

# IEEE 1484.14.1/D1, 2001-02-06

## Draft Guidelines for Learning Technology — Data Extension 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 P1484.14.1, Data Extensions 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 Application model.....</b>	<b>17</b>
4.1 Data access model .....	17
4.2 General data operations.....	18
4.3 Examples of application-specific data operations.....	18
<b>5 Conformance model .....</b>	<b>20</b>
5.1 Conformance level model .....	20
5.2 Is it enough to specify only "conforming".....	21
5.3 Is it enough to specify only "strictly conforming".....	21
5.4 Both must be specified.....	21
<b>6 Obligation of data elements.....</b>	<b>22</b>
6.1 Mandatory data elements .....	22
6.2 Optional data elements.....	22
6.3 Conditional data elements.....	22

6.4 Extended data elements.....	23
6.5 Mandatory vs. optional.....	24
6.6 Extended vs. reserved.....	25
<b>7 Longevity of data elements .....</b>	<b>26</b>
7.1 Obsolete data elements.....	26
7.2 Reserved data elements.....	26
7.3 Recursive/contextual nature of obligation/longevity .....	27
<b>8 Summary of conformance labels .....</b>	<b>28</b>
<b>9 Conformance wording template.....</b>	<b>30</b>
9.1 Conformance levels.....	30
9.1.1 Strictly conforming implementations.....	30
9.1.2 Conforming implementations.....	30
9.1.3 Non-conforming implementations .....	31
9.2 Coding conformance .....	31
9.2.1 Data set conformance.....	31
9.2.2 Data instance conformance.....	31
9.3 API conformance.....	32
9.4 Protocol conformance .....	33
9.5 Data Application conformance.....	33
9.5.1 Strictly conforming data application .....	33
9.5.2 Conforming data application .....	34
9.5.3 Data repository .....	34
9.5.4 Data reader.....	35
9.5.5 Data writer.....	36
<b>10 Both strictly conforming and conforming data applications .....</b>	<b>37</b>
<b>11 Handling extensions.....</b>	<b>38</b>
11.1 A notion of "core" sets of data elements .....	38
11.2 Approaches towards handling common problems .....	42
11.2.1 Supporting several implementation models .....	42
11.2.2 Conforming vs. strictly conforming.....	42
11.2.3 Separate phases of translation .....	43
11.2.4 Standards lifecycle for incorporating extensions.....	44
<b>12 Annex A: Document development .....</b>	<b>45</b>
12.1 Revision history.....	45
12.2 Release notes for this document .....	45
12.3 Resolved issues.....	45
12.4 Open issues .....	45
12.5 Comments on this document .....	45

# 1 Overview

## 1.1 Scope

This Guideline specifies extension techniques that are commonly used when standardizing data models and their bindings. In this context, "extension" refers to capabilities beyond those described in a standard. These techniques are general across many application areas.

## 1.2 Purpose

The purpose of this Guideline is to provide guidance for the developers of standards of data models and their bindings regarding commonly used techniques that support competing technical and business interests among users, vendors, and industry.

This Guideline addresses the several interoperability issues. The level of interoperability is related to the level of conformance to a standard. However, conformance is subtly different from interoperability: conformance is the satisfaction, by an implementation (or a system), of the requirements of a standard (or specification), while interoperability is the successful interaction among two or more implementations and the automation, to the level desired, of these interactions. For example, several combinations of interoperability are possible:

- Scenario #1: An implementation interoperates only among strictly conforming implementations. Example: The implementation might only include the features of a standard, but no extensions or proprietary features are used.
- Scenario #2: An implementation interoperates with other implementations from the same vendor, user, or institution.
- Scenario #3: An implementation interoperates with a wide variety of user-specific, vendor-specific, institution-specific, and/or industry-specific extensions.

It is expected that this Guideline will be used in connection with the consensus-building process itself. The use of extensions and their later standardization is a common migration path for new technologies merging into the mainstream.

**Example:** Users, vendors, institutions, industries, and others will desire additional features (extensions) to support their specific needs. Strictly conforming implementations are not permitted to use extensions, but conforming implementations may use extensions, as permitted by implementations and as permitted data interchange interoperability requirements. As certain extensions become widely used, the consensus-building process may choose to incorporate these features into a standard via a future amendment or revision. Thus, widely used extensions that are used now may later become additional requirements that will be imposed upon future, strictly conforming (maximally interoperable) implementations

## 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
- 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 Application model

For this Guideline, "applications" are characterized as one or more of the following: (1) data instance, (2) data reader, (3) data writer, (4) data repository, (5) data application. These applications may use or conform to one or more standards, such as coding bindings, API bindings, or protocol bindings.

### 4.1 Data access model

Applications may include the following features in the conceptual model of data access.

*Note: In the following, "shall" and "may" assertions may apply to the related standard, not this Guideline.*

- **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 an identifier.
- **Data Retrieval Model.** Data, including data sets, may be retrieved from a data object, as referenced by an identifier.
- **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 datatypes of standard *such-and-such* data elements use 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.** Standard bindings define access, if any, to data repositories.
- **Data Repository Security Model.** A standard may define a security model.
- **Data Persistence Model.** The lifetime of data objects shall be implementation-defined.
- **Data Navigation Model.** The techniques for navigating data structures are defined in bindings standards.
- **Data Identification Model.** The identification, labeling, namespace, and their associated techniques shall be 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 shall be 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 shall be implementation-defined.

- **Data Indexing Model.** The indexing methods for data repositories shall be 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 shall be implementation-defined.

