

IEEE 1484.14.3/D1, 2001-02-06

Draft Guidelines for Learning Technology — Rule-Based Dotted Name-Value Pair (DNVP) Binding Techniques

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 is an unapproved draft of a proposed IEEE Standard, subject to change. Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities. If this document is to be submitted to ISO or IEC, notification shall be given to the IEEE Copyright Administrator. Permission is also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position. Other entities seeking permission to reproduce this document for standardization or other activities, or to reproduce portions of this document for these or other uses, must contact the IEEE Standards Department for the appropriate license. Use of information contained in this unapproved draft is at your own risk.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA

[Note: Information about IEEE LTSC P1484.14 can be found at:

<http://ieee.ltsc.org/wg14>

This document is also available at:

<http://edutool.com/sxb>

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

Introduction

(This introduction is not part of IEEE P1484.14.3, Rule-Based Dotted Name-Value Pair (DNVP) Binding Techniques.)

** TO BE SUPPLIED **

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

Bruce Peoples, *Chair*

Frank Farance, *Technical Editor*

aaa

zzz

To be supplied

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

CONTENTS

1 Overview	6
1.1 Scope.....	6
1.2 Purpose.....	6
2 Normative references	7
3 Definitions	8
3.1 Definitions incorporated via normative reference	8
3.2 aggregate (datatype, value)	8
3.3 binding	8
3.4 coding.....	9
3.5 conditional data element	9
3.6 consume (data).....	9
3.7 data instance.....	11
3.8 data object	11
3.9 data set.....	11
3.10 data structure.....	11
3.11 encoding.....	11
3.12 extended data element	11
3.13 generate (data)	12
3.14 implementation behavior	12
3.15 implementation-defined behavior/value.....	12
3.16 implementation value.....	12
3.17 locale-specific behavior.....	13
3.18 longevity (data element).....	13
3.19 mandatory data element	14
3.20 nomadic (access, system).....	14
3.21 obligation (data element).....	14
3.22 obsolete data element	14
3.23 optional data element	15
3.24 produce (data)	15
3.25 repository	15
3.26 reserved data element	15
3.27 undefined behavior/value.....	15
3.28 unspecified behavior/value	16
3.29 Acronyms and abbreviations	16
4 Standards wording template	17
4.1 Dotted Name-Value Pairs (DNVPs)	17
4.1.1 Basic lexical elements.....	17
4.1.2 Field name and field value.....	19
4.1.3 Newline processing.....	19
4.1.4 Syntax summary.....	19
4.2 Generating and producing dotted name-value pairs	20
4.3 Consuming and interpreting dotted name-value pairs	21
4.4 Representation of basic data types.....	22
4.4.1 Characters and character strings.....	23
4.4.2 Integers	23
4.4.3 Real numbers.....	23
4.4.4 Date and time values.....	23

4.4.5 Void types.....	23
4.5 Encoding of character representations.....	23
4.6 Handling exceptions and extensions	24
4.6.1 Implementation-defined behavior	24
4.6.2 Unspecified behavior	24
4.6.3 Undefined behavior.....	24
4.7 Sample DNVP records derived from the binding.....	24
5 Annex A: Document development.....	25
5.1 Revision history	25
5.2 Release notes for this document	25
5.3 Resolved issues.....	25
5.4 Open issues	25
5.5 Comments on this document	25

1 Overview

1.1 Scope

This Guideline describes techniques for rule-based dotted name-value pair bindings for data models. The wording in this Guideline may be incorporated into standards to support these kind of bindings. This wording describes a rule-based approach for describing these standards words.

1.2 Purpose

The purpose of this Guideline is to provide common standard wording for commonly used bindings. A name-value pair is common information coding technique:

```
[Name]    [Value]
priority: high
```

A dotted name may be used to represent more complex naming and information structures:

```
[Name]                                [Value]
preferences.mail.priority: high
```

Because the this Guideline uses a rule-based approach, it can be adopted to many standards.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this International Standard. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. IEEE and members of ISO and IEC maintain registers of currently valid Standards.

