Learner contact information, (2) PAPI Learner relations information, (3) PAPI Learner security information, (4) PAPI Learner preference information, (5) PAPI Learner performance information, and (6) PAPI Learner portfolio information.

3.44 integrity (data) [ITV] [+]

The property of data whose accuracy and consistency are preserved regardless of changes made.

NOTE 1 — Data integrity is a technical policy about information security that reduces inbound security threats to an acceptable level.

NOTE 2 — Data integrity may include: controlling the creation of information, controlling changes to information, or other techniques. The policy may be implemented by various security techniques, security technologies, security procedures, practices, etc..

NOTE 3 — *See Also:* IEEE 1484.2.3, PAPI Learner Information Security Notes, for more details on data integrity.

3.45 internationalized string [+]

A localized string specified by an international standard.

NOTE 1 — A string is internationalized, i.e., has the same meaning across cultural, linguistic, or regional conventions, by virtue of the fact that the string specification is an international standard.

Example: The date "1999-07-01" is an internationalized string that represents July 1, 1999. The specification for this string is ISO 8601, Date and Time Formats.

NOTE 2 — There may be more than one internationalized string associated with a multicultural string, e.g., "1999-07-01" and "19990701" are both internationalized strings.

3.46 internationalized value [+]

A localized value specified by an international standard.

NOTE 1 — A value is internationalized, i.e., has the same meaning across cultural, linguistic, or regional conventions, by virtue of the fact that the value specification is an international standard.

Example: The value "004" is an internationalized value that represents Afghanistan. The specification for this string is ISO 3166, Country Codes.

NOTE 2 — There may be more than one internationalized string associated with a multicultural value, e.g., "AF" and "AFG" are both internationalized strings that represent Afghanistan in ISO 3166.

3.47 interpret (data) [+]

To process data to discover its meaning, to the extent required by this Standard. *Other Forms:* **interpret data**, **data interpreter**, **data interpretation**. *See Also:* **generate** (data); **consume** (data).

NOTE — Data is consumed before it is interpreted.

Example 1: In the following character stream:

```

<R>
  <A>123.45</A>
  <B>PQR</B>
  <C X="Y">Z</C>
</R>
<R>
  <D>JKL</D>
  <E>
    <F>XXX</F>
    <G>YYY</G>
  </E>
</R>

```

a data consumer might recognize: there are two records, both with tags "R"; the first "R" record contains three records with tags "A", "B", "C"; the second "R" record contains two records with tags "D" and "E". Because only these tags are recognized, only these tags are candidates for data interpretation. Assuming tag "E" represents an extended data element, a data interpreter might only recognize the standardized tags "A", "B", "C", and "D". Based on (1) the separation of the "consume" and "interpret" phases of translation, and (2) a particular standards binding (XML-like in this case), an application might only interpret the standardized features A, B, C, and D.

An application that combines data consumption and data interpretation, but only interprets standardized data elements, might be strictly conforming data reader.

3.48 learner [*]

An individual engaged in acquiring knowledge or skills with a learning technology system. *See also:* **learner entity**.

3.49 learner entity [*]

An individual learner, or a group of learners considered as a single entity that interacts with a learning technology system. *See also:* **learner**.

3.50 learner identifier [+]

A designator associated with a learner.

NOTE — A learner may have more than one learner identifier — a non-singular identifier. The policy of singularity or non-singularity of identifiers is outside the scope of this Standard.

Examples: ISO/IEC 21484-13 CD1 Simple Human Identifiers, passport numbers, E-mail addresses, social security numbers.

3.51 learner information [+]

The intersection of general learning technology information and human information for learners or learner entities.

3.52 learner performance granularity

The relative size, scope, or detail of a learner performance information.

NOTE — Learner performance records may have different granularities.

Example: recording key clicks (e.g., seconds of learning); minute-by-minute data samples of learner progress (e.g., minutes of learning); lesson scores (e.g., days of learning); semester grades (e.g., months of learning); professional certifications (e.g., years of learning).

3.53 learner profile

(1) Information about a learner used by specific learning technology components, learning technology applications, and learner administration. A subset of learner information, in general.

Example 1: A learner profile includes information such as contact, relations, security, preference, performance, portfolio, and, possibly, other types of information.

Example 2: PAPI Learner information is an example of a learner profile.

(2) A standards profile that contains normative references to standards and specifications of learner information.

Example 3: This Standard is incorporates, via normative references, other learner information standards and specifications, such as the IEEE 1484.2.2x series of standards.

3.54 learning management system [*]

A system that (1) schedules learning resources; (2) assists, controls, and/or guides the learning process; and (3) analyzes and reports learner performance.

3.55 locale [+]

An identifier or descriptor associated with a set of cultural, linguistic, or regional conventions. *See also:* **localized string, localized value, multicultural value, multicultural string.**

Example: The locale

```
en-GB;GMT0BST;dd.mm.yyyy hh:mm
```

might mean the following conventions apply:

- The language is English, specifically British English.
- The timezone in the winter is called GMT and is 0 hours West of UTC.
- The timezone in the summer is called BST.
- The date and time are written in the format "dd.mm.yyyy hh:mm".

3.56 locale-specific behavior [+]

Behavior that depends on local conventions of nationality, culture, language, institution, etc., which is documented by each implementation.

3.57 localized string [+]

A localized value based on characterstring datatype. *See also:* **localized value**, **multicultural string**, **multicultural value**.

NOTE — A locale may identify which set of cultural, linguistic, or regional conventions apply.

Example 1: A specification or implementation of a localized string might consist of the pairing of a characterstring with an explicit locale:

```
// The following assignment has an explicit locale.
locstring sample_phrase = {"phrase", "locale"}
// The following XML excerpt uses an explicit locale:
<tag xml:lang="locale">phrase</tag>
```

Example 2: A specification or implementation of a localized string might only consist of a characterstring if the cultural, linguistic, or regional conventions can be inferred by the usage context, i.e., an implicit locale:

```
// The following phrase is within the larger context
// of British English text ("en-GB").
("initialisation")
```

3.58 localized value [+]

A value, of a particular datatype, that is associated with a particular set of cultural, linguistic, or regional conventions. *See also:* **locale**, **localized string**, **multicultural string**, **multicultural value**.

NOTE — Separately (explicitly) or implicitly, a locale may identify which set of cultural, linguistic, or regional conventions apply.

Example: A specification or implementation of a localized value might include the pairing of a value with an explicit locale:

```
// An example definition of a datatype for some localized value.
record some_localized_value_datatype :
(
    // The description of the locale.
    locale: characterstring,
    // In contrast to a localized string, whose value must be
    // of the datatype "characterstring", the value of a localized
    // value may be any datatype, i.e., whatever "localized_value"
    // may be defined as.
    value: localized_value,
)
```

3.59 longevity (data element) [+]

An attribute of a data element specification that indicates intention for incorporation into past, present, or future editions of a standard. *See Also:* **obligation** (data element); **obsolete data element**; **reserved data element**.

NOTE — Longevity attributes are independent of obligation attributes.

Example 1: An obsolete data element might have been intended for inclusion in past editions of a standard, but is intended to be excluded in future editions of a standard.

Example 2: A reserved data element might not have been included in past editions of a standard, and might be intended for inclusion in future editions of a standard.

3.60 mandatory data element [*]

A data element of a data structure that is defined and is required within an instance of the data structure. *See also:* **conditional data element; data element obligation; extended data element; optional data element.**

NOTE — The "mandatory" nature of a data element is an obligation attribute.

3.61 multicultural string [+]

A set of localized strings that are intended to represent the same meaning. *See also:* **locale, localized string, localized value, multicultural value.**

Example 1: A multicultural string for the concept "July 1, 1999 6PM" (New York City time) might be:

```
mcstring_for_july_1 =
(
  ( "en-GB;GMT0BST;dd.mm.yyyy hh:mm",
    "01.07.1999 23:00" )
  ( "en-US;EST5EDT;mm/dd/yyyy hh:mm aa",
    "07/01/1999 06:00 PM" )
  ("iso_8601;UTC0;yyyy-mm-dd hh:mm tz",
    "1999-07-01 22:00 UTC" )
)
```

This multicultural string `mcstring_for_july_1` contains three elements: the localized string for London ("01.07.1999 23:00"), the localized string for New York City ("07/01/1999 06:00 PM"), and the localized string for ISO 8601 ("1999-07-01 22:00 UTC"). The last of the three elements is also known as an internationalized string.

Example 2: A multicultural string for the concept "information technology" might be:

```
mcstring_for_IT =
(
  ( "fr-CA", "Technologies de l'information" ),
  ( "en-US", "Information technology" ),
)
```

This multicultural string `mcstring_for_IT` contains two elements: the localized string for Canadian French, and the localized string for US English.

3.62 multicultural value [+]

A set of localized values, of a particular datatype, that are intended to represent the same meaning. *See also:* **locale, localized string, localized value, multicultural string.**

NOTE — When the set of localized values is grouped as a separate unit, they are typically represented as an array, set, table, or record aggregate with each element, a localized value, associated to or labeled with an individual locale.

Example 1: A multicultural value the spans several data elements, but is not grouped as a single unit:

```

some_value_in_locale_A =
(
    data_element_P ...,
    data_element_Q ...,
    // The localized value; locale is implicitly "A"
    localized_value_LV ...,
    data_element_T ...,
)

some_value_in_locale_B =
(
    data_element_P ...,
    data_element_Q ...,
    // The localized value; locale is implicitly "B"
    localized_value_LV ...,
    data_element_T ...,
)

some_value_in_locale_C =
(
    data_element_P ...,
    data_element_Q ...,
    // The localized value; locale is implicitly "C"
    localized_value_LV ...,
    data_element_T ...,
)

```

Example 2: See the term "multicultural string" for an examples of localized values grouped as a single unit with explicit locales.

3.63 nomadic (access, system)

(1) The appearance of continuity of service across separate communication sessions and geographic locations. (2) Sometimes-disconnected from the networks used for communication among its sub-systems and related systems.

NOTE — Also known as "sometimes-connectivity" and/or "sometimes-roaming".

3.64 obligation (data element)

The requirements and permissibility of data elements that determine the validity of a data structure. *See Also:* **longevity** (data element); **conditional data element**; **extended data element**; **mandatory data element**; **optional data element**.

NOTE — Obligation attributes are independent of longevity attributes.

Example: A data structure **X**, has four elements: **A** and **B** are mandatory, **C** is optional, and **D** is conditional if **B** has the value **true**. The following are sample valid and invalid data structures:

```

( A=123 )           // invalid: missing mandatory element B
( A=123, B=false ) // valid
( A=123, B=true )  // invalid: missing conditional element D
( A=123, B=true, D=17 ) // valid
( A=123, B=false, D=17 ) // valid: allowable because the example