## 4.2 General data operations

Standards may support the following data management operations on data sets:

- **Create Operation.** Creating a new instance of some information type, such as personal information.
- **Destroy Operation.** Discarding an instance of an information type in the context of its storage. Note: Compare destroying a record in application memory, in temporary storage, and in a database.
- **Copy Operation.** Creating a new instance of an information type with identical contents.
- **Move Operation.** Changing a label 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 creating new "hard links" to a new label, then deleting "hard links" to the old label. 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 and a reference 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.

The binding standards define the methods for accessing data management operations and the availability of the above operations.

## 4.3 Examples of application-specific data operations

Standards may support application-specific data operations on data sets:

- **Accumulation Operations.** Data sets may be accumulated, aggregated, or analyzed. Examples: The summation of all records that meet criteria X.
- **Time Compression and Expansion Operations.** For time series data, data sets 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.
- **Sort-Merge Operations.** Ordered and merged based upon criteria. Example: Sorting records alphabetically by a particular data element.

The binding standards define the methods for accessing application-specific data operations and the availability of the above operations.

## 5 Conformance model

This Guideline describes the conformance and extension techniques for data model standards and their binding standards. The conformance model describes how other standards address conformance issues. This Guideline is informative — there are no conformance requirements for this Guideline.

In this Guideline, "shall" is to be interpreted as a requirement on an implementation the conforms to a standard (*i.e., a related standard, not this document*); "shall not" is to be interpreted as a prohibition within a standard. If a "shall" requirement or "shall not" prohibition is violated, the behavior is undefined. Undefined behavior is otherwise indicated in this Guideline or a 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".

### Rationale

Conformance (to a standard) partly concerns the structure of data sets and partly concerns the behavior of systems. In other words, conformance has both a non-behavioral and a behavioral dimension, and both are important. The term "behavior" is used in a general sense, e.g., a data set doesn't exhibit "behavior", but implementations that store, retrieve, generate, and interpret data sets all exhibit "behavior" — the notions of "undefined", "implementation-defined", and "unspecified" behaviors are defined, even in the context of a data set.

### 5.1 Conformance level model

There are at least two levels of conformance necessary to support technical and industry demands of users, vendors, institutions, and others. A standard that only specifies a minimal set of features may be inadequate for industry and interoperability needs, e.g., too few features to meet needs technical and business needs. A standard that specifies too many features may be too complex or expensive to implement, which may impede adoption of the standard.

For this subclause, the term "participant" is used in the context of a "participant in an interoperability scenario". Interoperability scenarios can include any of the "applications" (data instances, data readers, data writers, etc.) and any of the data interchange standards (e.g., codings bindings, API bindings, protocol bindings). A "participant" is any "application" in the interoperability scenario.

The use of "conformance levels" may permit a range of implementations, and a range of interoperability. Some standards provide N conformance levels, which requires the need to specify and test N\*N interoperability scenarios. This Guideline does not address an N-level conformance paradigm.

This Guideline addresses the kind of standard that has a single level of conformance, but permits some kind of extended capabilities outside the standard. To make use of these extended features, the participants of this interoperability scenario must make prior agreements, such as

bilateral arrangements, specifications, or other standards. Thus, one of the following is true for the participants:

1. Only the features of the standard are used. No extensions are used. This kind of participant is known as "strictly conforming". Conclusion: Interoperability is achieved because participants only used the minimum features, as specified by the standard.
2. The features of the standard are used. Some extensions are used, based upon mutual agreement (e.g., bilateral agreements, specifications, other standards). This kind of participant is known as "conforming". Conclusion: Interoperability is achieved because participants agree upon the use of a *particular set* of extensions.
3. The features of the standard are used. Some extensions are used, but agreements are only unilateral. This kind of participant is "conforming" but might not interoperate with other participants because there is no prior agreement on the use of *specific* extensions. Conclusion: Interoperability has *not* been assured.

## 5.2 Is it enough to specify only "conforming"?

Is it possible that a standard need only specify "conforming" aspects and not distinguish the "strictly conforming" aspects? The answer is No because certain conforming implementations will want the distinction that they interoperate with all conforming systems, not just a subset. That distinction is known as "strictly conforming". Vendors want the distinction because it makes their products more desirable. Consumers want it because they reduce the integration and maintenance risk.

Thus, both "conforming" and "strictly conforming" are needed.

## 5.3 Is it enough to specify only "strictly conforming"?

Is it possible that a standard need only specify the minimum requirements and those that "extend" the standard are not conforming? The answer is No because consumers and vendors will demand systems with extended capabilities — they would not like their "conforming" status lost because a few features were added.

Thus, both "conforming" and "strictly conforming" are needed.

## 5.4 Both must be specified

The distinction between "strictly conforming" and "conforming" implementations is necessary to address the simultaneous needs for interoperability and extensions. Standards specify both. This Guideline describes techniques for defining standards that promote interoperability. Extensions are motivated by needs of users, vendors, institutions, and industries (1) that are not directly specified by the related standard, (2) that are specified and agreed to outside the related standard, and (3) that may serve as trial usage for future editions of the related standard.

## 6 Obligation of data elements

There are four types of obligation attributes for data elements: mandatory, optional, conditional, and extended. The obligation attribute concerns the validity of the data structure.

### 6.1 Mandatory data elements

Mandatory data elements are always required for the data structure to be valid. All data sets (and data instances) are required to include these elements. All data applications are required to support these elements.

An implementation that does not support or include one or more mandatory data elements is a non-conforming implementation.

### 6.2 Optional data elements

Optional data elements are permitted, but not required, for the data structure to be valid. A data set (and data instance) is permitted, but not required, to include these data elements. Because all data repositories and data readers are required to support all valid data sets, effectively, data repositories and data readers are required to support all optional data elements. This might be confusing because "optional" is *not* optional for data repositories and data readers — the obligation attribute "optional" applies to the *validity of the data structure* ("optional" is optional for instances of the data structure). A data writer is required to generate and produce the optional elements of each data instance that is generated and produced.