- IEEE 1484.3, Learning Technology Glossary.
- RFC 822
- IETF RFC 2068, Hypertext Transfer Protocol (HTTP/1.1)
- W3C XML, Extensible Markup Language (XML)
<http://www.w3.org/TR/REC-xml>
- ISO/IEC 11404:1996, Language Independent Datatypes.
- ANSI X3.30:1998, "Representation of Calendar Date and Ordinal Date for Information Interchange"
- ANSI X3.42:1990, "Representation of Numeric Values in Character Strings for Information Interchange"
- ANSI X3.285:1998, "Metamodel for Data Representation"
- ISO/IEC 11179
- ISO/IEC 8601

3 Definitions

Note 1: The following definitions are being/have been harmonized with the IEEE 1484.3 Glossary and Reference Materials.

Note 2: Portions of this Clause would be included in the standard that is being created from the template wording in Clause 5.

3.1 Definitions incorporated via normative reference

Note: The following terms and their definitions have been incorporated via the normative references:

- ISO/IEC 2382 "Information Technology — Vocabulary" (multiple parts)
- ANSI "American National Standard Dictionary for Information Technology (ANSDIT)"
- IEEE 1484.3 Glossary and Reference Materials

3.2 aggregate (datatype, value)

A generated datatype or value, each of whose datatypes or values is, in principle, made up of the component datatypes or values. The datatype or value is generated by applying an algorithmic procedure to combine the component datatypes or values. The component values are accessible via characterizing operations. The properties of the aggregate are independent of the properties of its components.

Example 1: An array aggregate contains components *all of the same type*. A characterizing operation uses an index (a number) to access individual components.

```
my_array:
    array (0..9) of (integer), // an array of integers
my_array(4) // accessing element #4
```

Example 2: A record aggregate contains components, each component *individually typed and labeled*. A characterizing operation uses a element *name* (an *identifier* — a word) to access individual components.

```
A: record
(
    B: integer,
    C: void,
    D: characterstring(iso-10646-1),
),
A.B // accessing element labeled B
```

Note: This definition is adapted from ISO/IEC 11404.

3.3 binding

An application or mapping from one framework or specification to another.

3.4 coding

(1) In information interchange, a formalized or structured representation of information. See Also: encoding.

(2) A process of representing information in some structure.

3.5 conditional data element

Within the appropriate context, an element of a data structure that is defined and required within an instance of the data structure, if certain conditions are satisfied.

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

See Also: extended data element; mandatory data element; obligation (data element); optional data element.

3.6 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"
- 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?
- might limit the depth of its analysis: "R" is only explored one level deep to discover tags "D" and "E", but only a limited analysis (e.g., finding balanced tags) of the contents of "E" is performed such that tags "F" and "G" are not analyzed or discovered.

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: the header "std_data.h",
//// and a strictly conforming application that includes
//// 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 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.

3.7 data instance

A data set rendered in some binding.

3.8 data object

A unit of data processing within the conceptual model of accessing implementations' data.

Note 1: A data object may be a data element or an implementation-defined object. Strictly conforming implementations only use or access data objects that are data elements.

Note 2: The behavior of a data object, further defined and constrained in the semantic definition, is a data structure. An instance of a data structure is a data set. A data set, further defined, constrained, and rendered in some binding is a data instance.

See Also: data element; data instance; data set; data structure.

3.9 data set

A data structure in its second definition, i.e., "an instance ... of data elements".

Note: A data set is independent of binding (binding-independent).

3.10 data structure

(1) The datatype of an aggregate of zero or more data elements.

(2) An instance of an aggregate of zero or more data elements.

Note 1: In a different context a data structure may be considered a whole, indivisible unit, i.e., in this context a data structure is a data element of some higher level data structure.

Note 2: The term "aggregate" is defined in ISO/IEC 11404.

Examples: a record; a set; a sequence; a list; an array.

3.11 encoding

The bit and byte format and representation of information.

3.12 extended data element

Within the appropriate context, an element of a data structure that is defined outside a standard, and may be used within an instance of the data structure, as permitted by data interchange participants and data interchange implementations.

The "extended" nature of a data element is an obligation attribute.