```

Copyright © 2001 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

// "conditional" wording above only
// makes requirements and makes no
// prohibitions
( A=123, B=nil, C=345 ) // valid

```

3.65 obsolete data element [*]

A data element of a data structure that is defined but should no longer be included within an instance of the data structure. *See also:* **data element longevity**; **reserved data element**.

NOTE 1 — The "obsolete" nature of a data element is a longevity attribute.

NOTE 2 — The use of obsolete data elements is deprecated and their specification may be removed from future revisions of a standard

3.66 optional data element [*]

A data element of a data structure that is defined but is not required within an instance of the data structure. *See also:* **conditional data element**; **data element obligation**; **extended data element**; **mandatory data element**.

NOTE — The "optional" nature of a data element is an obligation attribute

3.67 outbound security threat

The theft or unauthorized duplication of information such that the information becomes available outside the security perimeter or no longer remains inside the security perimeter. *See Also:* IEEE 1484.2.3, PAPI Learner Information Security Notes, for related information.

Examples: Snooping network packets; taking information.

3.68 out-of-scope [*]

Pertaining to features that are not specified in a standard and may be specified elsewhere.

3.69 PAPI Learner

(1) An abbreviated name for the IEEE 1484.2.* PAPI Learner series of standard and guides.

(2) A subset of public and private information (human information) about learners as defined by this Standard.

NOTE — The term "learner profile information" is a generic name and the PAPI Learner Standard is *one* description of "learner profile information".

3.70 PAPI Learner data application

A data repository, data reader, or data writer, as specified in this Standard.

3.71 PAPI Learner extensions

Extended data elements and/or data services extensions.

NOTE — Strictly conforming implementations do not use extensions. Conforming implementations may use extensions.

3.72 PAPI Learner information

See: **PAPI Learner (2)**.

3.73 PAPI Learner information type

An information type for the PAPI Learner Standard.

3.74 PAPI Learner record

A collection of learner information represented as defined by the PAPI Learner Standard.

3.75 PAPI Learner record reference

An identifier that points to a PAPI Learner record.

3.76 PAPI Learner system

See: **PAPI Learner application**.

3.77 PAPI server

See: PAPI Learner server.

3.78 parent element (XML) [+]

For each non-root element **C** in an XML document, a unique element **P** in the document such that **C** is in the content of **P**, but **C** is not in the content of any other element that is in the content of **P**.

NOTE — **P** is referred to as the parent of **C**.

3.79 peer service (conformance paradigm) [+]

An implementation that implements a peer-to-peer protocol specification.

Example: Implementation P, a software program, conforms as a peer service because it implements the peer-to-peer protocol according to the requirements of that protocol's specification.

NOTE — Typically, peers interact with other peers. Peers may interact with clients and servers when they can act in the role of servers and clients, respectively.

3.80 privacy (data) [ITV] [+]

Freedom from intrusion into the private life or affairs of an individual when that intrusion results from undue or illegal gathering and use of data about that individual.

NOTE — Privacy is a technical policy about information security that reduces outbound security threats to an acceptable level. Privacy may include: controlling the copying of information, control-

ling transfer of information, or other techniques. The security policy may be implemented by various security techniques, security technologies, security procedures, practices, etc..

3.81 produce (data)

To process data to the extent that lexical or coding boundaries are defined and then write the resultant data. *Other Forms:* **produce data, data producer, data production.** *See Also:* **generate** (data); **consume** (data).

NOTE — Data is generated before it is produced.

3.82 protocol [+]

A set of rules that determines the behavior of functional units in achieving communication.

3.83 public and private information

Human-related information, of varying degrees of public and private accessibility, exchanged and administered within information technology systems and communication networks.

3.84 repository [*]

(1) A collection of information. (2) An implementation of a collection of information along with data access and control mechanisms, such as search, indexing, storage, retrieval and security.

3.85 reserved data element [*]

A data element of a data structure that is not defined and is not permitted within an instance of the data structure. *See also:* **data element longevity; obsolete data element.**

3.86 risk acceptance [ITV] [+]

A managerial decision to accept a certain degree of risk, usually for technical or cost reasons.

3.87 risk analysis [ITV] [+]

A systematic method of identifying the assets of a data processing system, the threats to those assets, and the vulnerability of the system to those threats.

3.88 risk assessment [ITV] [+]

See: risk analysis.

3.89 role-based access control

A security technique for authentication that authorizes operations or allows access to resources based upon the user's identity and his/her relationship to other users.

Example 1: A teacher has read/write access to the grades for his/her students (role: "the teacher of the student"), but no access to other students' grades.

Example 2: A principal has read-only access to the grades of all of his/her teachers' students (role: "the principal of the teachers of the students"), but the principal is not permitted to change any grades.

3.90 root element (XML) [+]

An element such that no part of it appears in the content of any other element within an XML document.

NOTE — For all other elements, if the start tag is in the content of another element, then the end tag is in the content of the same element. More simply stated, the elements, each delimited by a start tag and end tag, nest properly within each other.

3.91 security administrator

A person who is responsible for security management within a security perimeter.

NOTE — A security administrator may be concerned about several types of security, including information security, physical security, financial security, and personal security.

3.92 security information (PAPI Learner)

This subclause is informative and not normative.

Information that supports a security policy.

NOTE 1 — The term "security information" has a variety of meanings in various contexts.

NOTE 2 — This Standard only describes learner security information, and only describes security information to the extent that a variety of security implementations are *possible*. This Standard makes no requirements for specific security policies, procedures, techniques, or technologies.

3.93 security perimeter [+]

A continuous, closed partition that separates the "inside" from the "outside". The "inside" of the security perimeter is intended to be secure. The "outside" of the security perimeter may not be secure.

3.94 security perimeter integrity [+]

A level of security protection such that inbound security threats and outbound security threats are maintained at an acceptable level of risk. *See Also:* IEEE 1484.2.3, PAPI Learner Information Security Notes, for related information.

Example 1: The loss of security perimeter integrity implies that (1) the level of security protection is less than an acceptable level; (2) the inbound security threats have increased to more than an acceptable level; or (3) the outbound security threats have increased to more than an acceptable level.

Example 2: User permissions and administration are aspects of security perimeter integrity.

3.95 security policy [ITV] [+]

A plan or course of action adopted for providing computer security.

3.96 security strength [+]

The degree of security that characterizes the implementation of a security perimeter.

NOTE — The following features may characterize security strength:

- Quality Level: What level of security is provided? Examples of security levels: minimal security, auditing-capable, provable design.
- Contingencies: What happens when the system is violated? What fallbacks are available? What is the extent of the damage?
- Penalties: What happens to violators of the security mechanism?

Example: Security strength might concern the number of bits used in encryption keys.

3.97 semi-structured data

An aggregate datatype that has the following characteristics:

- Each element of the aggregate datatype is a list or a terminal.
- A list is a *sequence* (an ISO/IEC 11404 term) or an *array* (an ISO/IEC 11404 term) of elements of unspecified datatype; each element itself is a list or a terminal.
- A terminal element has at least one of: a known, specified, or agreed upon label, or a known, specified, or agreed upon datatype.

3.98 server service (conformance paradigm) [+]

An implementation that implements the server portion of a client-server protocol specification.

Example: Implementation S, a software program, conforms as a server service because it implements the server portion of the protocol according to the requirements of that protocol standard.

NOTE — Typically, server services respond to requests from client services and, possibly, peer services acting as clients.

3.99 simple human identifier [*]

A human identifier specified by IEEE 1484.13 Simple Human Identifiers. *See Also:* **human identifier**.

3.100 smallest permitted maximum [*]

For implementation-defined values, the smallest permitted maximum value.

Example: "The smallest permitted maximum string length of element X shall be 17."

3.101 transformation phase (data)

A single phase of a series of phases of data processing whereby data is received (e.g., phase **N-1**), processed (phase **N**), and emitted for further processing (e.g., phase **N+1**).

3.102 undefined behavior/value [*]

Implementation behavior for which a standard imposes no requirements. Possible undefined behaviors include, but are not limited to:

- ignoring the situation completely,
- unpredictable results,
- behaving in a documented manner characteristic of the environment,
- terminating processing.

See also: **implementation behavior; implementation-defined behavior; unspecified behavior.**

3.103 unspecified behavior/value [*]

Implementation behavior for which a standard provides two or more alternatives and imposes no further requirements on what is chosen in any instance. *See also:* **implementation behavior; implementation-defined behavior; undefined behavior.**

3.104 user [+]

A human, his/her agent, or a surrogate that interacts with information technology systems.

NOTE — For the PAPI Learner Standard, a user is typically concerned about the proper creation, correction, or destruction of records, the potential use of records; the security of records, and the usefulness of records within information technology systems. Users may be learners, teachers, employers, administrators, etc..

3.105 wildcard pattern [+]

A technique of identification whereby the elements of a set are described by means of a pattern. A wildcard pattern may be an Extended Backus Naur Form (EBNF) production rule definitions list (i.e., the right-hand side of the production rule) that contains no non-terminals, no defining operators, no comments, and no rule terminators.

NOTE — This Standard uses ISO/IEC 14977 Extended Backus Naur Form (EBNF) for its wildcard pattern syntax.

Example: The following wildcard pattern, as described in EBNF:

```
"ab", { ? any ISO/IEC 10646 character ? }
```

matches any string beginning with "ab", such as "ab", "abc", "abd", "ababab", but not "a" or "ba". The pattern is decomposed into the terminal string "ab", the concatenation character ",", the special sequence "? any ISO/IEC 10646 character ?", and the repeat-zero-or-more-times "{ ... }" grouping.

3.106 Acronyms and abbreviations

- API: Application Programming Interface
- ASN.1: Abstract Syntax Notation One; also known as ISO/IEC 8824 and 8825
- DCTP: Data and Control Transfer Protocol
- ICS: Implementation Conformance Statement
- IETF: Internet Engineering Task Force

- LID: Language Independent Datatypes; also known as ISO/IEC 11404
- LMS: learning management system
- LTSC: Learning Technology Standards Committee
- MDAS API: ISO/IEC 20944-x Metadata Access Service API
- PAPI: Public and Private Information specification
- RFC: Request for Comments; a citation prefix for specifications developed by the Internet Engineering Task Force
- SC/C: strictly conforming/conforming
- SPM: smallest permitted maximum
- W3C: World Wide Web Consortium
- XML: Extensible Markup Language

4 Conformance

In this Standard, "shall" is to be interpreted as a requirement on an implementation; "shall not" is to be interpreted as a prohibition. If a "shall" requirement or "shall not" prohibition is violated, the behavior is undefined. Undefined behavior is otherwise indicated in this Standard by the words "undefined behavior" or by the omission of any explicit definition of behavior. There is no difference in emphasis among these three; they all describe "behavior that is undefined".

NOTE 1 — Implementations claim conformance to particular features of the PAPI Learner Standard in their Implementation Conformance Statement (ICS).

NOTE 2 — For statements that apply to both strictly conforming and conforming implementations, i.e., a parallel construction, the abbreviation SC/C may be used. Example: "The difference between a SC/C data set, a SC/C coding, and a SC/C data instance is X" is equivalent to the two statements: (1) "The difference between a strictly conforming data set, a strictly conforming coding, and a strictly conforming data instance is X", and (2) "The difference between a conforming data set, a conforming coding, and a conforming data instance is X".

4.1 Conformance level

The following subclauses define strictly conforming implementations and conforming implementations. In the context of conformance, the terms "support", "use", "test", "access", and "probe" are defined in subclause 4.3, Coding Conformance, subclause 4.4 API Conformance, subclause 4.5, Protocol Conformance, and subclause 4.6, Data Application Conformance.

4.1.1 Subsets

Implementations shall indicate which Parts of the PAPI Learner Standard are supported in their ICS and in the conformance label(s). See subclause 4.2, Conformance Labels.

NOTE — Implementations should use automated techniques to convey subset information so that interoperability problems can be avoided.

4.1.2 Strictly conforming implementations

A strictly conforming implementation shall be at least one of: a strictly conforming coding, a strictly conforming API, a strictly conforming protocol, or a strictly conforming data application.

A strictly conforming implementation:

1. shall support all mandatory and optional data elements;
2. shall not use, test, access, or probe for any extension features or extended data elements;
3. shall not exceed limits or smallest permitted maximum values specified by this Standard; and
4. shall not interpret or generate data elements that are dependent on any unspecified, undefined, implementation-defined, or locale-specific behavior.

NOTE — The use of extension features or extended data elements is undefined behavior.

4.1.3 Conforming implementations

A conforming implementation shall be at least one of: a conforming coding, a conforming API, a conforming protocol, or a conforming data application.

A conforming implementation:

1. shall support all mandatory and optional data elements;
2. may use, test, access, or probe for extension features or extended data elements, as permitted by the implementation and data interchange participants, as long as the meaning and behavior of strictly conforming implementations is unchanged;
3. shall not support or use extension features or extended data elements that change the meaning or behavior of strictly conforming implementations;
4. may exceed limits or smallest permitted maximum values specified by this Standard, and to the extent permitted by the implementation; and
5. may interpret or generate data elements that are dependent on implementation-defined, locale-specific, or unspecified behavior.

NOTE 1 — The use of extension features or extended data elements is undefined behavior.

NOTE 2 — All strictly conforming implementations are also conforming implementations.

NOTE 3 — An implementation does not conform to this Standard if it redefines Standard features via extension methods, and these features change the meaning or behavior of strictly conforming implementations.

4.2 Conformance labels

A conformance label may summarize ICSs. Annex B, Conformance Labels, describes the requirements for human-readable and machine-readable conformance labels.

4.3 Coding conformance

A strictly conforming PAPI Learner coding shall be at least one of: a strictly conforming data set, or a strictly conforming data instance.

A conforming PAPI Learner coding shall be at least one of: a conforming data set, or a conforming data instance.

4.3.1 Data set conformance

Data set conformance is independent of binding.

A strictly conforming data set shall be a set of data that: (1) is structured independent of binding, (2) strictly conforms to the functionality, conceptual model, and semantics of the PAPI Learner Standard, (3) shall include all mandatory data elements, (4) may include optional data elements, and (5) shall not include extended data elements.

A conforming data set shall be a set of data that: (1) is structured independent of binding, (2) conforms to the functionality, conceptual model, and semantics of the PAPI Learner Standard (3) shall include all mandatory data elements, (4) may include optional data elements, and (5) may include extended data elements.

Conformity assessment of data sets shall be performed by (1) rendering the data set in ISO/IEC 11404 notation, and (2) verifying the requirements described by this Standard.

4.3.2 Data instance conformance

A strictly conforming data instance shall (1) be a strictly conforming data set, and (2) strictly conform to at least one PAPI Learner coding.

A conforming data instance shall (1) be a conforming data set, and (2) conform to at least one PAPI Learner coding.

NOTE 1 — The term "PAPI Learner coding" is used in the two paragraphs above and its requirements are specified in the third paragraph of subclause 4.3, Coding Conformance.

NOTE 2 — The difference between a SC/C data set, a SC/C coding, and a SC/C data instance is: (1) a data set is an instance of data that is independent of binding, (2) a coding can refer to an instance of data, a set of instances of data, or a syntax of instances of data, and (3) a strictly conforming/conforming data instance is associated with a specific binding.

Definitions: support, use

In the context of conformance, the terms "support" and "use" are defined individually in each PAPI Learner coding binding.

Definitions: test, access, probe

In the context of conformance, the terms "test", "access", and "probe" are defined as the null operation, i.e., for data instance conformance, the operations "test", "access", and "probe" perform no operations and have no effect.

4.4 API conformance

A strictly conforming PAPI Learner API shall strictly conform to at least one PAPI Learner API binding.

A conforming PAPI Learner API shall conform to at least one PAPI Learner API binding.

NOTE 1 — The conformance paradigm of API bindings is comprised of two parts: the API application and the API environment.

Definitions: support, use, test, access, probe

The following terms are defined in the context of API conformance for data interchange participants:

- A "supported" feature is one that is implemented by the API environment and may be used by any API application.
- A feature is "used" if it is read, written, or operated upon by an API application.
- A feature is "tested" if an API application inquires about the existence of that feature in the API environment.
- A feature is "accessed" if an API application attempts to read or write data associated with that feature.
- A feature is "probed" if an API application implicitly tests the existence of that feature by attempting to *use* the feature (see "use" above) within an API environment that permits error recovery.

NOTE 2 — API conformance makes requirements upon all data interchange participants: the API environment (implementations of the interface, services, resources, etc., of the API binding); and the API application (applications that use the API binding).

4.5 Protocol conformance

A strictly conforming PAPI Learner protocol shall strictly conform to at least one PAPI Learner protocol binding.

A conforming PAPI Learner protocol shall conform to at least one PAPI Learner protocol binding.

NOTE 1 — The conformance paradigm of protocol bindings is comprised of three parts: the client service, the server service, and the peer service.

Definitions: support, use, test, access, probe

The following terms are defined in the context of protocol conformance for data interchange participants:

- A "supported" feature is one that is implemented by the client, server, or peer service and may be used by any client, server, or peer service within the appropriate role among the data interchange participants, e.g., client-server, peer-to-peer.
- A feature is "used" if it is transmitted, received, or controlled by a client, server, or peer service.
- A feature is "tested" if client, server, or peer service inquires about or negotiates for the existence of that feature with other data interchange participants.
- A feature is "accessed" if it is transmitted or received by a client, server, or peer service.
- A feature is "probed" if a client, server, or peer service implicitly tests the existence of that feature by attempting to use the feature (see "use" above) with other data interchange participants that permit error recovery.

NOTE 2 — Protocol conformance makes requirements upon all data interchange participants: the client service, the server service, and the peer service.

4.6 Data application conformance

There are two types of data application conformance: strictly conforming and conforming.

A strictly conforming data application is a data application that strictly conforms to the PAPI Learner Standard.

NOTE 1 — A strictly conforming data application *may be minimally conforming but is maximally interoperable with respect to the PAPI Learner Standard*. Strict conformance concerns (1) the assessment, measurement, and/or availability of a minimal set of features; (2) the data application's non-use of feature-probing; and (3) the data application's non-use of extended feature sets.

A conforming data applications is a data application that conforms to the PAPI Learner Standard.

NOTE 2 — A conforming data application *may be more useful, but may be less interoperable with respect to the PAPI Learner Standard*. Conformance concerns (1) the assessment, measurement, and/or availability of a minimal set of features; (2) feature-probing for and/or prior agreement to the existence (or availability) of extended features, as permitted by the implementation; and (3) extended features specified external to this Standard.

There are three types of SC/C data applications: data repository, data reader, data writer.

4.6.1 Data repository

A strictly conforming data repository shall:

1. receive data sets for subsequent retrieval;
2. use strictly conforming data interpretation for receiving data sets;
3. store data sets in persistent storage so that data extensions may not persist;
4. send, on request, previously stored data sets;
5. use strictly conforming data generation for sending data sets;
6. strictly conform to at least one PAPI Learner coding binding; and
7. strictly conform to at least one PAPI Learner API or PAPI Learner protocol binding.

NOTE 1 — A strictly conforming data repository does not require "preservation" of extended data elements, i.e., data interchange should not be dependent upon expecting extended data elements to persist in a strictly conforming data repository but does not prohibit it either. See IEEE 1484.2.2, Subclause x.x.x, Both Strictly Conforming and Conforming Data Applications, for more information on the storage of extensions in strictly conforming data repositories.

A conforming data repository shall:

1. receive data objects for subsequent retrieval;
2. use conforming data interpretation for receiving data sets;
3. store data sets in persistent storage so that data extensions may persist;
4. send, on request, previously stored data objects;
5. use conforming data generation for sending data sets;
6. conform to at least one PAPI Learner coding binding; and
7. conform to at least one PAPI Learner API or PAPI Learner protocol binding.

NOTE 2 — A conforming data repository may, upon storage, add, delete, or change extended data elements for subsequent retrieval.

NOTE 3 — A conforming data repository may store *some* data extensions, but it is not required to store and retrieve *all* data extensions.

NOTE 4 — A conforming data repository may store and retrieve data objects that are not data sets.

4.6.2 Data reader

A strictly conforming data reader shall interpret data that strictly conforms to (1) the PAPI Learner Standard, and (2) at least one binding of the PAPI Learner Standard.

NOTE 1 — A strictly conforming data reader does not interpret extended data elements.

NOTE 2 — Depending upon the binding of the PAPI Learner Standard, a strictly conforming data reader may "ignore" data extensions, e.g., a strictly conforming data reader may consume data extensions but the data reader is able to ignore (not interpret) these extensions.

A conforming data reader shall interpret data that conforms to (1) the PAPI Learner Standard, and (2) at least one binding of the PAPI Learner Standard.

NOTE 3 — A conforming data reader may interpret extended data elements.

4.6.3 Data writer

A strictly conforming data writer shall generate data that strictly conforms to (1) the PAPI Learner Standard, and (2) at least one binding of the PAPI Learner Standard.

NOTE 1 — A strictly conforming data writer does not generate extended data elements.

A conforming data writer shall generate data that conforms to (1) the PAPI Learner Standard, and (2) at least one binding of the PAPI Learner Standard.

NOTE 2 — A conforming data writer may generate extended data elements.

4.7 Registry conformance

A conforming implementation that uses datatypes that are defined by a registry shall conform to those requirements of that (dependent) registry. A conforming implementation shall, within 6 months of date of publication, incorporate all updates to dependent registries.

5 Functionality

A PAPI Learner implementation shall participate in data interchange of learning-related human information (learner information) for use by learning technology systems. A PAPI Learner implementation shall represent learner information or shall communicate learner information among PAPI Learner data applications. A PAPI Learner implementation participates in the data interchange as one or more of the following roles: coding (data set or data instance), API, protocol, or data application (data repository, data reader, data writer).

PAPI Learner implementations may affect the transfer of data via:

- PAPI Learner application programming interfaces (APIs);
- PAPI Learner protocols; or
- methods outside the PAPI Learner Standard.

PAPI Learner codings and data formats may be used to encode the data for any of the above methods.

6 Conceptual model

This Clause defines the conceptual model of conforming PAPI Learner implementations.

The PAPI Learner Standard describes a specification and a framework of human information data exchange, representation, interfaces, services, and repositories that are, conceptually, partitioned according to their applications' use and administration.

6.1 Learner information types

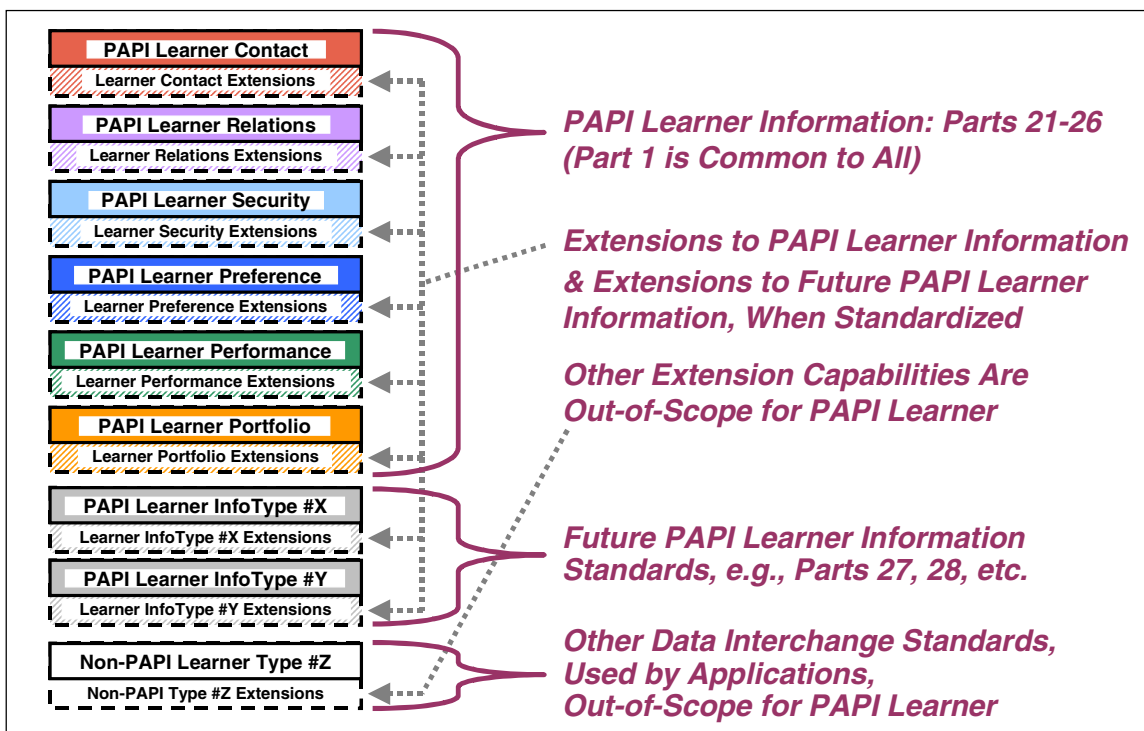


Figure 1. Relationships among PAPI Learner information (the PAPI Learner Standard), future enhancements to the PAPI Learner Standard, extensions to these information types, and non-PAPI Learner information.

Learner information, also known generically as a "learner profile", is a subset of general information about learning technology. Learner information includes contact, preference, performance, portfolio,

and, possibly, other types of information. The PAPI Learner Standard describes a *particular subset* of all possible types of learner information, i.e., in this edition of the PAPI Learner Standard there are six information types (Parts 21-26), but future editions of the PAPI Learner Standard may include the additional information types.

NOTE 1 — For each of the information types, the PAPI Learner Standard describes a subset that is useful and can be widely implemented. The PAPI Learner Standard does not describe all possible learner information, but it includes only the minimum information necessary to satisfy the functional requirements and to be maximally portable, and it includes the ability to extend this information.

The following is a brief description of the information types of the PAPI Learner Standard. Throughout the PAPI Learner Standard, one-letter mnemonics and colors are used as a shorthand for types of PAPI Learner information types.

- **Learner contact information (Part 21)** is not directly related to the measurement and recording of learner performance and is primarily related to administration. NOTE — Typically, this type of information is private and secure. Shorthand: **C**, the mnemonic is "**Con-tact**". The color is red or rose.
- **Learner relations information (Part 22)** is about the learner's relationship to other users of learning technology systems, such as teachers, proctors, and other learners. Shorthand: **R**, the mnemonic is "**Relations**". The color is violet or light violet.
- **Learner security information (Part 23)** is about the learner's security credentials, such as: passwords, challenge/responses, private keys, public keys, biometrics. Shorthand: **S**, the mnemonic is "**Security**". The color is sky blue or light sky blue.
- **Learner preference information (Part 24)** describes preferences that may improve human-computer interactions. Shorthand: **M**, the mnemonic is "**My configuration**". The color is deep blue or pastel blue.
- **Learner performance information (Part 25)** relates to the learner's history, current work, or future objectives and is created and used by learning technology components to provide improved or optimized learning experiences. Shorthand: **G**, the mnemonic is "**Grades**". The color is deep green or pastel green.
- **Learner portfolio information (Part 26)** is a representative collection of a learner's works or references to them that is intended for illustration and justification of his/her abilities and achievements. Shorthand: **W**, the mnemonic is "**Works**". The color is goldenrod or light goldenrod.

NOTE 2 — Implementations may extend or combine these information types. Implementations may link separate data repositories of information types via, say, database keys.

Example: A data repository of learner contact information is linked to a data repository of learner performance information by the use of a learner identifier.

NOTE 3 — The PAPI Learner Standard does not require these information types to be separated, but many implementations maintain separate repositories to satisfy security, administration, regulatory, and system performance needs.

6.2 Public and private information

Some information may be available to the public, some information may have limited accessibility to the public, some information may be private, and other combinations are possible. PAPI Learner information may be partitioned and administered and secured separately, e.g., learner contact information is private and secure, while the learner portfolio information is public. The public, private, semi-public, etc., nature of the learner information is chosen by the administrators (e.g., institutional administrators, or learners themselves for personal repositories) and the requirements for this choice is outside the scope of this Standard.

6.3 Information types vs. data repositories

Conceptually, (1) the PAPI Learner Standard is a collection of information types, (2) the PAPI Learner Standard is *not* a "master student/learner record", (3) learner information is partitioned into data repositories according to information type. The emphasis is on the (conceptual) partitioning of learner information. This partitioning permits separate security, administration, and access all within a simple framework.

A data repository may hold one or more information types. A data set may hold one or more information types.

NOTE — Data repository administrators are free to optimize information organization, such as combining information when the security, administration, access, etc., needs can be adequately addressed with the combined information.