An implementation that does not support one or more optional data elements is a non-conforming implementation.

If an implementation includes or supports these data elements, their use is specified by the related standard.

An implementation that includes or supports an optional data element, but includes or supports it in ways that are inconsistent with this Guideline, is a non-conforming implementation. The attribute "optional" does *not* imply that the implementation has license to implement the data element in any way ("at the option of the implementer"); if the data element is implemented, its requirements are specified in the related standard.

### 6.3 Conditional data elements

Conditional data elements are required, but their requirement is dependent upon certain conditions (as defined elsewhere in the related standard). Each conditional data element may individually have a set of conditions. If the conditions are met, the data element is required to be included for the data structure to be valid. Thus, a data set (and data instance) is required to include these elements if, individually, each condition is met. By the same reasoning as for

optional data elements (above), all data repositories and data readers are required to support all conditional data elements. By the same reasoning above, a data writer is required to support all the conditional elements for each and every data instance generated and produced.

An implementation that does not support one or more conditional data elements is a non-conforming implementation.

An implementation that includes or supports a conditional data element, but includes or supports it in ways that are inconsistent with the related standard, is a non-conforming implementation.

## 6.4 Extended data elements

Extended data elements are not permitted within strictly conforming implementations.

Extended data elements are permitted within conforming implementations to the extent that the implementation individually supports each extended data element, i.e., (1) the implementation allows and uses specified extended data elements, (2) the data interchange participants allow and use specific data elements, and (3) other extended data elements are not used.

For conforming implementations that support extended elements, these elements individually may have their own obligation attributes, e.g., it is possible to have mandatory extended data elements, optional extended data elements, and conditional extended data elements. These obligation attributes determine the validity of the data structure in the context of extended data elements, e.g., an optional extended data element (1) permits but does not require the data element for the data structure to be valid, (2) for conforming implementations that support this extended data element.

Note: Mandatory extended data elements can cause interoperability problems because a mandatory extended data element (1) requires the data element to exist for the data structure to be valid, (2) for conforming implementations that support this extended data element. In other words, (1) only implementations that support this extended data element are interoperable; and (2) *no* strictly conforming implementations will interoperate because *extended features are required* for interoperability.

There are no generic techniques or methods for both supporting extended data elements or extension features and supporting full semantic interoperability; there are only specific techniques and methods for supporting extended data elements (e.g., supported to the extent allowed, as above).

The use of extended data elements outside these circumstances (unsupported environments) causes undefined behavior, which might be:

- appropriate, e.g., ignoring an offending data element if it is an unimportant feature
- inappropriate, e.g., ignoring an offending data element if it is an important feature, such as a security classification
- innocuous, e.g., error messages
- disruptive, e.g., error messages

- predictable, e.g., a program aborting, exiting ungracefully, exiting unexpectedly, or "hanging" indefinitely
- unpredictable, e.g., a program aborting, exiting ungracefully, exiting unexpectedly, or "hanging" indefinitely

There is no correct generalized method for handling undefined behavior. Any particular method for handling undefined behavior can be desirable, undesirable, or both.

Some bindings "relax" the processing of unrecognized extended data elements. Normally, extended data elements create *undefined behavior* but certain bindings "relax" these requirements to *implementation-defined behavior* or even *ignoring* unrecognized extended data elements — both of these "relaxed" processing requirements (implementation-defined behavior; ignoring unrecognized or extended data elements) can be less disruptive. Annex J, Extension Techniques, contains informative wording on the relationship among data modeling, conformance levels, interoperability, industry concerns, and the standards process.

Extended data elements are both an obligation (data modeling) feature and a conformance level feature (strictly conforming vs. conforming).

## 6.5 Mandatory vs. optional

Certain data elements are defined as mandatory data elements while others are defined as optional data elements. These notions of "mandatory" and "optional" can be confusing for vendors, implementers, applications, and repositories that intend to conform to a standard. Some application vendors believe "optional" means optional for applications, too, but this is not true. The following is an illustration of "mandatory" vs. "optional", and "data instance" vs. "application":

A person is filling out a paper form at some government agency. Some of the blanks on the paper form are marked "required" (e.g., name and address) — *these are the mandatory data elements*. Some of the blanks are marked "optional" or have no marking at all (e.g., home phone number) — *these are the optional data elements*.

When the paper form is handed into the agency clerk, if some of the "required" fields are incomplete or completed improperly, the clerk rejects the form — *the form is invalid because some mandatory data elements are missing, i.e., the data instance is not conforming*.

Regarding the "optional" fields, such as home phone number, the agency clerk can accept the paper form with or without the home phone number, but the result might not be as useful (e.g., it might be quicker to contact the person if they give their home phone number) — *the form is valid because the data instance is conforming*.

Although certain blanks in the form are "required" and others are "optional", the data entry clerks, databases, and computer systems must "understand" all the elements of the paper form — *a conforming application must support all mandatory and all optional data elements*.

Thus, all optional data elements are required to be "understood" by conforming applications.

## 6.6 Extended vs. reserved

An extended data element may be defined in some external specification, but not defined in the (this) standard. A reserved data element is explicitly undefined. A typical migration of a particular data element might be:

- Edition N of the standard: **x** is a reserved data element; any implementation that attempts to use **x** is not a strictly conforming implementation (however, the application may be a conforming implementation).
- Edition N+1 of the standard: **x** is an extended data element; the standard might define **x** for certain levels of conformance.
- Edition N+2 of the standard: **x** is mandatory, optional, or conditional data element.
- Edition N+3 of the standard: **x** is a mandatory data element.