The "extended" nature of a data element is a conformance level feature (e.g., strictly conforming implementations vs. conforming implementations).

Example: mandatory extended data element, optional extended data element, conditional extended data element.

See Also: conditional data element; mandatory data element; obligation (data element); optional data element.

3.13 generate (data)

To transform data from its meaning to some form suitable for data interchange.

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

See Also: interpret (data); produce (data).

3.14 implementation behavior

External observation, appearance, or action.

See Also: implementation-defined behavior; implementation value; undefined behavior; unspecified behavior.

3.15 implementation-defined behavior/value

Unspecified behavior or an unspecified value(s) where each implementation documents how the choice is made.

See Also: implementation behavior; undefined behavior/value; unspecified behavior/value.

Example: Permitting a maximum size, as measured in octets, of a coding.

3.16 implementation value

A quantifiable artifact associated with an implementation.

See Also: implementation behavior; implementation-defined behavior/value; undefined behavior/value; unspecified behavior/value.

To process data to discover its meaning, to the extent required by this Guideline.

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.

As described above, an application that combines data consumption and data interpretation, but only interprets standardized data elements, might be strictly conforming data reader.

3.17 locale-specific behavior

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

3.18 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 this Guideline, but is intended to be excluded in future editions of this Guideline.

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

3.19 mandatory data element

Within the appropriate context, an element of a data structure that is defined and required within an instance of the data structure.

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

See Also: conditional data element; extended data element; obligation (data element); optional data element.

3.20 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 subsystems and related systems.

Note: Also known as "sometimes-connectivity" and/or "sometimes-roaming".

3.21 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
// "conditional" wording above only
// makes requirements and makes no
// prohibitions
( A=123, B=nil, C=345 ) // valid
```

3.22 obsolete data element

Within the appropriate context, an element of a data structure that is defined but should not be used within an instance of the data structure.

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

See Also: longevity (data element); reserved data element.

Note: The use of obsolete data elements is deprecated and their specification may be removed from future revisions of a standard.

3.23 optional data element

Within the appropriate context, an element of a data structure that is defined and permitted, but not required within an instance of the data structure.

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

See Also: conditional data element; extended data element; mandatory data element; obligation (data element).

3.24 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.

3.25 repository

A collection of data sets and data access methods for storing, indexing, searching, and retrieving information.

3.26 reserved data element

Within the appropriate context, an element of a data structure that is not defined and not permitted to be used within an instance of the data structure.

The "reserved" nature of a data element is a longevity attribute.

See Also: longevity (data element); obsolete data element.

3.27 undefined behavior/value

Implementation behavior or an implementation value(s) for which a standard imposes no requirements.

See Also: implementation behavior; implementation value; implementation-defined behavior/value; unspecified behavior/value.

Example 1: 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

Example 2: Possible undefined values include infinities, null values, and "Not A Number".

3.28 unspecified behavior/value

Implementation behavior or an implementation value(s) for which a standard provides two or more possibilities and imposes no further requirements on which possibility is chosen in any instance.

See Also: implementation behavior; implementation value; implementation-defined behavior/value; undefined behavior/value.

Example 1: An application's choice of algorithm for creating object identifiers.

Example 2: The order in which procedure call parameters are pushed on a calling stack.

3.29 Acronyms and abbreviations

- API: Application Programming Interface
- HTTP: Hypertext Transfer Protocol
- ICS: Implementation Conformance Statement
- IETF: Internet Engineering Task Force
- LID: Language Independent Datatypes; also known as ISO/IEC 11404
- LTSC: Learning Technology Standards Committee
- RFC: Request for Comments; a citation prefix for specifications developed by the Internet Engineering Task Force
- SPM: smallest permitted maximum
- W3C: World Wide Web Consortium
- XML: Extensible Markup Language

4 Standards wording template

Note 1: The following wording may be included in the normative (or conditionally normative) part of the standards wording. The phrase "Standard Such-And-Such" should be replaced with the name of the standard where this wording is being incorporated.

Note 2: The implementation varieties are defined in IEEE 1484.14.1, Guidelines for Extension Techniques.

If a Standard Such-And-Such application includes a "Standard Such-And-Such [*implementation variety*]" with DNVP coding binding" in its implementation conformance statement, then that application shall conform to the requirements of this Clause/Annex.

Examples: "Standard Such-And-Such strictly conforming data instance with DNVP coding binding"; "Standard Such-And-Such conforming data reader with DNVP coding binding".

Note 2: 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
```

4.1 Dotted Name-Value Pairs (DNVPs)

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

4.1.1 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 character in the range 0-31 (decimal) or the octet 127 (decimal) but not the octet 9 (HT).