Example 1: Separate Data Repositories: A conforming implementation might have one data repository for each information type: a learner contact information repository, a learner relations information repository, a learner security information repository, a learner preference information repository, a learner performance information repository, a learner portfolio information repository.

Example 2: Combined Data Repository: A conforming implementation might have one repository for all types of learner information. In this example, the security administrator might allow unrestricted access to "learner preference data elements" (i.e., learner preference information), while restricting access to "learner contact data elements" (i.e., learner contact information); thus learner contact information is "private" and learner preference information is "public", yet they are both accessed via the same data repository. The application or user might be unaware that the same data repository is accessed for each information type.

6.4 Common features, information types, and bindings

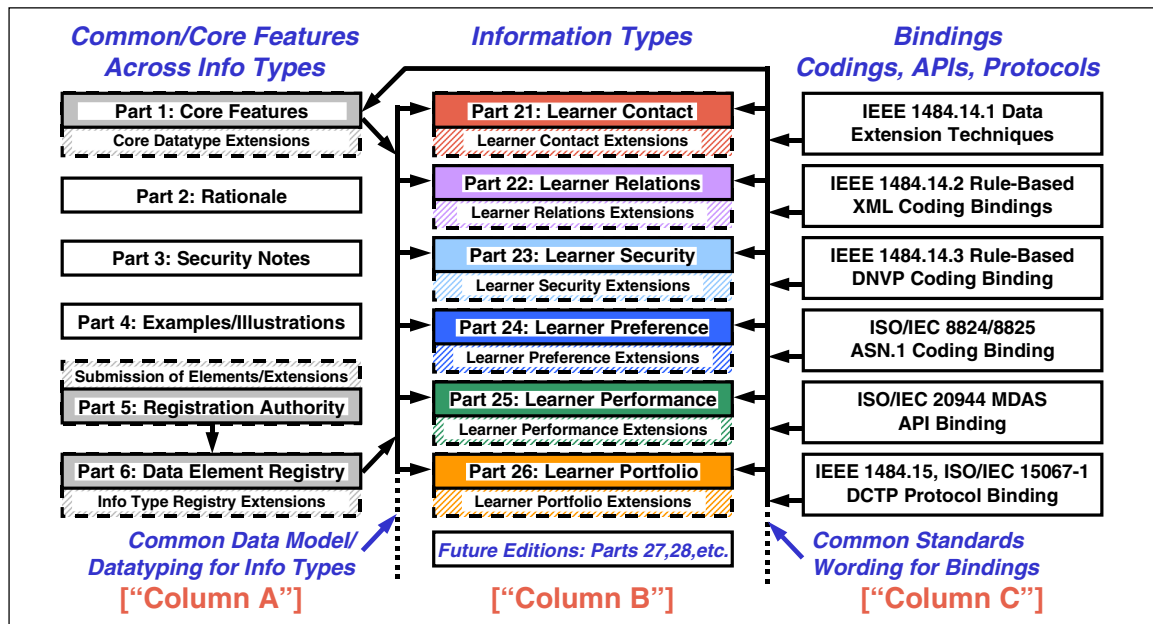


Figure 2. Relationship among the Parts of the PAPI Learner Standard, and relationship to other standards, specifications, guides, and technical reports.

Conceptually, the PAPI Learner Standard is organized in three areas:

- **Common Features (Parts 1-6):** Implementations may conform to Part 1 (Core Features) and Part 6 (Data Element Registry). The core features include datatypes that are common across other Parts of the PAPI Learner Standard. The data element registry contains information about data elements, e.g., enumerated value spaces, "permissible values", etc., that may be updated frequently (too frequent for the standards process). The data element registry may be available on-line on the internet. Part 5, Registration Authority Process, describes how committees maintain the data element registry, i.e., committees (not IT implementations) conform to Part 5. Part 2 (Rationale), Part 3 (Learner Information Security Issues), and Part 4 (Examples and Illustrations) are informative Guides, i.e., an implementation would not claim conformance to Parts 2-4.
- **Information Types (Parts 21-26):** These are the learner information types, as described above in subclause 6.1. Future editions of the PAPI Learner Standard may include additional information types, e.g., Part 27, Part 28, etc..
- **Bindings (codings, APIs, protocols):** The PAPI standards is mapped to various standards, specifications, and technical reports.

Implementations may be characterized by the choices from column A, column B, and column C. For example, an implementation conforms to Parts 1 and 6 (column A) and Part 25 (column B) using a rule-based XML binding (column C).

6.5 Metadata model

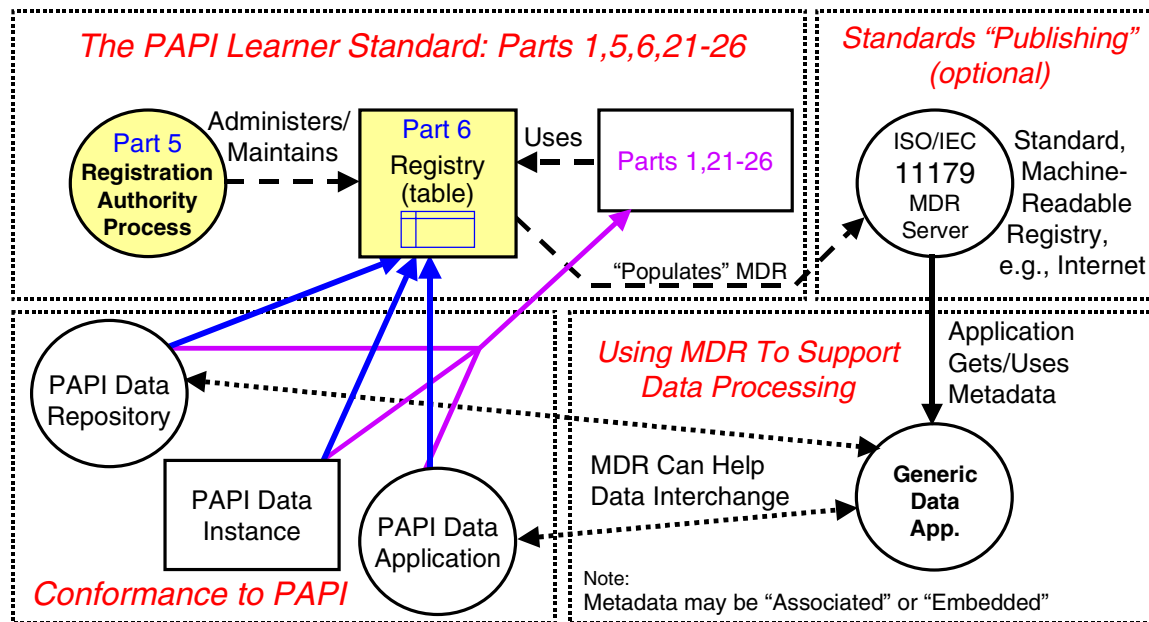


Figure 3. The metadata model, showing the use of metadata to describe certain PAPI Learner data elements, conformance to PAPI Learner metadata (Part 6), the (optional) "publishing" of metadata, the operational use of PAPI Learner metadata.

Metadata is descriptive information about data elements. Significant portions of PAPI Learner Parts 1, 6, and 21-26 contain normative wording that is metadata. Metadata may be structured in a tabular form (e.g., a metadata registry) or organized differently. Metadata may be stored and retrieved electronically (e.g., an electronic metadata registry), or may be stored in some form that does not afford automated processing (e.g., a paper-based standard).

Metadata registries, as described in ISO/IEC 11179, may be used to describe all PAPI Learner data elements, but the use of ISO/IEC 11179 is not required by this Standard. ISO/IEC 11179 is used in Part 6 in that the standard uses ISO/IEC 11179 as a descriptive and notational technique, but does not require PAPI Learner implementations to use ISO/IEC 11179.

NOTE — In other words, it is possible that strictly conforming and conforming PAPI Learner implementations do not use ISO/IEC 11179 at all.

6.6 Data access model

A PAPI Learner data set is a collection of information about a learner in a learning technology system. This information may include one or more types of learner information, such as contact, relations, security, preference, performance, and portfolio information.

Data interchange of PAPI Learner data sets is affected by:

- **PAPI Learner codings.** A PAPI Learner data set may be coded in a PAPI Learner coding binding. The data interchange is facilitated among data exchange participants by mutual agreement via methods external to the PAPI Learner coding specification.
- **PAPI Learner application programming interfaces (APIs).** The API binding is a control transfer mechanism (control is transferred from caller to callee) that affects data interchange.

Copyright © 2001 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- **PAPI Learner protocols.** The protocol binding is both a control transfer mechanism and a data transfer mechanism.

Collectively, these bindings are called "PAPI Learner codings, APIs, and protocols".

NOTE — PAPI Learner data applications may use one or more PAPI Learner codings, APIs, and protocols.

The following is the conceptual model of data access:

- **Data Object Model.** A data object shall be at least one of: a data element, or an implementation-defined object.
- **Data Storage Model.** Data, including data sets, may be stored in a data object, as referenced by one or more identifiers.
- **Data Retrieval Model.** Data, including data sets, may be retrieved from a data object, as referenced by one or more identifiers.
- **Data Typing Model.** Data objects that are data elements shall have a datatype. Datatypes may prescribe certain value spaces (e.g., domains), representation, encoding, storage, layout, conversion to other types, methods, and operations. The datatype of PAPI Learner data elements is defined by the PAPI Learner Standard, which uses the semantics and notation of ISO/IEC 11404.
- **Data Attribute Model.** A data attribute shall be an implementation-defined object associated with a data object. These attributes themselves may be accessed as data objects. NOTE — Attributes are also known as "properties".
- **Data Repository Access Model.** PAPI Learner bindings define access, if any, to data repositories.
- **Data Repository Security Model.** See subclause 6.11, Security Model, below.
- **Data Persistence Model.** The lifetime of data objects is implementation-defined.
- **Data Navigation Model.** The techniques for navigating data structures are defined in PAPI Learner bindings.
- **Data Identification Model.** The identification, labeling, namespace, and their associated techniques are implementation-defined.
- **Data Referencing Model.** A data repository may create a reference to a data object for the purpose of subsequent dereference. The naming conventions, lifetime, and scoping of a reference are implementation-defined.
- **Data Dereferencing Model.** A data repository may access a data object based upon supplying a reference, i.e., dereferencing a reference. The dereferencing methods are implementation-defined.
- **Data Indexing Model.** The indexing methods for data repositories are implementation-defined. NOTE — The term "indexing" is used in the context of database systems, i.e., methods for organizing database records.
- **Data Searching Model.** The searching methods for data repositories are implementation-defined.

6.7 Extending PAPI within an application area

The specification of extended features is outside the scope of the PAPI Learner Standard.

NOTE — Even though information technology components may support a wide variety of implementations, data repository administrators may choose to include and/or limit certain extensions. See IEEE 1484.14.1, Data Extensions Techniques, for more information.

6.8 Limiting PAPI extensions

The specification or limitation of these extended features is outside the scope of the PAPI Learner Standard.

NOTE — Even though information technology components may support a wide variety of implementations, data repository administrators may choose to limit the availability or use of extensions. See IEEE 1484.14.1, Data Extensions Techniques, for more information.

6.9 Distance, distributed, and nomadic systems

Distance features are supported by permitting global namespaces and by the infrastructure to search, store, and retrieve information within a global namespace. The namespace conventions and resolution methods are implementation-defined.

Distributed features are permitted by a single repository storing one or more information types; and by one or more repositories acting as a single repository. The synchronization, replication, commit, rollback, and fallback methods are implementation-defined.

Nomadic features are permitted by roaming users within a sometimes-connected infrastructure. The methods of dynamic quality of service and continuity of connection are implementation-defined.

6.10 Correlation of granularity levels

The correlation of various levels of granularity for learner performance information are implementation-defined.

Example: If a teacher changes a course grade (smaller granularity), when does the grade point average (larger granularity) automatically change?

6.11 Security model

Security is defined and bounded by a security perimeter. The following features are implementation-defined:

- The boundary of security perimeter(s).
- The nature, type, and acceptable level of risk of inbound security threats.
- The nature, type, and acceptable level of risk of outbound security threats.
- The security strength.
- The parameterization, setup, negotiation, and knockdown of security features.
- The administration of the security perimeter integrity.

The interoperability agreements may be defined externally to the PAPI Learner Standard.

NOTE — The security model of the PAPI Learner Standard is harmonized with ISO/IEC 17799-1, Code of Practice for Information Security Management.

The following security features are defined in the conceptual model:

- **Session-View Security Model.** Security features are provided on a per-session, per-view basis. Each security session is initiated by an accessor (a user or agent that requests access). The accessor provides security credentials that authenticate the accessor, authorize the accessor, or both. A view represents a portion of PAPI Learner information; a "view" is similar to the notion of a database "view". Each view that is *established* represents a session, i.e., the "session" represents the duration of access and the "view" represents the scope of access.
- **Security Parameter Negotiation Model.** Data interchange participants negotiate security parameters prior to, during, and after each session. The security parameters are defined in the bindings of the PAPI Learner Standard, e.g., Parts 1, 6, 21-26.
- **Security Extension Model.** Additional security features may be used that were not foreseen. The method of incorporating extensions is defined in the bindings of the PAPI Learner Standard, e.g., Parts 1, 6, 21-26.
- **Access Control Model.** Accessors may attempt to read data elements, may attempt to write data elements, may attempt to create new data elements (separately or within aggregates), may attempt to destroy data elements (separately or within aggregates), and may attempt to change attributes of data elements. Other access methods, if any, are implementation-defined.
- **Identification Model.** The methods for identifying learners are implementation-defined. NOTE — IEEE 1484.13, Simple Human Identifiers, defines the datatype associated with a learner identifier.
- **Authentication Model.** The methods of authenticating users are outside the scope of this Guide and outside the scope of the PAPI Learner Standard..
- **De-identification Model.** All information, except learner contact information, should be de-identified. The methods of de-identifying learners and their information are outside the scope of this Guide and outside the scope of the PAPI Learner Standard. NOTE — Administrators and, possibly, learners are responsible for choosing appropriate learner identifiers that do not reveal the learner's identity. Example: Choosing a learner's name as his/her identifier is a poor choice from the perspective of de-identification. A variety of better de-identification techniques are possible, including the use of random numbers with short "lifetimes".
- **Authorization Model.** The methods of authorizing operations are implementation-defined.
- **Delegation Model.** The methods of delegating administration, authority, or credentials are implementation-defined.
- **Non-Repudiation Model.** The methods of non-repudiation are implementation-defined.
- **Repudiation Model.** The methods of repudiating data, users, or credentials are implementation-defined.
- **Privacy Model.** The PAPI Learner Standard does not specify a privacy model and does not specify privacy requirements, but supports security frameworks and approaches that permit the implementation of a wide variety of privacy frameworks.
- **Confidentiality Model.** The PAPI Learner Standard does not specify a confidentiality model and does not specify confidentiality requirements, but supports access controls and the

partitioning of information types that permit the implementation of a wide variety of confidentiality frameworks.

- **Encryption Model.** The PAPI Learner Standard does not specify an encryption model and does not specify encryption requirements, but supports several security frameworks and techniques that permit the integration of various encryption models and technologies.
- **Data Integrity Model.** The PAPI Learner Standard does not specify a data integrity model and does not specify data quality requirements, but supports information assurance frameworks and approaches that permit the implementation of a wide variety of data integrity frameworks
- **Validation of (Learner) Certificates.** The PAPI Learner Standard does not require validation of learner performance information or learner portfolio information, but supports the parameterization of automated validation of either type of learner information.
- **Digital Signature Model.** The PAPI Learner Standard does not specify a digital signature model and does not specify digital signature requirements, but supports several signing frameworks and techniques permit the integration of various digital signature models, policies, and technologies. This Digital Signature Model is harmonized with ISO/IEC 15945 "Specification of Trusted Third Party Services to Support the Application of Digital Signatures".

IEEE 1484.2.3, PAPI Learner Information Security Notes, contains information about applying security techniques and technologies to PAPI Learner implementations.

7 Semantics

The meaning of PAPI Learner information shall be the same regardless of which PAPI Learner binding and PAPI Learner encoding is chosen.

NOTE 1 — The choice of PAPI Learner bindings and/or PAPI Learner encodings is for the convenience of the data interchange participants.

NOTE 2 — PAPI Learner coding bindings define a mapping for each datatype defined in this Clause.

NOTE 3 — PAPI Learner API bindings define a mapping for each operation and datatype defined in this Clause.

NOTE 4 — PAPI Learner protocol bindings define a mapping for each operation and datatype defined in this Clause, and define methods for setup and knockdown of supporting communication networks.

NOTE 5 — Throughout this Clause, the abbreviation SPM is used, which means "smallest permissible maximum". The SPM value is intended to give implementers a lower limit on conforming implementations. Applications should not assume that implementations support capabilities beyond the SPM value unless prior arrangements have been made.

7.1 General data operations

PAPI Learner APIs, protocol, and repositories shall support the following data management operations on data sets (see Note at end of this subclause):

- **Create Operation.** Creating a new instance of some information type, such as contact information in some storage context. NOTE — Compare the lifetime of records created in application memory, in temporary storage, and in a database.
- **Destroy Operation.** Discarding an instance of an information type in the context of its storage. NOTE — Compare destroying a record using various techniques: erasure, lazy storage management, garbage collection.
- **Erase Operation.** Obliterating the contents of the information type so that it cannot be recovered. NOTE — The Destroy operation only implies that the data is no longer available to an application; the Destroy operation does not imply that the data is unrecoverable.
- **Copy Operation.** Creating a new instance of an information type with identical contents.
- **Move Operation.** Changing a label that is associated with an instance of an information type by changing the storage of the information (implicit label change) or changing the label itself (explicit label change). Example: An implicit label change might be affected by copying the data from source to target (e.g., copy the fifth element to the third), then erasing data at the source; the label is now implicitly change from source to target (e.g., the data is now the third element). An explicit label change might be affected by changing the label in some "directory" of information.
- **Label Operation.** Creating (or removing) a name, specified by the "caller", to be associated with an instance of information.
- **Navigate Operation.** Using a naming method (absolute, relative, complete, progressive) to locate an instance of an information type.
- **Search Operation.** Finding instances of an information type that match search criteria and returning the found information via references, labels, or copies.
- **Reference Operation.** Creating a handle to an instance of an information type. NOTE — The difference between a Label operation and a Reference operation is: the "caller" chooses the name for a label, while the "callee" chooses the name for a reference.
- **Dereference Operation.** Using a handle, created through reference, to access an instance of an information type.
- **Aggregation Operation.** Combining several instances of one or more information types into a single container.
- **Decomposition Operation.** Extracting instances of information types from a container.

NOTE — Certain operations may be impossible, such as creating or destroying data on read-only media. In these cases, the operations are still required to exist, but their usage causes run-time errors, e.g., an API must still support a Create operation, but the Create operation, when attempted, causes an error.

7.2 Application-specific data operations

PAPI Learner APIs, protocol, and repositories may support the following application-specific data operations on data sets:

- **Search Operation.** PAPI Learner records may be search, based on criteria. Examples: "which learners received a grade below 60?", "which learners portfolios have a movie that they have created?"
- **Sort Operation.** PAPI records may be ordered, based upon sort criteria. Example: users may be ordered alphabetically by name.
- **Join Operation.** Tables of information may be combined along certain columns.
- **Accumulation Operation.** PAPI Learner records may be accumulated, aggregated, or analyzed. Examples: "what it the average score among third graders?", "what is the grade point average for a particular learner?"
- **Time Series Compression and Expansion Operations.** PAPI Learner records may be recorded at various levels of granularity. Time compression reduces the set of records to larger granularity. Time expansion creates records of finer granularity by interpolation. Example: quarterly grades and a final exam are rolled up into a final grade; after compression, the data set is reduced because only the final grade remains.

NOTE — Applications that wish to use these features should "probe" the implementation to determine whether or not the feature is supported. See the API binding, protocol binding, and repository specification for more information.

7.3 Data compatibility

PAPI Learner APIs, protocol, and repositories may support the following data compatibility operations on data sets:

- **Promotion of Data Types.** Transforming a value from lesser capabilities to greater capabilities. Example: Converting an integer to a real; converting a character array of length 10 to a character array of length 20; converting a record of elements {`name`, `address`} to a record of a superset of elements {`name`, `address`, `telephone`}.
- **Demotion of Data Types.** Transforming a value from greater capabilities to lesser capabilities. Example: Converting a real to an integer; converting a character array of length 20 to a character array of length 10; converting a record of elements {`name`, `address`, `telephone`} to a record of a subset of elements {`name`, `address`}.
- **Conversion To/From Data Types.** Transforming a value from one data type to another data type. Example: Converting an integer to a string; converting the pair {`primary-name`, `secondary-name`} to {`sort-name`}.
- **Text Formats.** Transforming values to/from textual representations.

NOTE — Applications that wish to use these features should "probe" the implementation to determine whether or not the feature is supported. See the API binding, protocol binding, and repository specification for more information.

7.4 Foundational datatypes

The following datatypes are used by more than one PAPI Learner information type.

In this subclause and the remaining subclauses of this Clause, the ISO/IEC 11404 summary provides additional information, such as size and smallest permitted maximum values, if not already provided in the Description wording.

7.4.1 `ssd_record`(semi-structured data)

ISO/IEC 11404 summary