Although strictly conforming implementations cannot use extended or reserved data elements, a conforming implementation may use a reserved data element by simply considering it as an extension to the standard.

Reserving a data element so that it cannot be overridden by extended data elements is achieved by defining (within the standard) an optional data element with type **"void"**.

## 7 Longevity of data elements

The following longevity attributes indicate intentions for incorporation into past, present, or future editions of this Guideline.

Longevity attributes are independent of obligation attributes.

### 7.1 Obsolete data elements

Obsolete data elements are defined in the current edition of the related standard and may be defined in prior editions. The "obsolete" feature indicates that the definition of the data element is intended to be removed from future editions of this the related standard.

Implementations should not use obsolete data elements. Implementations that do use obsolete data elements should plan accordingly for future editions of this the related standard.

An implementation's use of an obsolete data element does not imply that the implementation is non-conforming. Strictly conforming implementations and conforming implementations may still use obsolete data elements for this edition of the Standard.

The "obsolete" feature is independent of the obligation attribute, so there might be obsolete mandatory data elements, obsolete optional data elements, obsolete conditional data elements, and obsolete extended data elements.

### 7.2 Reserved data elements

Reserved data elements are not defined in this edition of the related standard. Data elements may be reserved because (1) they were defined in previous edition(s) of the related standard, or (2) they will be defined in some future edition(s) of this the related standard.

A reserved data element is not permitted in a strictly conforming implementation.

A "reserved data element" might be used in a conforming implementation if (1) the reserved data element were defined, (2) it were defined as an extended data element, and (3) the extended data element were "supported" by implementations and data interchange participants (see below). In other words, a particular implementation extends or overrides (the non-definition of) the "reserved data element" by defining implementation extensions.

Although the "reserved" feature is independent of the obligation attribute, a reserved data element has no definition. Therefore, there are no reserved mandatory data elements, no reserved optional data elements, no reserved conditional data elements, and no reserved extended data elements because mandatory data elements, optional data elements, conditional data elements, and extended data elements all imply a definition of a data element, which conflicts with the undefined nature of "reserved".

Data elements that are defined, but are to be incorporated into future editions of the related standard, are extended data elements (i.e., they are *not* reserved data elements). As these extended data elements become incorporated into a future edition, they will become mandatory data elements, optional data elements, or conditional data elements.

Extended data elements may be defined (1) in an informative Annex in the related standard, (2) in a conditionally normative Annex in the related standard, or (3) in a specification outside the related standard.

Extended data elements are not required for this edition of the Standard, i.e., (1) extended data elements are prohibited for strictly conforming systems; (2) extended data elements are not required for conforming systems; and (3) extended data elements, if defined, are not in the Clauses of the related standard.

Some bindings "relax" the processing of unrecognized data elements, such as reserved data elements. Normally, reserved data elements create *undefined behavior* but certain bindings "relax" these requirements to *implementation-defined behavior* or even *ignoring* unrecognized or reserved data elements — both of these "relaxed" processing requirements (implementation-defined behavior; ignoring unrecognized or reserved data elements) can be less disruptive.

Conforming implementations may use extended data elements to the extent permitted by the implementation and data interchange participants. See subclause x.x, Extended Data Elements, above, for further details.

Reserving a data element so that it cannot be overridden by extended data elements is achieved by defining an optional data element with ISO/IEC 11404 datatype **void**.

### 7.3 Recursive/contextual nature of obligation/longevity

An obligation attribute or a longevity attribute of an aggregate data element applies to the aggregate itself, but only indirectly to its components. In the context of the existence of an aggregate and its components, each component individually has its own obligation and longevity attributes (among other attributes). This determination of context and obligation/longevity attributes is applied recursively for all aggregate data elements.

Example: A data element **x** is optional, and **x** has two subelements: **y** is mandatory and **z** is optional. Letting the notation **P.Q** represent the subelement **Q** of **P**, then

- if **x** does not exist, then **x.y** and **x.z** cannot exist; stated differently, if **x.y** or **x.z** exists, then **x** exists
- if **x** exists, then **x.y** is required to exist for all conforming implementations
- if **x** exists, then **x.z** is permitted to exist for all conforming implementations
- if **x** exists and **x.y** does not exist, then the implementation is non-conforming

Thus, **y** only becomes mandatory if **x** exists.

## 8 Summary of conformance labels

The following is an *informative* summary of the possible implementation varieties used in implementation conformance statements:

- **Strictly conforming data set:** binding-independent; all mandatory data elements shall exist; some optional data elements may exist; extended data elements shall not exist.
- **Conforming data set:** binding-independent; all mandatory data elements shall exist; some optional data elements may exist; some extended data elements may exist.
- **Strictly conforming data instance:** a binding shall be specified; all mandatory data elements shall exist; some optional data elements may exist; extended data elements shall not exist.
- **Conforming data instance:** a binding shall be specified; all mandatory data elements shall exist; some optional data elements may exist; some extended data elements may exist.
- **Strictly conforming data repository:** 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.
- **Conforming data repository:** 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.
- **Strictly conforming data reader:** binding(s) shall be specified; only mandatory and optional data elements are interpreted, but no extended data elements are interpreted. Note: Supplying extended data elements to a strictly conforming data reader is undefined behavior.
- **Conforming data reader:** binding(s) shall be specified; mandatory and optional data elements are interpreted and some extended data elements may be interpreted. Interpreting extended data elements is undefined behavior (note: not just "unspecified" behavior, but "undefined" behavior). This undefined behavior may be relaxed (e.g., becomes "implementation-defined" behavior), depending upon the binding.
- **Strictly conforming data writer:** binding(s) shall be specified; shall generate all mandatory data elements; may generate optional data elements; shall not generate extended data elements.
- **Conforming data writer:** binding(s) shall be specified; shall generate all mandatory data elements; may generate optional data elements; may generate extended data elements.
- **Conforming X API environment:** \*\*\* TO BE SUPPLIED \*\*\*.
- **Strictly Conforming X API application:** \*\*\* TO BE SUPPLIED \*\*\*.
- **Conforming X API application:** \*\*\* TO BE SUPPLIED \*\*\*.
- **Strictly Conforming X protocol:** \*\*\* TO BE SUPPLIED \*\*\*.
- **Conforming X protocol:** \*\*\* TO BE SUPPLIED \*\*\*.
- **Negotiable Conforming X protocol:** \*\*\* TO BE SUPPLIED \*\*\*.