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 word hello, and the quote characters that surround the word hello have no special meaning.

4.1.2 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.

4.1.3 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.

4.1.4 Syntax summary

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>

```

4.2 Generating and producing dotted name-value pairs

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

- **Rule 1:** 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, elements are represented by repeated DNVP field names based on the identifier of the aggregate, not the index of the element. A field name is followed by a colon (":"), followed by the value of its associated data element.
- **Rule 2:** Transform the following DNVP field names (wildcard notation):
`Standard_Such_And_Such_*`
 to the following DNVP field names (wildcard notation):
`IEEE_LTSC_Standard_Such_And_Such_*`

Rationale

The following is a rationale for these two rules for *this specific* transformation of the Standard Such-And-Such datatypes.

Note: This RFC 822-like binding (Standard Such-And-Such → 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: 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", re-

quires 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 Standard Such-And-Such 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: xxx
A.F: yyy
A.F: zzz
```

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.

4.3 Consuming and interpreting dotted name-value pairs

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

- Rule 1:** Transform the following XML tags (wildcard notation):
`IEEE_Standard_Such_And_Such_*`
 to the following XML tags (wildcard notation):
`Standard_Such_And_Such_*`
- Rule 2:** For each field name that is associated with an identifier defined by a Standard Such-And-Such data element in this Guideline, 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 20.2, 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 Standard Such-And-Such data element in this Guideline, map each field name to the corresponding data element identifier", (1) ignores all identifiers that are unknown to this Guideline, (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 Guideline, 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.

4.4 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.

4.4.1 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 .

4.4.2 Integers

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

4.4.3 Real numbers

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

4.4.4 Date and time values

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

4.4.5 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

```

4.5 Encoding of character representations

Conforming Standard Such-And-Such 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.

4.6 Handling exceptions and extensions

4.6.1 Implementation-defined behavior

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

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 20.1.1, Basic Lexical Elements, above.
- The maximum size, in characters, of a strictly conforming Standard Such-And-Such 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.

4.6.2 Unspecified behavior

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

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

- The order of processing data elements.

4.6.3 Undefined behavior

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

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 Guideline.

4.7 Sample DNVP records derived from the binding

Note: The standard might supply examples of the DNVP binding.

5 Annex A: Document development

This section concerns the development of this document. The past (revision history, resolved issues), present (release notes, comment returns), and future (open issues) releases of this document are identified here.

5.1 Revision history

- **Draft 1, 2001-02-06**, the first draft. This document was excerpted from IEEE 1484.2, Draft 6.

5.2 Release notes for this document

The following notes apply to this release of this Guideline:

- There may be some "disconnected" wording due to the cut and paste.

5.3 Resolved issues

The following issues have been resolved:

- None yet.

5.4 Open issues

The following issues are outstanding:

- Need to write corresponding PAR.

5.5 Comments on this document

All comments are appreciated. Please return all comments on this release of this document by **Friday, 2001-03-16 23:00 UTC**. Deliver all comments to the IEEE 1484.14 Semantics and Exchange Bindings Working Group by sending E-mail to:

ltsc-sxb@majordomo.ieee.org

To subscribe to the working group mailing list, send the one-line message

subscribe ltsc-sxb

to the E-mail address "majordomo@majordomo.ieee.org".

The Technical Editor may be contacted at any one of the following:

Telephone: +1 212 486 4700

Fax: +1 212 759 1605

E-mail: frank@farance.com

Frank Farance

2001-02-06

P1484.14.3D1

Farance Inc.
Island Box 256
New York, NY
10044-0205
USA