```
type ssd_record(type_description) = // see normative description below
```

Description

The `ssd_record` datatype is semi-structured data that is limited to the datatypes in the `type_description`. The parameters to the `ssd_record` datatype are identical to the parameters to the `record` datatype generator. Elements of the semi-structured data may have labels, i.e., identifiers associated with one or more elements, are of the datatype `papi_learner_context_label_type`. Elements of the semi-structured may be accessed by index (e.g., 0, 1, 2, ...). Elements of the semi-structured data may be accessed by label (e.g., `name`, `address`, `phone`, etc.).

The `ssd_record` is a core datatype that is used to generate other aggregate datatypes.

NOTE — Although the use of `ssd_records`(`type_description`) imposes provisions (e.g., assertions, constraints, etc.) upon a generated datatype, additional provisions associated with the generated datatype may be described in normative wording outside the use of `ssd_record`().

Example

The datatype:

```
type myrecord =
  ssd_record
  (
    name : name_type,
    telephone : telephone_type,
    email : email_type,
    postal : postal_type,
  ),
```

describes a datatype that might permit the following instances within an XML binding.

```
<!-- Example #1 -->
<myrecord>
  <name>nnn</name>
  <phone label="home">111</phone>
  <phone label="emergency">222</phone>
  <email>x@y.com</email>
  <postal>123 Main Street</postal>
</myrecord>

<!-- Example #2 -->
<myrecord>
  <list_element label="home">
    <name>nnn</name>
```

```

        <phone>111</phone>
        <email>x@y.com</email>
        <postal>123 Main Street</postal>
    </list_element>
    <list_element label="work">
        <name>Ms. nnn</name>
        <phone>333</phone>
        <email>x@work.com</email>
        <postal>234 Work way</postal>
    </list_element>
</myrecord>

```

7.4.2 papi_learner_context_label_type

ISO/IEC 11404 summary

```

type papi_learner_context_label_type =
    characterstring(iso-10646-1), // SPM: 500

```

Description

All sizes or limits are smallest permitted maximum values.

A **characterstring** of size 500. The meaning of the values are implementation-defined.

Example

The value "**!(japan)**" might be "not within the context of Japan" — whatever that means for the implementation.

7.4.3 arraylist

ISO/IEC 11404 summary

```

type arraylist(type_spec,size) =
    array (0..size-1) of (type_spec),

```

Description

A shorthand for multiple array elements. This type declaration is only a "synonym" (in the ISO/IEC 11404 sense of a "**new**" type) for an existing type.

7.4.4 locstring_type

ISO/IEC 11404 summary

```

type locstring_type =
record
(
    locale:
        characterstring(iso-646), // SPM: 255
    string:
        characterstring(iso-10646-1), // SPM: 1000
),

```

Description

A localized string. The following components define this data element. All components are optional. All sizes or limits are smallest permitted maximum values.

- **locale:** The localization (L10N) mapping, known as the "locale".
- **string:** The localized string itself.

Example

The following are sample ISO/IEC 11404 (data set) values:

```
// ISO/IEC 11404 data set
(
    locale = "en-US",
    string = "hello world",
),
```

7.4.5 mcstring_array

ISO/IEC 11404 summary

```
type mcstring_array_type(limit) =
    array (0..limit-1) of (locstring_type),
```

Description

An array of multicultural strings, also known as a "message catalog features". This type declaration is only a "synonym" (in the ISO/IEC 11404 sense of a **"new"** type) for an existing type.

7.4.6 locvalue_type

ISO/IEC 11404 summary

```
type locvalue_type(datatype) =
record
(
    locale:
        characterstring(iso-646), // SPM: 255
    value:
        (datatype),
),
```

Description

A localized value of the specified datatype. The following components define this data element. All components are optional. All sizes or limits are smallest permitted maximum values.

- **locale:** The localization (L10N) mapping, known as the "locale".
- **value:** The localized value (data object) itself.

Example

The following are sample ISO/IEC 11404 (data set) values:

```
// ISO/IEC 11404 data set
```

```
(
    locale = "en-US",
    // The localized datatype in this example is a person's name.
    value = (primary: "Doe", secondary: "John" ),
),
```

7.4.7 mcvalue_array

ISO/IEC 11404 summary

```
type mcvalue_array_type(limit) =
    array (0..limit-1) of (locvalue_type),
```

Description

An array of multicultural values. This type declaration is only a "synonym" (in the ISO/IEC 11404 sense of a "new" type) for an existing type.

7.4.8 papi_learner_nvp_bucket_type

ISO/IEC 11404 summary

```
type papi_learner_nvp_bucket_type =
record
(
    name:
        characterstring(iso-10646-1), // SPM: 200
    value:
        octetstring, // SPM: 4096
),
```

Description

A "bucket" for adding name-value pairs (NVPs) to a PAPI Learner information type. The following components define this data element. All components are optional. All sizes or limits are smallest permitted maximum values.

- **name:** The name portion of the name-value pair.
- **value:** The value portion of the name-value pair.

NOTE — This feature permits a limited extension capability that works on all strictly conforming systems.

Example

The following are sample ISO/IEC 11404 (data set), XML (data instance), and DNVP (data instance) values:

```
// ISO/IEC 11404 data set
(
    name = "special_parameter_1",
    value = "xyz",
),

<!-- XML data instance ("..." is replaced by outer tags) -->
<....>
```

```

    <name>special_paramter_1</name>
    <value>xyz</value>
</...>

```

```

#### DNVP data instance ("..." is replaced by outer context)
....name: special_parameter_1
....value: xyz

```

7.4.9 papi_learner_identifier_type

ISO/IEC 11404 summary

```

type papi_learner_identifier_type =
record
(
    identifier_kind :
        papi_learner_identifier_kind_type,
    identifier_value :
        octetstring, // SPM: 1024
),

```

Description

An internal identifier that is used to link across databases. The meaning, namespace, scoping, and resolution of this feature is implementation-defined.

The following components define this data element. All components are optional. All sizes or limits are smallest permitted maximum values.

- **identifier_kind**: The kind of the identifier.
- **identifier_value**: The value of the identifier.

Example

The following are sample ISO/IEC 11404 (data set), XML (data instance), and DNVP (data instance) values:

```

// ISO/IEC 11404 data set
(
    identifier_kind = "pointer",
    identifier_value = "0x12345678",
),

<!-- XML data instance ("..." is replaced by outer tags) -->
<...>
    <identifier_kind>pointer</identifier_kind>
    <identifier_value>0x12345678</identifier_value>
</...>

#### DNVP data instance ("..." is replaced by outer context)
....identifier_kind: pointer
....identifier_value: 0x12345678

```

7.4.10 papi_learner_hid_type

ISO/IEC 11404 summary

```

type papi_learner_hid_type =
record
(
  identifier_type :
    papi_learner_identifier_type_type,
  identifier_value :
    octetstring, // SPM: 1024
),

```

Description

An external identifier that is used to correlate PAPI Learner information across repositories. The meaning, namespace, scoping, and resolution of this feature is implementation-defined.

The following components define this data element. All components are optional. All sizes or limits are smallest permitted maximum values.

- **identifier_type**: The type of the identifier.
- **identifier_value**: The value of the identifier.

Example

The following are sample ISO/IEC 11404 (data set), XML (data instance), and DNVP (data instance) values:

```

// ISO/IEC 11404 data set
(
  identifier_type = "ISO_IEC_21484_13",
  identifier_value = "00112233",
),

<!-- XML data instance ("..." is replaced by outer tags) -->
<...>
  <identifier_type>ISO_IEC_21484_13</identifier_type>
  <identifier_value>00112233</identifier_value>
</...>

#### DNVP data instance ("..." is replaced by outer context)
....identifier_type: ISO_IEC_21484_13
....identifier_value: 00112233

```

7.4.11 papi_learner_identifier_type_type

ISO/IEC 11404 summary

```

// This datatype describes the identifier type (URL, key,
// pattern, etc.).
type papi_learner_identifier_type_type =
  octetstring, // string length SPM: 256

```

Description

The varieties of PAPI Learner identifiers are describe by this datatype. IEEE 1484.2.6, Clause 10.2, Learner Identifier Kinds, further constrains the value space of this datatype, as described by a registry in that Clause.

NOTE — A conforming implementation shall conform to the values of the registry.

7.4.12 papi_learner_data_certification_type

ISO/IEC 11404 summary

```

type papi_learner_data_certification_type =
record
( // note: all components are optional; all sizes are SPM
  certification_source : // who certified the record
    octetstring, // string length SPM: 2048
  certification_method : // how it was certified
    octetstring, // string length SPM: 1024
  certification_parameter_list :
    octetstring, // string length SPM: 16384
  certification_subset : // which data elements were certified
    octetstring, // string length SPM: 1024
  certification_identifier : // ID associated with cert. event
    octetstring, // string length SPM: 2048
  certification_bucket : // other information, SPM: 100
    papi_learner_nvp_bucket_type,
),

```

Description

PAPI Learner data certification information is defined by this datatype. This datatype is used by the PAPI Learner performance and PAPI Learner portfolio information types.

The following components define this data element. All components are optional. All sizes or limits are smallest permitted maximum values.

- **certification_source**: Who certified this record.
- **certification_method**: What certification method was used.
- **certification_parameter_list**: The options and parameters necessary to validate the certification. NOTE — With the **certification_source**, **certification_method**, and **certification_parameter_list**, it is possible to validate the **certification_identifier**.
- **certification_subset**: A list of which elements in this record are certified. NOTE — This is necessary for generating automatic validation.
- **certification_identifier**: The identifier that is passed to the certification validator, i.e., the certificate ID.
- **certification_bucket**: A "bucket" for adding name-value pairs that provides limited extension capabilities to PAPI Learner data certificate information.

NOTE 1 — With the following elements:

```
certification_source
```

```
certification_method  
certification_parameter_list,  
certification_subset
```

it is possible to validate the `certification_identifier`.

NOTE 2 — The techniques for generating certificates and automatically validating certificates are outside the scope of this Standard.

8 Bindings and encodings

PAPI Learner information is bound to codings, APIs, and protocols; and encoded as data formats, calling conventions, and communication layers. Bindings and encodings are specified in annexes of this Part and other parts of this Standard.

9 Annex A: Bibliography (informative)

This Annex is informative and not normative.

The following are related documents:

- IEEE P1484.2.2/D8, Standard for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Rationale
- IEEE P1484.2.3/D8, Guide for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Security Information Notes
- IEEE P1484.2.4/D8, Guide for Learning Technology — Public and Private Information (PAPI) for Learners (PAPI Learner) — Examples and Illustrations
- IEEE P1484.20, Competency Definitions.
- ISO JTC1 CAW (Cultural Adaptability Workshop)
- University of California, Project Leap Architecture, dated 1998-01-04.

10 Annex B: Conformance labels (normative)

This Annex describes the requirements for human-readable and machine-readable conformance labels.

A conformance label may summarize implementation conformance statements (ICSs). Conformance labels may be used to convey ICS information via manual, semi-automated, and automated methods. The methods and techniques for associating or affixing a conforming label are outside the scope of this Standard.

10.1 Syntax for human-readable conformance labels

[EDITOR'S NOTE: TEXT TO BE SUPPLIED]

10.2 Identification of subsets

The conformance label shall indicate which subset of information types are supported using the notation "Parts *part-number-list*". Specific editions of a Part may be referenced with the notation "*part-number : edition-year*", e.g., "Parts 1, 6, 21:2002" refers to the latest editions of Part 1 and Part 6, and the 2002 edition of Part 21. If no subset is identified, then implicitly the implementation conforms to at least Parts 1, 6, 21, 22, 23, 24, 25, 26, and any other learner information Parts.

Example 1: A conforming(-only) implementation that only supports the core features, learner preference information, learner performance information, and their registries would indicate this subset with "Parts 1, 6, 24, 25", e.g., "*Strictly Conforming PAPI Learner Data Instance, Parts 1, 6, 24, 25, Using Embedded XML Coding Binding*".

Example 2: A strictly conforming implementation that supports the core features and all learner information types would have no subset indication, e.g., "*Strictly Conforming PAPI Learner Data Instance, Using Containerized XML Coding Binding*".

10.3 Identification of bindings

For "*bindings*" descriptions in conformance labels, bindings shall be listed in the order of codings, APIs, and protocols. For the codings, APIs, and protocols listed, each coding binding shall interoperate with each API binding, each API binding shall interoperate with each protocol binding, and each coding binding shall interoperate with each protocol binding.

NOTE — If all combinations of the listed codings, APIs, and protocols do not interoperate, then multiple conformance labels may be used to indicate these kind of conformance claims.

Example 1: "*Using XML and DNVP Coding and Java and JavaScript API Bindings*" implies that the implementation conformance to all four combinations: XML-Java, DNVP-Java, XML-JavaScript, DNVP-JavaScript.

Example 2: In the previous example, if the implementation did not support the XML-JavaScript combination, then two conformance labels *"Using XML and DNVP Coding and Java API Bindings"* and *"Combined DNVP Coding and JavaScript API Binding"* might describe implementation conformance claim.

10.4 Implementation varieties

The following is a summary of the possible implementation varieties used in implementation conformance statements (ICS) and their conformance labels:

- **Strictly Conforming IEEE 1484.2 PAPI Learner Data Set [, Subset]**: binding-independent; all mandatory data elements shall exist; some optional data elements may exist; extended data elements shall not exist.
- **Conforming IEEE 1484.2 PAPI Learner Data Set [, Subset]**: binding-independent; all mandatory data elements shall exist; some optional data elements may exist; some extended data elements may exist.
- **Strictly Conforming IEEE 1484.2 PAPI Learner Data Instance [, Subset], binding**: a binding shall be specified; all mandatory data elements shall exist; some optional data elements may exist; extended data elements shall not exist. Example: The file "**papi.xml**" is a *"Strictly Conforming PAPI Learner Data Instance, Using XML Coding Binding"*.
- **Conforming IEEE 1484.2 PAPI Learner Data Instance [, Subset], binding**: a binding shall be specified; all mandatory data elements shall exist; some optional data elements may exist; some extended data elements may exist. Example: The file "**papi.txt**" is a *"Conforming PAPI Learner Data Instance, Using DNVP Coding Binding"*.
- **Strictly Conforming IEEE 1484.2 PAPI Learner Data Repository [, Subset], binding(s)**: binding(s) shall be specified; shall support storing/retrieving all mandatory data element attributes, shall support storing/retrieving all optional data elements; data interchange applications shall not attempt to store/retrieve extended data elements. Example: The server XYZ is a *"Strictly Conforming PAPI Learner Data Repository, Using XML Coding and SOAP Protocol Bindings"*.
- **Conforming IEEE 1484.2 PAPI Learner Data Repository [, Subset], binding(s)**: binding(s) shall be specified; shall support storing/retrieving all mandatory data elements; shall support storing/retrieving all optional data elements; may support storing/retrieving some extended data elements. The server XYZ is a *"Conforming PAPI Learner Data Repository, Using DNVP Coding and DCTP Protocol Bindings"*.
- **Strictly Conforming IEEE 1484.2 PAPI Learner Data Reader [, Subset], binding(s)**: binding(s) shall be specified; only mandatory and optional data elements are interpreted, but no extended data elements are interpreted. Example: The import tool XYZ is a *"Strictly Conforming PAPI Learner Data Reader, Using XML Coding and Java API Bindings"*.
- **Conforming IEEE 1484.2 PAPI Learner Data Reader [, Subset], binding(s)**: binding(s) shall be specified; mandatory and optional data elements are interpreted and some extended data elements may be interpreted. Example: The import tool XYZ is a *"Conforming PAPI Learner Data Reader, Using DNVP Coding and HTTP Tunneling Protocol Bindings"*.
- **Strictly Conforming IEEE 1484.2 PAPI Learner Data Writer [, Subset], binding(s)**: binding(s) shall be specified; shall generate all mandatory data elements; may generate optional data elements; shall not generate extended data elements. Example: The export tool

XYZ is a *"Strictly Conforming PAPI Learner Data Writer, Using DNVP Coding and JavaScript API Bindings"*.

- **Conforming IEEE 1484.2 PAPI Learner Data Writer [, Subset, binding(s):** binding(s) shall be specified; shall generate all mandatory data elements; may generate optional data elements; may generate extended data elements. Example: The export tool XYZ is a *"Conforming PAPI Learner Data Writer, Using XML Coding and SOAP Protocol Bindings"*.
- **Strictly Conforming IEEE 1484.2 PAPI Learner API Environment [, Subset] binding(s):** binding(s) shall be specified; shall support all mandatory and optional data elements; extended data elements and extended services shall not be probed by applications of the API binding. Example: The software development kit XYZ is a *"Strictly Conforming PAPI Learner API Environment, Using Java Binding"*.
- **Conforming IEEE 1484.2 PAPI Learner API Environment [, Subset, binding(s):** binding(s) shall be specified; shall support all mandatory and optional data elements. Example: The software development kit XYZ is a *"Conforming PAPI Learner API Environment, Using JavaScript Binding"*.
- **Strictly Conforming IEEE 1484.2 PAPI Learner API Application [, Subset, binding(s):** binding(s) shall be specified (in order: codings, APIs, protocols); shall support all mandatory and optional data elements; extended data elements and extended services shall not be probed. Example: The application XYZ is a *"Strictly Conforming PAPI Learner API Application, Using MDAS C++ Binding"*.
- **Conforming IEEE 1484.2 PAPI Learner API Application [, Subset, binding(s):** binding(s) shall be specified; shall support all mandatory and optional data elements; extended data elements and extended services may be used to the extent permitted by data interchange participants and to the extent permitted by specifications external to the PAPI Learner Standard. Example: The application XYZ is a *"Conforming PAPI Learner API Application, Using MDAS Perl Binding"*.
- **Strictly Conforming IEEE 1484.2 PAPI Learner Communication System [, Subset, binding(s):** binding(s) shall be specified; shall support all mandatory and optional data elements; extended data elements and extended services shall not be probed by applications of the protocol binding. At least one binding shall be a protocol binding, as described by the ISO OSI communication layering model. Example: The back office gateway XYZ is a *"Strictly Conforming PAPI Learner Communication System, Using XML Coding and SOAP Session Layer Protocol Bindings"*.
- **Conforming IEEE 1484.2 PAPI Learner Communication System [, Subset, binding(s):** binding(s) shall be specified; shall support all mandatory and optional data elements; extended data elements and extended services may be used to the extent permitted by data interchange participants and to the extent permitted by specifications external to this Standard. At least one binding shall be a protocol binding, as described by the ISO OSI communication layering model. Example: The back office gateway XYZ is a *"Conforming PAPI Learner Communication System, Using DNVP Coding, C++ API Coding, and DCTP Protocol Binding"*.

The following is an example of a conformance label attached to a software application.

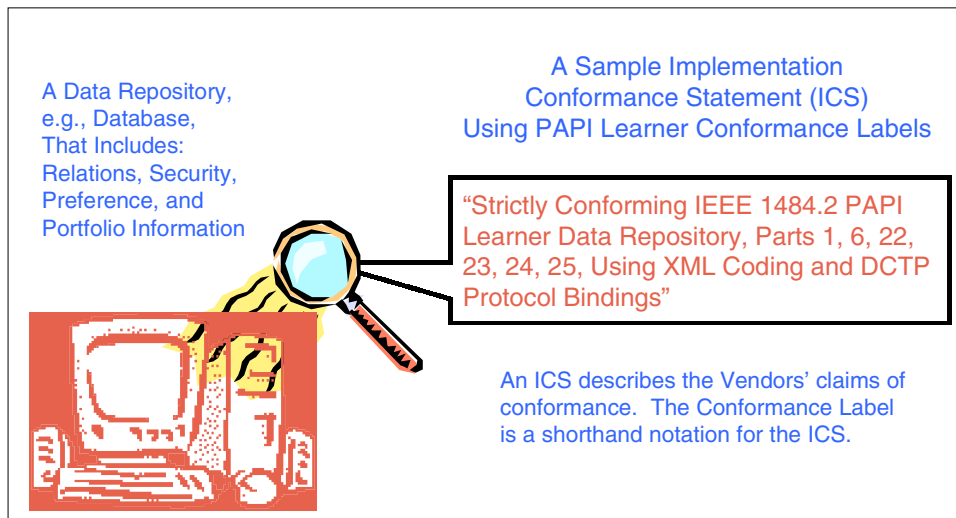


Figure 4. Using a conformance label as a shorthand for an ICS.

NOTE — An implementation may claim more than one type of conformance in its implementation conformance statement (ICS), e.g., more than one conformance label may be attached to an implementation.

10.5 Coding for automated interpretation

[EDITOR'S NOTE: TEXT TO BE SUPPLIED]

A machine-readable conformance label string may be used for automated adaptation and data interchange. The following is the syntax of the string:

IEEE-1484.2.21-2001

IEEE-1484.2.1-2001, binding:XML--encoding:UTF-8

ISO/IEC 21484-1:2001, binding: XML, encoding: UTF-8

org=ISO/IEC, doc=21484-1, edition=2001, binding=XML, encoding=UTF-8

org=ISO/IEC, doc=21484-1, edition=2001-am1-am2-tc1-tc2, binding=ASN.1

11 Annex C: ISO/IEC 11404 data model summary (informative)

This Annex is informative and not normative.

The following is a summary of foundational datatypes in ISO/IEC 11404 notation.

NOTE 1 — ISO/IEC 11404 notation is similar to many structured programming languages. Each data element is declared as the pair: "**identifier** : **datatype**". A datatype may be a native ISO/IEC 11404 datatype, a generated datatype (e.g., "**record**", "**array**"), or a created datatype (e.g., "**type X = octetstring**" creates a type "**X**", and "**Y : X**" then declares **Y** with effective datatype "**octetstring**"). Statements are separated by commas, grouping is by parentheses, and datatypes may nest.

NOTE 2 — Unless specified otherwise, all data elements are optional.

```

////////////////////////////////////
//// Foundation datatypes for other PAPI datatypes.
////////////////////////////////////

// The context label associated with a data element.
// Example: "!(japan)" might mean "not within the context
// of Japan"; implementation-defined meaning
type papi_learner_context_label_type =
    characterstring(iso-10646-1), // SPM: 500

// A shorthand for multiple array elements
type arraylist(type_spec,size) =
    array (0..size-1) of (type_spec),

// A multicultural string.
// string: The localized string.
// map: The localization (L10N) mapping, also known
// as "locale".
type locstring_type =
record
(
    string:
        characterstring(iso-10646-1), // SPM: 1000
    locale:
        characterstring(iso-646), // SPM: 255
),

// An array of multicultural strings.
// Also known as a "message cataloge" feature.
type mcstring_array_type(limit) =
    array (0..limit-1) of (locstring_type),

// A bucket for adding name-value pairs to a PAPI Learner
// information type. This feature gives some limited extension
// capability that works on all strictly conforming systems.

```

```

type papi_learner_nvp_bucket_type =
record
(
    name:
        characterstring(iso-10646-1), // SPM: 200
    value:
        octetstring, // string length SPM: 4096
),

// An internal identifier that is used to link across
// databases.
type papi_learner_identifier_type =
record
(
    context_label :
        papi_learner_context_label_type,
    identifier_kind :
        papi_learner_identifier_type_type,
    identifier_value :
        octetstring, // string length SPM: 1024
),

// An external identifier that is used to correlate
// PAPI Learner information across repositories.
type papi_learner_hid_type =
record
(
    context_label :
        papi_learner_context_label_type,
    identifier_kind :
        papi_learner_identifier_kind_type,
    identifier_value :
        octetstring, // string length SPM: 1024
),

// This datatype describes the identifier type (URL, key,
// pattern, etc.).
type papi_learner_identifier_kind_type =
    octetstring,

// This datatype describes the data certification information.
type papi_learner_data_certification_type =
record
( // note: all components are optional; all sizes are SPM
    certification_source : // who certified the record
        octetstring, // SPM: 2048
    certification_method : // how it was certified
        octetstring, // SPM: 1024
    certification_parameter_list :
        octetstring, // SPM: 16384
    certification_subset : // which data elements were certified
        octetstring, // SPM: 1024
    certification_identifier : // ID associated with cert. event
        octetstring, // SPM: 2048
    certification_bucket : // other information, SPM: 100
        papi_learner_nvp_bucket_type,
),

```

12 Annex D: XML coding binding (normative)

NOTE 1 — Because the inclusion of this Annex' requirements are dependent on an implementation conformance statement, it may be stated that this Annex is "conditionally normative".

If a PAPI Learner implementation includes the statement "PAPI Learner XML Coding Binding [*learner information subset*]" in its implementation conformance statement, then that implementation shall conform to the requirements of this Annex.

NOTE 2 — The implementation varieties are defined in Clause 4, Conformance, and summarized in subclause 4.2, Conformance Labels.

12.1 XML-related terminology

The following definitions are based on the XML 1.0 specification and adapted for use in this Annex.

[EDITOR'S NOTE: TEXT TO BE SUPPLIED]

12.2 Generating and producing XML

The following rules describe the transformation of PAPI Learner data elements, as described by this Standard and by ISO/IEC 11404 notation, to XML records.

- **Rule 1 (main transformation to XML tags):**
 - **Rule 1A:** For each data element in ISO/IEC 11404 notation, map all identifiers to XML tags, except as noted in Rule 2 below. Balanced XML tags delimit the boundary of the value associated with the data element. The nesting of the XML tags represents the structure of data elements, as described by its "aggregate datatype generator" (ISO/IEC 11404 terminology).
 - **Rule 1B:** For array and sequence aggregates, (1) an XML tag of the same name as the identifier of the aggregate represents the group of aggregates, (2) the individual data elements are represented by repeated XML tags based on the identifier of the aggregate minus the suffix "`_list`" or "`_bucket`", not the index of the element.
- **Rule 2 (special rewrite rule for certain datatypes, XML tags, and XML attributes):** Map all `locstring_type` datatypes to:
 - **Rule 2A:** The `locale` element of `locstring_type` sets the `LANG` attribute in parent XML element.
 - **Rule 2B:** The `string` element sets content of parent tag (i.e., the current target).
 Map all PAPI Learner contact information to ...[editor's note: "vCard" mapping to be added here]
- **Rule 3 (special rewrite rule for namespace conventions):** Transform the following XML tags (wildcard notation):
 - `papi_learner_*`
 to the following XML tags (wildcard notation):
 - `ieee_ltsc_papi_learner_*`

All data produced shall be well-formed XML.

Rationale

The following is a rationale for these three rules for *this specific* transformation of the PAPI Learner datatypes.

NOTE — This XML binding (PAPI Learner → XML) requires 3 transformation rules. *Other standards and different XML bindings may require more, fewer, or different transformation rules.*

Rationale for Rule 1

Rule 1 is the main transformation from ISO/IEC 11404 datatypes to XML tagging conventions. The following examples use the following definition to illustrate the transformations:

```

A: record
(
  B: integer,
  C: record
  (
    D: integer,
    E: characterstring(iso-10646-1),
  ),
  F_list: array (0..limit) of (integer),
  G: sample_mcstring_list_type,
)

```

The first sentence, "for each data element in ISO/IEC 11404 notation, map all identifiers to XML tags", transforms identifiers, e.g., "**X**:" ⇒ "<**X**>".

The second sentence, "balanced XML tags delimit the boundary of the value associated with the data element", requires that (1) the tags are balanced, and (2) the value of the data element is between the tags, e.g., "**X**: 17" ⇒ "<**X**>17</**X**>".

The third sentence, "the nesting of the XML tags represents the structure of data elements, as described by its aggregate datatype generator", requires that the nesting implied in aggregates (records, arrays, sequences/lists) results in similar nesting of the XML tags. Using the definition of **A** above, the following nesting is implied for elements **B**, **C**, **D**, and **E**:

```

<A>
  <B>...</B>
  <C>
    <D>...</D>
    <E>...</E>
  </C>
  ...
</A>

```

The fourth sentence, "for array and sequence aggregates, data elements are represented by repeated XML tags based on the identifier of the aggregate, not the index of the element", requires arrays and sequences (lists) to be represented as multiple tags with the same name — a typical XML style convention. For example, the data element **F** would be represented as:

```

<!-- correct XML binding of F_list -->
<A>

```

```

...
<F_list>
  <F>...</F>
  <F>...</F>
  <F>...</F>
</F_list>
...
</A>

```

but not as:

```

<!-- incorrect XML binding of F_list -->
<A>
...
<F_list>
  <0>...</0>
  <1>...</1>
  <2>...</2>
</F_list>
...
</A>

```

Rationale for Rule 2

PAPI Learner records use several specialized datatypes, such as multicultural datatypes for describing certain `characterstring`-type data elements that must be represented in a multicultural and multicultural context — commonly called internationalization (I18N) and localization (L10N) features. Below, is a sample version of a multicultural data type that is not intended to clash with definitions of other multicultural datatypes defined elsewhere in this Standard. In this illustration, the datatype `sample_mcstring_type` represents a single pair: a localized string and a locale specification (L10N mapping). The datatype `sample_mcstring_array_type` represents an array of these string pairs. In this example, the array `example_remarks` contains three elements, each element is a pair of strings. Presumably, an application would choose the appropriate string from use `example_remarks` based on the country (locale) that the application was operating in. The following are sample type definitions and value definitions.