Note: An implementation may claim more than one type of conformance in its implementation conformance statement (ICS).

## 9 Conformance wording template

Note: This Clause specifies a template for standards wording of the related standard.

### 9.1 Conformance levels

#### 9.1.1 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 minimum 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.

#### 9.1.2 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 minimum 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.

### **9.1.3 Non-conforming implementations**

An implementation that does not conform to this Standard (either strictly conforming or merely conforming), is a non-conforming implementation.

## **9.2 Coding conformance**

A strictly conforming Standard Such-And-Such coding shall be at least one of: a strictly conforming data set, or a strictly conforming data instance.

A conforming Standard Such-And-Such coding shall be at least one of: a conforming data set, or a conforming data instance.

A Standard Such-And-Such coding shall conform to Clause X, Functionality; conform to Clause Y, Conceptual Model; and, conform to the datatypes specified in Clause Z, Semantics.

### **9.2.1 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 this 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 this 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.

### **9.2.2 Data instance conformance**

A strictly conforming data instance shall (1) be a strictly conforming data set, and (2) strictly conform to at least one Standard Such-And-Such coding.

A conforming data instance shall (1) be a conforming data set, and (2) conform to at least one Standard Such-And-Such coding.

Note 1: The term "Standard Such-And-Such coding" is used in the two paragraphs above and its requirements are specified in the third paragraph of subclause x.x, Coding Conformance.

Note 2: The difference between a strictly conforming/conforming data set, a strictly conforming/conforming coding, and a strictly conforming/conforming 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 Standard Such-And-Such 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.

### **Rationale**

In addition to the three application conformance perspectives (data repository, data reader, data writer), there is a fourth perspective on conformance: the data instance. Users will want to claim conformance for particular data instances ("My Standard Such-And-Such conforms to the Standard").

## **9.3 API conformance**

A strictly conforming Standard Such-And-Such API shall strictly conform to at least one Standard Such-And-Such API binding.

A conforming Standard Such-And-Such API shall conform to at least one Standard Such-And-Such API binding.

A Standard Such-And-Such API shall conform to Clause X, Functionality; conform to Clause Y, Conceptual Model; and, conform to the operations and datatypes specified in Clause Z.

### **Definitions: support, use, test, access, probe**

In reference to Standard Such-And-Such conformance, the following terms are defined in the context of API conformance: a "supported" feature is one that may be used by any application of the Standard Such-And-Such API; a feature is "used" if it is read, written, or operated upon by an application of the Standard Such-And-Such API; a feature is "tested" if an application of the Standard Such-And-Such API inquires about the existence of said feature; a feature is "accessed" if an application of Standard Such-And-Such API attempts to read or write data associated with the feature; a feature is "probed" if an application implicitly tests the existence of a feature by attempting to *use* the feature (see "use" above) within a "safe" environment that does not cause undefined behavior.

Note: API conformance makes requirements on both the API binding and on applications that use the API binding, i.e., conformity assessment of a Standard Such-And-Such implementation based on API conformance is determined by proper definition of the API and proper use of the API.

## 9.4 Protocol conformance

A strictly conforming Standard Such-And-Such protocol shall strictly conform to at least one Standard Such-And-Such protocol binding.

A conforming Standard Such-And-Such protocol shall conform to at least one Standard Such-And-Such protocol binding.

A Standard Such-And-Such protocol shall conform to Clause X, Functionality; conform to Clause Y, Conceptual Model; conform to the operations and datatypes specified in Clause Z, Semantics; and define methods for setup and knockdown of supporting communication networks.

### Definitions: support, use, test, access, probe

In reference to Standard Such-And-Such conformance, the following terms are defined in the context of protocol conformance: a "supported" feature is one that may be used by any application of the Standard Such-And-Such protocol; a feature is "used" if it is read, written, or operated upon by an application of the Standard Such-And-Such protocol; a feature is "tested" if an application of the Standard Such-And-Such protocol inquires about the existence of said feature; a feature is "accessed" if an application of Standard Such-And-Such protocol attempts to read or write data associated with the feature; a feature is "probed" if an application of the Standard Such-And-Such protocol implicitly tests the existence of a feature by attempting to *use* the feature (see "use" above) within a "safe" environment that does not cause undefined behavior.

Note: Protocol conformance makes requirements on both the protocol binding and on applications that use the protocol binding, i.e., conformity assessment of a Standard Such-And-Such implementation based on protocol conformance is determined by proper definition of the protocol and proper use of the protocol.

## 9.5 Data Application conformance

Data application conformance is measured by how well the data application behaves according to this standard.

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

### 9.5.1 Strictly conforming data application

For all strictly conforming data applications,

- Mandatory features shall exist (or shall be available) and shall conform to this standard.
- Optional features may exist (or may be available) and, if they exist (or are available), shall conform to this standard.
- Extended features shall not be directly used and shall not be tested for existence or availability. Note: A strictly conforming application might indirectly use an extended feature because that feature is hidden within an implementation; see definition of "consume" in Clause 3, Definitions, for this special case.

