

# IEEE 1484.14.2/D1, 2001-02-06

## Draft Guidelines for Learning Technology — Rule-Based XML 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 P1484.14.2, Guidelines for Learning Technology — Rule-Based XML 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

<b>1 Overview .....</b>	<b>6</b>
1.1 Scope.....	6
1.2 Purpose.....	6
<b>2 Normative references .....</b>	<b>7</b>
<b>3 Definitions .....</b>	<b>8</b>
3.1 Definitions incorporated via normative reference .....	8
3.2 aggregate (datatype, value) .....	8
3.3 binding .....	8
3.4 coding.....	8
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 .....	13
3.20 nomadic (access, system).....	14
3.21 obligation (data element).....	14
3.22 obsolete data element .....	14
3.23 optional data element .....	14
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 .....	15
3.29 Acronyms and abbreviations .....	16
<b>4 XML coding binding template.....</b>	<b>17</b>
4.1 Generating and producing XML.....	17
4.2 Consuming and interpreting XML .....	20
4.3 Representation of basic data types.....	21
4.3.1 Characters and character strings .....	21
4.3.2 Integers .....	22
4.3.3 Real numbers.....	22
4.3.4 Date and time values.....	22
4.3.5 Void types.....	24
4.4 Encoding of character representations.....	25
4.5 Handling exceptions and extensions .....	25
4.5.1 Implementation-defined behavior .....	25
4.5.2 Unspecified behavior .....	25

4.5.3 Undefined behavior.....25

**5 Annex A: Document development.....27**

5.1 Revision history .....27

5.2 Release notes for this document .....27

5.3 Resolved issues.....27

5.4 Open issues .....27

5.5 Comments on this document .....27

# **1 Overview**

## **1.1 Scope**

This Guideline describes techniques for rule-based XML coding 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 when rule-based binding approaches are possible. Because 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: The following definitions are being/have been harmonized with the IEEE 1484.3 Glossary and Reference Materials.

### 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 "<B>...</B>" 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
- L10N: localization; the internationalized spelling of "localization" (the letter "L", followed by 10 letters, then the letter "N")
- LID: Language Independent Datatypes; also known as ISO/IEC 11404
- LTSC: Learning Technology Standards Committee
- MDAS API: Metadata Access Service API
- 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 XML coding binding 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 XML 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 XML coding binding"; "Standard Such-And-Such conforming data reader with XML coding binding".

### 4.1 Generating and producing XML

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

- **Rule 1:** 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). 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:** Map all **mlstring\_type** datatypes to:
  - **Rule 2A:** The **locale** element of **mlstring\_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 Standard Such-And-Such personal information to ...[editor's note: "vCard" mapping to be added here]
- **Rule 3:** Transform the following XML tags (wildcard notation):  
**standard\_such\_and\_such\_\***  
 to the following XML tags (wildcard notation):  
**IEEE\_LTSC\_standard\_such\_and\_such\_\***

All data produced shall be well-formed XML.

#### Rationale

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

Note: This XML binding (Standard Such-And-Such  $\rightarrow$  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_mlstring_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:"  $\Rightarrow$  "<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"  $\Rightarrow$  "<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

Standard Such-And-Such records use several specialized datatypes, such as multilingual datatypes for describing certain `characterstring`-type data elements that must be represented in a multilingual and multicultural context — commonly called internationalization (I18N) and localization (L10N) features. Below, is a sample version of a multilingual data type that is not intended to clash with definitions of other multilingual datatypes defined elsewhere in this Standard. In this illustration, the datatype `sample_mlstring_type` represents a single pair: a localized string and a locale specification (L10N mapping). The datatype `sample_mlstring_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_mlstring_type =
record
(
  L10N_string: characterstring(iso-10646-1),
  L10N_locale: string_type,
),

type sample_mlstring_array_type =
array (0..limit) of (sample_mlstring_type),

value example_remarks:
sample_mlstring_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 Standard Such-And-Such personal 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.

## 4.2 Consuming and interpreting XML

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

All data consumed shall be well-formed XML.

- **Rule 1:** Transform the following XML tags (wildcard notation):  
`IEEE_LTSC_Standard_Such_And_Such_*`  
to the following XML tags (wildcard notation):  
`Standard_Such_And_Such_*`
- **Rule 2:** Transform the following:
  - **Rule 2A:** The `LANG` attribute of the XML element sets the `locale` element of the corresponding `mlstring_type` data element.
  - **Rule 2B:** The contents of the tagged element sets the `string` element of the corresponding `mlstring_type` data element.
- **Rule 3:** For each XML tag, that is associated with an identifier defined by a Standard Such-And-Such 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 12.1, 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\_LTSC\_**") 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 **mlstring\_type** datatype. This rule is careful to transform only known **mlstring\_type** data elements because all other XML **LANG** attributes do not correspond to **mlstring\_type** data elements in this Standard.

[Editor's Note: In the next draft, Rule 2, will include "vCard" mapping which transforms Standard Such-And-Such personal 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 Standard Such-And-Such 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.

## **4.3 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.

### **4.3.1 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 "&#169; ", to represent these symbols.

### 4.3.2 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
```

### 4.3.3 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
```

### 4.3.4 Date and time values

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 timezones 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 basic format shall be used. Example: **"19990102"** and **"030405"** are valid, but **"1999-01-02"** and **"03:04:05"** are not. Rationale: The additional processing for extended formats may be more error prone and can reduce interoperability; the additional lexical elements of the extended formats may interfere or conflict with other lexical, embedded, or surrounding information processing environments.
- Decimal fractions shall use the full stop character ("**.**"), also known as the period character. Example: **"199901020304.1"** represents 03:04:06 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: The comma character may interfere or conflict with other lexical, embedded, or surrounding information processing environments.

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 in-

terpretation, 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."

### 4.3.5 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>
```

## 4.4 Encoding of character representations

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

Conforming Standard Such-And-Such 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 Standard Such-And-Such 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.

## 4.5 Handling exceptions and extensions

### 4.5.1 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 Standard Such-And-Such 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.

### 4.5.2 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.

### 4.5.3 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.

## 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.
- Need to include examples where typing information is the main identifier.

### 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](mailto: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](mailto: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](mailto:frank@farance.com)

2001-02-06

P1484.14.2D1

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