```

type sample_mcstring_type =
record
(
  L10N_string: characterstring(iso-10646-1),
  L10N_locale: string_type,
),

type sample_mcstring_array_type =
array (0..limit) of (sample_mcstring_type),

value example_remarks:
sample_mcstring_array_type =
(
(
  L10N_string: "abc abc abc",
  L10N_locale: "en-US",
),
(
  L10N_string: "def def def",
  L10N_map: "fr-CA",
),
),

```

```
(
    L10N_string: "ghi ghi ghi",
    L10N_map: "de-DE",
),
),
```

Rule 2, along with array handling of Rule 1, transforms these data elements into the following XML:

```
<example_remarks LANG="en-US">abc abc abc</example_remarks>
<example_remarks LANG="fr-CA">def def def</example_remarks>
<example_remarks LANG="de-DE">ghi ghi ghi</example_remarks>
```

[Editor's Note: Rule 2, will include "vCard" mapping which transforms PAPI Learner contact information to/from "vCard" structures.]

Rationale for Rule 3

This rule is used for rewriting tags to use certain namespace conventions. This rule could have specified XML namespaces by choosing a different namespace convention (prefixes).

After Rule 3, the implementation is required to assure that the result of these transformations is well-formed XML.

12.3 Consuming and interpreting XML

The following rules describe the transformation of XML records to PAPI Learner data elements, as described by this Standard and by ISO/IEC 11404 notation.

All data consumed shall be well-formed XML.

- **Rule 1 (special rewrite rule for namespace conventions):** Transform the following:
 - **Rule 1A:** XML tags that match (in wildcard notation):


```
ieee_ltsc_papi_learner_*
```

 to the following XML tags (in wildcard notation):


```
papi_learner_*
```
- **Rule 2 (special rewrite rule for certain datatypes, XML tags, and XML attributes):** Transform according to the following:
 - **Rule 2A:** The `LANG` attribute of the XML element sets the `locale` element of the corresponding `locstring_type` data element.
 - **Rule 2B:** The contents of the tagged element sets the `string` element of the corresponding `locstring_type` data element.
- **Rule 3 (special rule for certain datatypes):** For each XML tag, that is associated with an identifier defined by a PAPI Learner data element in this Standard, its corresponding opening and closing balanced XML tags are matched. For each XML tag, except as modified in Rule 2 above, map each XML tag to the corresponding data element identifier. The nesting of the XML tags represents the nesting of the data elements, i.e., the reverse of the operation in Rule #1 of subclause 13.2, Generating and Interpreting XML, above. The contents of each tagged element is converted to the value of the corresponding data element.

Rationale

Rationale for Rule 1

Before processing, the implementation is assured that it is consuming and interpreting well-formed XML.

This rule strips the XML namespace prefixes and suffixes as necessary. In this illustration, XML namespaces were not used, but a namespace prefix ("`ieee_1tsc`") was used to reduce the possibility of namespace collisions.

Rationale for Rule 2

This rule does the reverse mapping from the `LANG` attribute to the `locstring_type` datatype. This rule is careful to transform only known `locstring_type` data elements because all other XML `LANG` attributes do not correspond to `locstring_type` data elements in this Standard.

[Editor's Note: In the next draft, Rule 2, will include "vCard" mapping which transforms PAPI Learner contact information to/from "vCard" structures.]

Rationale for Rule 3

This rule handles the main transformation of XML tags and their contents to data elements.

The first sentence, "for each XML tag, that is associated with an identifier defined by a PAPI Learner data element in this Standard, its corresponding opening and closing balanced XML tags are matched", (1) ignores all identifiers that are unknown to this Standard, and (2) properly pairs them.

The second sentence, "for each XML tag, except as modified in Rule 2 above, map each XML tag to the corresponding data element identifier", creates the association with data elements, but does not assign the values of the data elements..

The third sentence, "the nesting of the XML tags represents the nesting of the data elements, i.e., the reverse of the operation in Rule #1 of subclause 19.1", assures that the internal structure of the XML tags, to the extent required by this Standard, agree with the internal structure of the data elements.

The fourth sentence, "the contents of each tagged element is converted to the value of the corresponding data element", transforms the contents within the XML tags to values of the data elements, i.e., it "populates" the data elements.

12.4 Representation of basic data types

The following subclauses describe the transformation of data element values to/from character representations for information interchange for use within an XML binding.

12.4.1 Characters and character strings

Data elements that are of type `character` shall be represented as per the XML specification.

NOTE 1 — Special characters, such as "&" "<" ">" ";" require translation methods and, possibly, lossless translation.

NOTE 2 — Some encodings, such as ISO-8859-1 and UTF-8 permit the direct encoding of the representation of characters such as "©" (the copyright symbol). Other encodings, such as ASCII, require encoding extensions, such as "©", to represent these symbols.

12.4.2 Integers

Data elements that are of type **integer** shall be represented as per ISO/IEC 9899:1999, C Programming Language, subclause 6.4.4.1, Integer Constants; excluding "U", "L", and "LL" suffixes and their lowercase variants; and may include an optional leading sign, either plus ("+") or minus ("-"), but not both.

Examples:

```
0          // zero
23         // twenty-three
0x17      // same in hexadecimal
027       // same in octal
-34       // negative thirty-four
+34       // positive thirty-four
+34       // same
```

12.4.3 Real numbers

Data elements that are of type **real**:

- if integral, may be represented integers, as specified above in subclause 19.5, Integers;
- if not integral or not represented as integers, shall be represented as per ISO/IEC 9899:1999, C Programming Language, subclause 6.4.4.2, Floating Constants; excluding "F" and "L" suffixes and their lowercase variants; and may include an optional leading sign, either plus ("+") or minus ("-"), but not both.

Examples:

```
0          // zero
0.0        // same
130.0      // one hundred thirty
1.3E2     // same
+1.3E2    // same
```

12.4.4 Date and time values

[EDITOR'S NOTE: TEXT TO BE SUPPLIED]

Data elements of type **time** shall be represented as per ISO 8601, Data elements and interchange formats — Information interchange — Representation of dates and times.

NOTE 1 — ISO 8601 is intended to represent dates and times between 1 January 0001 and 31 December 9999 using the Gregorian calendar. It is well understood that there are anomalies with this approach, e.g., the month of September 1752 has less than 30 days, the lack of correlation of time-

zones prior to the introduction of transcontinental railroads, changing the beginning month of the calendar, the inability to represent dates Before the Common Era, and the inability to represent dates after 31 December 9999.

ISO 8601 is applied as follows:

- Only the extended format shall be used. Example: **"1999-01-02"** and **"03:04:05"** are valid, but **"19990102"** and **"030405"** are not. Rationale: Compatibility with ANSI standards and XML Schema specifications.
- Decimal fractions shall use the full stop character ("."), also known as the period character. Example: **"1999-01-02T03:04:05.1"** represents 03:04:05.1 AM on 2 January 1999. Note: ISO 8601 permits the use of the comma (",") and full stop characters, and specifies that comma is the preferred sign. This Standard only permits the full stop (period) character. Rationale: (1) The comma character may interfere or conflict with other lexical, embedded, or surrounding information processing environments. (2) The full stop character is compatible with the XML Schema specifications.

For points in time, ISO 8601 is applied as follows:

- Dates shall be represented by calendar date format only. Note: Calendar dates are represented by year, month, and date. Other formats permitted by ISO 8601, but prohibited in this Standard include ordinal date (i.e., the date is represented by three decimal digits, e.g., **"1985032"** is 1 February 1985) and calendar week and day number (i.e., the date is represented by the week number and the day of the week, e.g., Sunday, 1 January 1995 is **"1994W527"**, and Tuesday, 31 December 1996 is **"1997W012"**). Rationale: The additional processing for these other date formats may be more error prone and can reduce interoperability.
- Dates and times shall be represented as an ISO 8601 complete representation and shall not use the **"T"** time indicator. Example: **"19990102030405"** represents 03:04:05 AM on 2 January 1999. Rationale: The **"T"** time indicator is omitted because it is not necessary; ISO 8601 permits this omission when there is no ambiguity. The ambiguity is avoided by (1) permitting only ISO 8601 basic formats, and (2) requiring the identification of missing date and time components.
- The underscore character ("_") shall be used to indicate missing components. One underscore character shall be used for each digit that is missing from the integer portion of the date or time. Example: **"__0102"** represents 2 January of the current year. Note: Data that has missing components (e.g., the century) can cause significant interoperability problems, such as the "Year 2000 Problem". It is recommended that implementations avoid generating data with missing components, but it may not be possible to resolve all circumstances. Rationale: ISO 8601 uses hyphens for missing components, but hyphens may interfere or conflict with other lexical, embedded, or surrounding information processing environments. ISO 8601 uses the hyphen to indicate that one component is missing (e.g., **"-0102"**), while this Standard uses one underscore character per digit (e.g., **"__0102"**), which is less error prone for information processing.
- Times shall be represented in local time or Coordinated Universal Time (UTC). Times represented in local time shall use no suffix and shall not indicate the difference between local time and UTC. Times represented in UTC shall use the **"Z"** suffix, as per ISO 8601. Example: **"19990102030405"** local time in New York City (5 hours west of UTC) is the same time as **"19990102030905Z"**. Rationale: The use of time zone information, other than

UTC, causes additional processing for data interpretation, which may be more error prone. The prohibition of time differences (e.g., "19990102030405+0700") eliminates the use of the plus ("+") and minus ("-") characters which may interfere or conflict with other lexical, embedded, or surrounding information processing environments.

For durations of time, ISO 8601 is applied as follows:

- The time duration shall begin with the prefix "P", as per ISO 8601. Example: "P2Y" means two years. Note: Not all date and time components are required for a duration of time.
- The time duration shall use the case-sensitive designators: "Y" for years, "M" for months, "D" for days, "W" for weeks, "h" for hours, "m" for minutes, and "s" for seconds. Example: "P2Y10M25h2m5s" represents the duration 2 years, 10 months, 25 hours, 2 minutes, and 5 seconds. For durations of time, a date and time component may exceed the maximum number of units in a corresponding component of a point in time, e.g., months may exceed 12, hours may exceed 24, and minutes may exceed 60. Rationale: The use of case-sensitive designators simplifies the processing of date and time information.

NOTE 2 — The characters described in ISO 8601 only specify the names of characters, not their encodings. This is emphasized by ISO 8601, subclause 4.4, Characters used in the representations:

The representations specified in this International Standard use digits, alphabetic characters, and special characters specified in ISO 646. *** Note 2: Encoding of characters for the interchange of dates and times is not in the scope of this Standard.

12.4.5 Void types

A void type shall have no representation and shall have no encoding.

Example: The following record

```
A: record
(
  B: integer,
  C: void,
  D: characterstring(iso-10646-1),
)
```

is represented in XML as:

```
<!-- correct XML representation of C -->
<A>
  <B>17</B>
  <D>hello</D>
</A>
```

but not as:

```
<!-- incorrect XML representation of C -->
<A>
  <B>17</B>
  <C></C>
  <D>hello</D>
</A>
```

12.5 Encoding of character representations

The encoding of character representations to octet values is specified by the XML encoding technique.

Conforming PAPI Learner data instances with XML coding binding shall be encoded in one of: ASCII, ISO/IEC 8859-1, ISO/IEC 10646-1 UTF-8, or ISO/IEC 10646-1 UTF-16.

Conforming PAPI Learner applications with XML coding binding shall support all of the following encodings: ASCII, ISO/IEC 8859-1, ISO/IEC 10646-1 UTF-8, and ISO/IEC 10646-1 UTF-16.

12.6 Handling exceptions and extensions

12.6.1 Implementation-defined behavior

The following are implementation-defined behaviors in addition to those described elsewhere in this Standard.

The following are implementation-defined behaviors in the production and consumption of XML codings:

- The maximum size, in octets, of a strictly conforming PAPI Learner data instance, as coded in XML, that may be processed successfully.
- The maximum nesting depth of XML records.
- The time zone information for data elements of `time` type that have unspecified timezones.

12.6.2 Unspecified behavior

The following are unspecified behaviors in addition to those described elsewhere in this Standard.

The following is unspecified behavior in the generation or interpretation of XML codings:

- The order of processing data elements.

The following is unspecified behavior in the production or consumption of XML codings:

- The use of additional whitespace characters outside those of the data element value and those required by the XML specification.

12.6.3 Undefined behavior

The following are undefined behaviors in addition to those described elsewhere in this Standard.

The following are undefined behaviors in the production or consumption of XML codings:

- The use of XML tags that correspond to extended data elements.
- The use of XML tags that correspond to reserved data elements.
- The use of XML tags or attributes not specified in this XML coding binding.
- The use of characters outside the repertoire described in this Standard.

13 Annex E: DNVP coding binding (normative)

NOTE 1 — Because the inclusion of this Annex' requirements are dependent on an implementation conformance statement, it may be stated that this Annex is "conditionally normative".

If a PAPI Learner implementation includes a "PAPI Learner DNVP Coding Binding [*learner information subset*]" in its implementation conformance statement, then that implementation shall conform to the requirements of this Annex.

NOTE 2 — The implementation varieties are defined in Clause 4, Conformance, and summarized in subclause 4.2, Conformance Labels.

NOTE 3 — The Dotted Name-Value Pair (DNVP) notation is based on an RFC 822 style of messaging. RFC 822, Standard for the Format of ARPA Internet Text Messages, describes text messages that are commonly referred to as internet E-mail messages. In this style of messaging, the beginning of a message contains a header with header elements. For example,

```
From: sender@host.com
To: user@host.com
Subject: a subject line
```

might represent three header elements — a portion of a header. This kind of messaging is described in RFC 822, subclause 3.1, General Description. Additionally, RFC 2068, Hypertext Transfer Protocol — HTTP/1.1, subclause 4.2, Message Headers, itself is harmonized with RFC 822 (from RFC 2068, subclause 4.2: "HTTP header fields ... follow the same generic format as that given in Section 3.1 of RFC 822").

This style of binding (RFC-822-like) is in common use in many interchange environments, such as E-mail systems and web servers.

The following is an example of this generalized format:

```
name_1: value_1
name_2: value_2
name_3: value_3
```

13.1 Dotted Name-Value Pairs (DNVPs)

NOTE — The following wording was extracted, excerpted and adapted from, and harmonized with RFC 2068 (HTTP/1.1).

13.1.1 Basic lexical elements

A newline character (CRLF) is defined by its encoding.

NOTE 1 — A newline character might be line feed (e.g., Unix/Linux), carriage return (e.g., Macintosh), or carriage return line feed combination (e.g., Windows).

For maximum interoperability, implementations should use the carriage return line feed combination for the newline character when producing data. Implementations that use only one character, either

line feed or carriage return, for the newline character should ignore the other character, carriage return or line feed, respectively, when consuming data.

Leading whitespace (LWS) is defined as at least one or more space or tab characters that follow a newline.

NOTE 2 — LWS does not include the prior newline. A sequence of space or tab characters without a prior newline is not LWS.

A control character (CTL) is a ISO 646 IVR control character in the range 0-31 (decimal) or the control character 127 (decimal) but not the control character 9 decimal (HT — horizontal tab).

The 14977 syntax has been extended with the following:

A meta-identifier (terminal symbol) may include the underscore character (in addition to letters and digits).

The sequence `\Uhhhhhhhh`, where `h` is a hexadecimal digit, identifies a UCS character. The sequence `\uhhhh` identifies a UCS character in the Basic Multilingual Plane. NOTE — The UCS representation of C0 and C1 control characters is harmonized with the UCS character representation, e.g., a Form Feed character would be represented as `\U0000000C`, `\U0000000c`, `\u000C`, or `\u000c`.

A text characters is any character except control characters.

x