Note: A strictly conforming data application *may be minimally conforming but is maximally interoperable with respect to this 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.

### 9.5.2 Conforming data application

For all conforming data applications,

- Mandatory features shall exist (or shall be available) and shall conform to this standard.
- Optional features may exist (or may be available) and, if they exist (or are available), shall conform to this standard.
- Extended features may exist (or may be available), may be tested for existence (or availability), and their use and behavior shall be implementation-defined.

Note: A conforming data application *may be more useful, but may be less interoperable with respect to this 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 strictly conforming/conforming data applications: data repository, data reader, data writer.

#### Rationale

There are three separate application conformance perspectives: data repository, data reader, data writer. Vendors and administrators or data repositories will want to claim conformance ("My Standard Such-And-Such repository conforms to the Standard"). Vendors will want to claim conformance for their tools (data readers: "My application imports data and is a conforming Standard Such-And-Such data reader", data writers: "My application exports data and is a conforming Standard Such-And-Such data writer", or both). See x.x, Data Instance Conformance, above, for additional perspectives on conformance.

### 9.5.3 Data repository

A data repository is a data application that stores and retrieves data objects.

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; and
6. strictly conform to at least one Standard Such-And-Such coding binding and at least one Standard Such-And-Such API or Standard Such-And-Such 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 subclause 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 Standard Such-And-Such coding binding and at least one Standard Such-And-Such API or Standard Such-And-Such 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.

### 9.5.4 Data reader

A data reader is a data application that operates as if it (1) consumes data, and (2) interprets data which results in data sets.

Note 1: The "as if" rule implies that, conceptually, the data reader processes the information in two phases (consumption and interpretation), but the design of implementations are not constrained and implementations may use any number of phases of processing.

A strictly conforming data reader shall interpret data that strictly conforms to (1) this standard, and (2) at least one binding of this standard.

Note 2: A strictly conforming data reader does not interpret extended data elements.

Note 3: Depending upon the binding of this 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) this standard, and (2) at least one binding of this standard.

Note 4: A conforming data reader may interpret extended data elements.

### **9.5.5 Data writer**

A data writer is a data application that operates as if it (1) generates data from data sets, and (2) produces data.

Note 1: The "as if" rule implies that, conceptually, the data writer processes the information in two phases (generation and production), but the design of implementations is not constrained and implementations may use any number of phases of processing.

A strictly conforming data writer shall generate data that strictly conforms to (1) this standard, and (2) at least one binding of this standard.

Note 2: A strictly conforming data writer does not generate extended data elements.

A conforming data writer shall generate data that conforms to (1) this standard, and (2) at least one binding of this standard.

Note 3: A conforming data writer may generate extended data elements.

## 10 Both strictly conforming and conforming data applications

### Conformity assessment

Although all strictly conforming data applications are also conforming data applications, the *conformity assessment* of strictly conforming data applications may differ from the *conformity assessment* of conforming applications. The requirement that a data application must be both "a strictly conforming data application" and "a conforming data application", is a stronger requirement than the individual requirements of "a strictly conforming data application" and "a conforming data application", i.e., from the perspective of conformity assessment, an application may be "strictly conforming", "conforming", both, or neither.

### Illustrations

It is possible for a conforming data application to be simultaneously (1) a strictly conforming data repository, (2) a conforming data repository, and (3) a generator and/or interpreter of data extensions — which appears to be in conflict with the nature of strictly conforming implementations. The following examples show two difference implementation strategies. These examples use data repositories for illustration, but the illustration applies also to data readers and data writers.

Example 1: Data repository P uses an implementation strategy that allows an *arbitrary* set of data element identifiers to be stored and retrieved (in addition to those described in this Standard). When data is stored into and retrieved from P, P uses a particular binding data that data extensions are permitted to be ignored, e.g., as a coding binding that uses "extension prefixes", an API binding that "hides implementation details", or a protocol that uses "fallback negotiation and out-of-band messages". P will consume and interpret all strictly conforming data. P generates and produces strictly conforming data because the particular binding chosen "hides" the extensions; thus, all strictly conforming data readers can consume and interpret data from P. P can store extensions (although no claims are made about the specification and validity of extensions). Thus, (1) P is a strictly conforming data repository, (2) P is a conforming data repository that can store and retrieve extensions, (3) P can generate and interpret data extensions, (4) P is both a strictly conforming and a conforming data repository.

Example 2: Data repository Q uses an implementation strategy (1) that closely parallels this Standard, and (2) would be described as "minimalist". Q uses the same techniques as P does for consuming, interpreting, generating, and producing data. However, Q uses a "bucket" to store all data extensions. The "bucket" approach may have strengths (e.g., simplicity) and weaknesses (e.g., store/retrieve performance are poor) when compared to other implementations. Q also satisfies the same three requirements as P does (Q is a strictly conforming data repository; Q is a conforming data repository that can store and retrieve extensions; Q can generate and interpret data extensions) and has the same conclusion: Q is both a strictly conforming and a conforming data repository.

## 11 Handling extensions

It is commonly misunderstood that extensions can be handled, in general, consistently with desirable behavior. Subclause x.x, Extended Data Elements, contains informative wording on why there is no *generic* support for *generic* extended data elements or extension features, but only *specific* support for *specific* extended data elements (supported to the extent allowed by the data interchange participants). Ignoring unknown data elements is not a general solution to handling extended data elements.

Example: Ignoring an unknown data element might be appropriate behavior (if the data element is an unimportant feature) or might be inappropriate behavior (if the data element is an important feature, such as a security classification).

### 11.1 A notion of "core" sets of data elements

Some vendors, users, institutions, industries, etc., decide they need a "core set" of elements with respect to some standard. The term "core set", typically, means (1) a subset of the original data elements are only required, (2) stronger requirements are made for the "core set" of data elements, (3) optional extended data elements, or (4) some combination. Assuming the following record is defined in a standard:

```
STD: record // definition specified in the standard
(
  A: integer, // optional data element
  B: integer, // optional data element
  C: integer, // optional data element
)
```

The "core set" might only include data elements **B** and **C**, but these data elements would become mandatory.

```
CORE: record // "core set" definition
(
  A: integer, // mandatory data element
  B: integer, // mandatory data element
)
```

Applications that merely conform to the **CORE** specification will not be able to store and process records that conform to the **STD** standard. Example 1: An **STD** record that includes data element **C** will lose that element when stored in a **CORE** repository. Example 2: A conforming **STD** record that does not include the optional data element **B** will be rejected by a **CORE** repository because **B** is mandatory for **CORE**.

Although it may appear that creating a "core set" optimizes the data model, this notion of a "core set" creates significant interoperability problems, which can have a negative impact on industry adoption of a standard.

The following are several approaches and techniques for handling many of the competing needs of extensions and extended data elements.

In other words, if an Institution is creating a "core set", then Institution needs to carefully craft its words to make sure its extended records include those records that conform to the standard records, e.g., do all standard records also conform to Institution's "core set" and will they remain unstripped by the Institution's data repositories? This area of compatibility and interoperability is often confused.

In these three examples, the first example is a reasonable approach for the Institution to extending the standard record, i.e., just adding more elements.

The second example, which addresses "core sets", might be reasonable a reasonable approach towards extending standard records, but has this example still has some hazards that might or might not be worked around.

The third example of extending standard records, which also addresses "core sets", has some significant hazards and would be a "surprise" to most implementers of the standard (vendor, institutions, etc.), i.e., information is stripped from its original source. This approach towards "core sets" causes problems.

Institutions considering extensions and "core sets" should migrate towards the first two example and should migrate away from the third example.

Another way of inquiring about these compatibility issues for extensions is to ask two questions (need YES answers to both questions):

- Will all records that conform to the standard record be acceptable to (conform to) the specification that describes the institution extensions?
- For all records that conform to a standard, when stored in standard repositories, will these records contain at least the same elements from the original record with the original values of the said elements? In other words, for an standard record P that is stored in an institution-specific repository as record Q, is it true, applied recursively, that for each element P there exists the same element in Q with the same value?

The term "core set" should be avoided because it can be confusing and it may mean the wrong (undesirable) thing.

Compatibility issues should be carefully analyzed to make sure that the creators of conforming institution-specific records do not have their information lost or changed.

Creating variants like the third example, **INST\_3**, will cause the splintering of standards.

### Example #1

For example, say the standard records are specified as follows:

```
STD: record // definition specified in the standard
(
  A: integer, // optional data element
  B: integer, // optional data element
  C: integer, // optional data element
)
```

Now let's say that a vendor, institution, user, etc., wants to create their own specialized version of this standard record. This might make sense for a variety of reasons. For vendors, exten-

sions might create additional functionality that might make their product more attractive. For institutions, extensions might support institution-specific features that are not part of the standard. The following might be an institutional extension:

```
INST_1: record // institutional variant #1
(
  A: integer, // optional data element
  B: integer, // optional data element
  C: integer, // optional data element
  D: integer, // optional, institutional extension
)
```

In this institutional extension, standard records might have an additional element **D**. The following are samples of conforming records.

```
// SC = strictly conforming
// C = conforming
// NC = non-conforming
{ A } // SC/C to STD, SC/C to INST_1
{ B } // SC/C to STD, SC/C to INST_1
{ A, B, C } // SC/C to STD, SC/C to INST_1
{ A, D } // C to STD, SC/C to INST_1
{ A, B, C, D } // C to STD, SC/C to INST_1
{ A, B, C, D, E } // C to STD, C to INST_1
```

Note that the last three examples contain extensions to the standard record, so they are only "conforming" (not "strictly conforming"), and the last example has the extension **E** (e.g., a vendor extension) that is both outside the standard records and **INST\_1**'s version of the record. The example above is a typical extension that causes no problems.

Conclusion: This approach is a reasonable institutional extension.

## Example #2

Another possible approach is for institutions (or vendors, etc.) to make certain fields "mandatory" via some notion of a "core set". The main purpose of this might be to recognize that certain repositories have certain useful "keys", e.g., the mandatory fields become database keys which may optimize searches.

Note: The need for certain fields may be a data typing issue, a data quality issue, or both. This discussion only concerns data typing issues and does not discuss data quality issues, which are outside the scope of data extensions. Trying to solve a technical issue (data typing) with a management issue (data quality), or vice versa, is likely to fail. These are separate issues that should be addressed separately.

In this case, the data structure that defines the institutional specification might look like:

```
INST_2: record // institutional variant #2
(
  A: integer, // mandatory data element
  B: integer, // mandatory data element
  C: integer, // optional data element
  D: integer, // optional, institutional extension
)
```

Note that the **INST\_2** specification *invalidates* certain records that conform to standard:

```

// SC = strictly conforming
// C = conforming
// NC = non-conforming
{ A } // SC/C to STD, NC to INST_2
{ B } // SC/C to STD, NC to INST_2
{ A, D } // C to STD, NC to INST_2

```

In other words, those users and vendors that created standard records (and, created them to strictly conform to the standard) now find their records rejected by **INST\_2** because they are non-conforming to **INST\_2** (missing some mandatory elements). A typical workaround to this scenario is that standard records that arrive at this institution are mechanically transformed by supplying nil or dummy fields:

```

// SC = strictly conforming
// C = conforming
// NC = non-conforming
{ A } ==>
{ A, K=nil } // SC/C to STD, SC/C to INST_2
{ B } ==>
{ A=nil, B } // SC/C to STD, SC/C INST_2

```

One area that is missing from many data model standards is the notion of "equivalence" between records, how do I know that record **P** and record **Q** are the same? For example, are the following two records the same?

```

{ J }
{ J, K=nil }

```

These two records might or might not be considered the same.