```
CTRL = (* all ctrl characters except tab *)
    "\u0000" | "\u0001" | "\u0002" | "\u0003" |
    "\u0004" | "\u0005" | "\u0006" | "\u0007" |
    "\u0008" | "\u000A" | "\u000B" | "\u000C" |
    "\u000D" | "\u000E" | "\u000F" | "\u0010" |
    "\u0011" | "\u0012" | "\u0013" | "\u0014" |
    "\u0015" | "\u0016" | "\u0017" | "\u0018" |
    "\u0019" | "\u001A" | "\u001B" | "\u001C" |
    "\u001D" | "\u001E" | "\u001F" | "\u007F" ;
```

x

```
LPAREN = "(" ; (* left parenthesis *)
RPAREN = ")" ; (* right parenthesis *)
LTSIGN = "<" ; (* less-than sign *)
GTSIGN = ">" ; (* greater-than sign *)
ATSIGN = "@" ; (* at sign *)
COMMA = "," ; (* comma *)
SEMI = ";" ; (* semicolon *)
COLON = ":" ; (* colon *)
BSLASH = "\u005c" ; (* backslash *)
DQUOTE = "\u0022" ; (* double quote *)
SLASH = "/" ; (* forward slash *)
LSQBR = "[" ; (* left square brace *)
RSQBR = "]" ; (* right square brace *)
QMARK = "?" ; (* question mark *)
EQSIGN = "=" ; (* equals sign *)
LBRACE = "{" ; (* left curly brace *)
RBRACE = "}" ; (* right curly brace *)
TSPECIALS =
LPAREN | RPAREN | LTSIGN | GTSIGN | ATSIGN |
```

COMMA		SEMI		COLON		BSLASH		DQUOTE	
SLASH		LSQBR		RSQBR		QMARK		EQSIGN	
LBRACE		RBRACE		SPACE		HTAB		;	

x

A text characters is any character except control characters.

The following are token special characters:

()	<	>	@
,	;	:	\	"
/	[]	?	=
{	}	SP	HT	

A token is one or more characters that exclude control characters or token special characters.

A string of text is consumed as a single token (and token special characters are not special) if it is quoted using double-quote marks.

Example 1: The characters

```
"abcd - ( ) , : [ ] "
```

are consumed as a single token.

The backslash character ("\") may be used for single-character quoting, i.e., removing the special nature of a token special character, one character at a time.

Example 2: The characters

```
they're
```

are consumed as a single token and the single quote character (') has no special meaning.

Example 3: The characters

```
"say \"hello\""
```

are consumed as a single token, the token includes the double quote characters before and after the work hello, and the quote characters that surround the word hello have no special meaning.

13.1.2 Field name and field value

A Name-Value Pair (NVP) shall consist of a field name, followed by a colon (":"), followed by its field value. A field name is a token. Field names shall be case-sensitive.

NOTE — The case sensitivity of this DNVP binding differs from RFC 822.

A field value may be preceded by any amount of leading white space (LWS). Conforming implementations should use a single space (SP) as LWS. A field value is zero more characters consisting of combinations of tokens and/or token special characters.

A field value of zero characters represents an empty value being associated with the field name.

NOTE — A Name-Value Pair starts at the beginning of a line.

13.1.3 Newline processing

A Name-Value Pair may be extended over multiple lines by preceding each extra line with at least one space (SP) or horizontal tab (HT). All linear white space, including folding, has the same semantics as a single SP character.

A newline (CRLF) shall follow a completed Name-Value Pair.

13.1.4 Syntax summary

This subclause is informative and not normative.

NOTE — The following BNF syntax summary is extracted, excerpted and adapted from, and harmonized with RFC 2068 (HTTP/1.1):

```

CRLF      = <newline character>
CTL       = <any US-ASCII control character
           (characters 0 - 31) and DEL (127), except HT (9)>
LWS       = [CRLF] 1*( SP | HT )
token     = 1*<any character except CTLs or tspecials>
tspecials = "\" | "(" | ")" | "<" | ">" | "@"
           | "," | ";" | ":" | "\" | "<" | ">"
           | "/" | "[" | "]" | "?" | "="
           | "{" | "}" | SP | HT
quoted-pair = "\" character
quoted-string = <">*<any char except non-quoted
                double quotes><">
message-header = field-name ":" [ field-value ] CRLF
field-name    = token
field-value   = *( field-content | LWS )
field-content = <the characters making up the field-value
                and consisting of combinations of
                tokens or tspecials>

```

13.2 Generating and producing dotted name-value pairs

The following rules describe the transformation of PAPI Learner data elements, as described by this Standard and by ISO/IEC 11404 notation, to DNVP notation.

- **Rule 1 (main transformation to DNVPs):** For each data element, map all identifiers to fully-qualified field names. A fully-qualified name represents the nested structure of the data element, as described by its "aggregate datatype generator" (ISO/IEC 11404 terminology). A period (".") shall separate each level of nesting in a fully-qualified field name. For array and sequence aggregates, (1) the individual data elements are represented by repeated DNVP field names based on the identifier of the aggregate minus the suffix "**_list**" or "**_bucket**", not the index of the element, (2) each element of the array or sequence aggregate shall be bracketed by two null-value DNVPs, one at the beginning of each element with the field name suffix "**.__begin**" and one at the end of each element with the field name suffix

"**__end**". A field name is followed by a colon (":"), followed by the value of its associated data element.

- **Rule 2 (special rewrite rule for namespace conventions):** Transform the following DNVP field names (wildcard notation):

`papi_learner_*`

to the following DNVP field names (wildcard notation):

`ieee_ltsc_papi_learner_*`

Rationale

The following is a rationale for these two rules for *this specific* transformation of the PAPI Learner datatypes.

NOTE — This RFC 822-like binding (PAPI Learner → DNVPs) requires 2 transformation rules. *Other standards and different DNVP bindings may require more, fewer, or different transformation rules.*

Rationale for Rule 1

Rule 1 is the main transformation from ISO/IEC 11404 datatypes to DNVP notation. The following examples use the following definition to illustrate the transformations:

```
A: record
(
  B: integer,
  C: record
  (
    D: integer,
    E: characterstring(iso-10646-1),
  ),
  F_list: array (0..limit) of (integer),
)
```

The first three sentences, "For each data element, map all identifiers to fully-qualified field names. A fully-qualified name represents the nested structure of the data element, as described by its "aggregate datatype generator" (ISO/IEC 11404 terminology). A period (".") shall separate each level of nesting in a fully-qualified field name", transform identifiers in to field names, such as:

```
A.B
A.C.D
A.C.E
```

The fourth sentence, "for array and sequence aggregates, elements are represented by repeated DNVP field names based on the identifier of the aggregate, not the index of the element", requires arrays and sequences (lists) to be represented as multiple DNVPs with the same name — permitted in RFC 822-like systems, but with a slightly different meaning (RFC 822 allows these lines to be combed, while this PAPI Learner coding binding does not permit automatic consolidation of DNVPs). For example, the data element **F** would be represented as:

```
(correct DNVP binding of F)
A.F.__begin:
A.F: xxx
A.F.__end:
A.F.__begin:
A.F: yyy
```

```
A.F.__end:
A.F.__begin:
A.F: zzz
A.F.__end:
```

but not as:

```
(incorrect DNVP binding of F)
A.F.0: xxx
A.F.1: yyy
A.F.2: zzz
```

The fifth sentence, "A field name is followed by a colon (":"), followed by the value of its associated data element", associates the field name with its value and separates the two with a colon (":").

Example:

```
A.B: 17
A.C.D: 34
A.C.E: yellow pigs
A.F: 51
A.F: 68
A.F: 85
```

Rationale for Rule 2

This rule is used for rewriting tags to use certain namespace conventions.

13.3 Consuming and interpreting dotted name-value pairs

The following rules describe the transformation of DNVPs to PAPI Learner data elements, as described by this Standard and by ISO/IEC 11404 notation.

- **Rule 1:** Transform the following XML tags (wildcard notation):
`ieee_ltsc_papi_learner_*`
to the following XML tags (wildcard notation):
`papi_learner_*`
- **Rule 2:** For each field name that is associated with an identifier defined by a PAPI Learner data element in this Standard, map each field name to the corresponding data element identifier. The nesting of the field names represents the nesting of the data elements, i.e., the reverse of the operation in Rule #1 of subclause 14.2, Generating and Interpreting Dotted Name-Value Pairs, above. Each field value is converted to the value of the corresponding data element. An empty field value of an aggregate may represent the existence of the aggregate, but not the value of its components.

Rationale

Rationale for Rule 1

This rule transforms any namespace prefixes, as necessary. In this illustration, the namespace prefix ("`ieee_ltsc_`") was used to reduce the possibility of namespace collisions.

Rationale for Rule 2

This rule handles the main transformation of DNVPs to data elements.

The first sentence, "for each field name that is associated with an identifier defined by a PAPI Learner data element in this Standard, map each field name to the corresponding data element identifier", (1) ignores all identifiers that are unknown to this Standard, (2) properly pairs the field names and identifiers of data elements, and (3) creates the association with data elements, but does not assign the values of the data elements..

The second sentence, "The nesting of the field names represents the nesting of the data elements, i.e., the reverse of the operation in Rule #1 of subclause 20.2, Generating and Interpreting Dotted Name-Value Pairs, above", assures that the internal structure of the DNVPs, to the extent required by this Standard, agree with the internal structure of the data elements.

The third sentence, "each field value is converted to the value of the corresponding data element", transforms the field values of the data elements, i.e., it "populates" the data elements.

The fourth sentence, "an empty field value of an aggregate may represent the existence of the aggregate, but not the value of its components", merely creates the aggregate.

In the fragment below, **A.C**, which has an empty value, may be used to indicate the existence of an aggregate:

```
A.B: 17
A.C:
A.C.D: 34
A.C.E: yellow pigs
```

An empty value is a useful technique when aggregates are comprised of optional data elements, i.e., the signaling of the existence of the aggregate, but the lack of aggregate components:

```
A.B: 17
A.C:
```

An empty value is not used to indicate a `void` type.

13.4 Representation of basic data types

The following subclauses describe the transformation of data element values to/from character representations for information interchange for use within the DNVP binding.

13.4.1 Characters and character strings

Data elements that are of type `character` shall be represented only as per ISO/IEC 8859-1 when encoded according to the rules of RFC 1522 .

13.4.2 Integers

Data elements that are of type `integer` shall be represented as per subclause 19.3.2 above.

13.4.3 Real numbers

Data elements that are of type `real` shall be represented as per subclause 19.3.3 above.

13.4.4 Date and time values

Data elements that are of type `time` shall be represented as per subclause 19.3.4 above.

13.4.5 Void types

A void type shall have no representation and shall have no encoding.

Example: The following record

```
A: record
(
  B: integer,
  C: void,
  D: characterstring(iso-10646-1),
)
```

is represented in an RFC 822-like binding as:

```
(correct DNVP binding of C)
A.B: 17
A.D: hello
```

but not as:

```
(incorrect DNVP binding of C)
A.B: 17
A.C:
A.D: hello
```

13.5 Encoding of character representations

Conforming PAPI Learner DNVP coding bindings shall encode character representations to character values specified by ISO/IEC 8859-1. Characters outside this repertoire shall be encoded according to the rules of RFC 1522.

13.6 Handling exceptions and extensions

13.6.1 Implementation-defined behavior

The following are implementation-defined behaviors in addition to those described elsewhere in this Standard.

The following are implementation-defined behaviors in the production and consumption of DNVP codings:

- The encoding, in characters, of the newline character. NOTE — Implementations can avoid implementation-defined behavior by using the carriage return line feed combination characters for the newline character. See subclause 13.1.1, Basic Lexical Elements, above.
- The maximum size, in characters, of a strictly conforming PAPI Learner data instance, coded as a DNVP, that may be processed successfully.
- The time zone information for data elements of `time` type that have unspecified timezones.

13.6.2 Unspecified behavior

The following are unspecified behaviors in addition to those described elsewhere in this Standard.

The following is unspecified behaviors in the generation or interpretation of DNVPs:

- The order of processing data elements.

13.6.3 Undefined behavior

The following are undefined behaviors in addition to those described elsewhere in this Standard.

The following are undefined behaviors in the production or consumption of DNVPs:

- The use of field names that correspond to extended data elements.
- The use of field names that correspond to reserved data elements.
- The use of field names not specified in this DNVP coding binding.
- The use of characters outside the repertoire described in this Standard.

14 Annex F: ASN.1 coding binding (normative)

[EDITOR'S NOTE: TEXT TO BE SUPPLIED]

15 Annex G: MDAS API binding (normative)

[EDITOR'S NOTE: TEXT TO BE SUPPLIED]

16 Annex H: Document development (informative)

This Annex is informative and not normative.

NOTE — This Annex will be removed prior to publishing of this Standard.

16.1 Revision history

- **Draft 1, 1997-03-27**, the first draft. This draft was just the schema of the personal information (at the time, known as "Personal Information Management Systems (PIMS)").
- **Draft 2, 1997-10-07**, the second draft. This draft was largely complete and submitted as a base document for IEEE 1484.2.
- **Draft 3, 1998-04-02**, the third draft. Revised wording to incorporate comments from standards activity. Harmonized work with IMS specifications. Incorporated schema changes from prototyped implementations.
- **Draft 4, 1998-06-08**, the fourth draft. Added TCP and Java server bindings.
- **Draft 5, 1998-11-26**, the fifth draft. Added functionality and conceptual model.
- **Draft 6, 2000-06-23**, the sixth draft. Added semantics and bindings sections. Converted to IEEE format.
- **Draft 7, 2000-11-28**, the seventh draft. Added examples and bindings. Split document into several pieces.
- **Draft 8, 2001-11-25**, the current draft. Split information types into several documents.

16.2 Release notes for this document

The following notes apply to this release of this document:

- The glossary needs harmonization with the IEEE 1484.3 work.
- The codings and API work will be harmonized with IEEE 1484.14 and other activities.
- The protocols will be harmonized with IEEE 1484.15 and other activities.
- PARs need to be written for other parts of the document.

16.3 Resolved issues

The following issues have been resolved:

- Multiple syntax bindings. After much lively discussion in IEEE 1484.2, there is agreement that multiple syntax bindings are necessary.
- Functionality and conceptual model. This work has benefited from much discussion in IMS.
- Conceptual model and conformance wording has been defined.
- Informative material has been moved to the end of the document.
- Significantly reduced size of document.

16.4 Open issues

The following issues are outstanding:

- Interoperability negotiation. When two systems interoperate, they need to determine and resolve incompatibilities in their data models.
- Completing remaining coding, API, and protocol definitions.
- Agreement on semantic codings. Need to have naming conventions for identifying semantic coding conventions, e.g., creating the name "US-NY-LETTER-GRADE". Need to consider mechanisms for translating from coding to another.
- Other syntax bindings. What other syntax bindings are necessary? What tools are available to convert syntax bindings and how are they invoked?

16.5 Comments on this document

All comments are appreciated. Please return all comments on this release of this document by **Friday, 2002-02-22 23:00 UTC**. The Technical Editor may be contacted at the following:

Frank Farance, Farance Inc. (Edutool division)
Island Box 256, New York, NY, 10044-0205, USA
Tel: +1 212 486 4700, Fax: +1 212 759 1605, E-mail: frank@farance.com