Conclusion: **INST\_2** might cause some problems because existing records that conform to the standard might be rejected by **INST\_2**. A simple workaround is to create nil/dummy fields, but (1) it is ambiguous as to whether or not these new records are equivalent, and (2) if these records are not equivalent, there might be semantic, legal, and interoperability concerns.

### Example #3

Another third possible approach is for institutions (or vendors, etc.) keep a "core set" of important elements and eliminate "unnecessary" or "unused" elements.

```

INST_3: record // institutional variant #3
(
  A: integer, // mandatory data element
  B: integer, // mandatory data element
  // C is missing because it is not considered "core"
  D: integer, // optional, institutional extension
)

```

This notion of a "core set" is one that trims down the standard record only to its so-called "most important" elements. This approach does conform to original standards. In the following examples:

```

// SC = strictly conforming
// C = conforming
// NC = non-conforming
{ A } // SC/C to STD, SC/C to INST_3
{ B } // SC/C to STD, SC/C to INST_3
{ A, B, C } // SC/C to STD, **C** to INST_3

```

```

{ A, D }           // C to STD, SC/C to INST_3
{ A, B, C, D }    // C to STD, **C** to INST_3

```

The surprise here is that { **A, B, C** } strictly conforms to the standard, but only conforms to the **INST\_3** (a similar problem exists for { **A, B, C, D**}). Stated differently, all systems that conform to the standard must (shall) be able to store { **A, B, C** }. Systems that conform to **INST\_3** specification are *not* required to accept the record { **A, B, C** } or may just strip the record of element **D** — *that is the surprise*.

This is a problem for content developers, publishers, and learning management system developers because (1) they created the data according to the standard, and (2) some of the data was discarded or changed.

Conclusion: Just because an element is "optional" doesn't mean that the storage/retrieval of element is optional. The **INST\_3** approach will definitely cause problems for content and for software that are dependent on these lost/changed fields. This issue arises in:

- conformance testing: may pass **INST\_3** conformance tests, but will fail standard conformance tests
- procurement: need to watch for vendors taking shortcuts
- interoperability: loss/change of information can cause operational/legal problems

The **INST\_3** approach is incompatible with the standard record.

## 11.2 Approaches towards handling common problems

The following are several approaches and techniques for handling many of the competing needs of extensions and extended data elements.

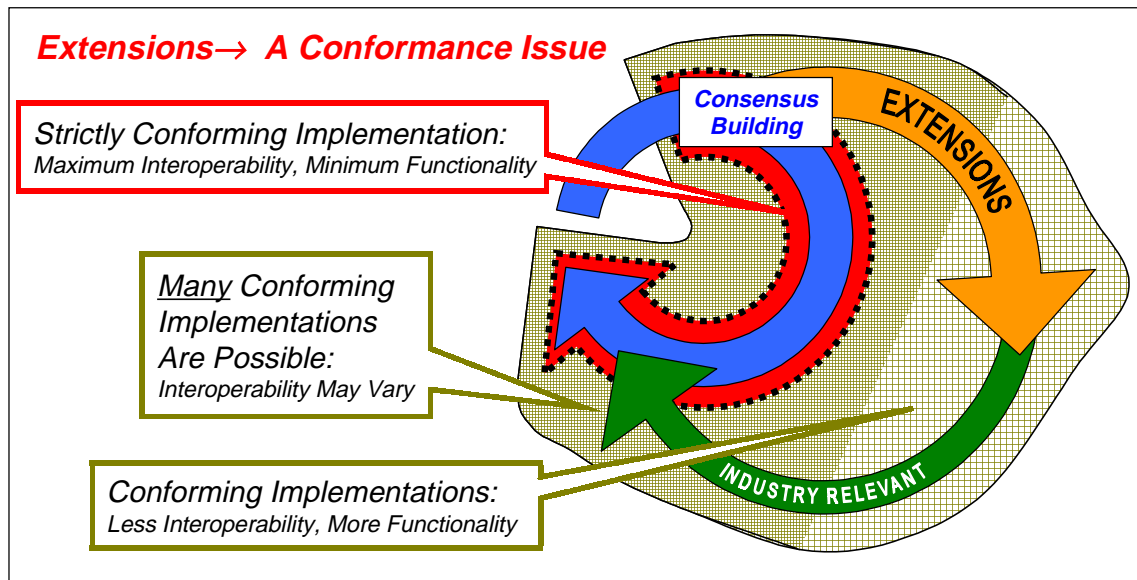
### 11.2.1 Supporting several implementation models

A single type of conformance does not address all technical, business, and interoperability needs. The notions of data instances, data repositories, data readers, and data writers allows implementers to choose the type of conformance claim they desire.

Example 1: Tools that export data will only want to claim conformance as data writers and not be burdened with the requirements of a data repository — *the vendor doesn't want all that functionality and the consumer doesn't need it.*

Example 2: A vendor of a database system might want to claim conformance as a data repository and distinguish their product from other products — *the vendor does want all that functionality and the consumer wants it, too.*

### 11.2.2 Conforming vs. strictly conforming



**Figure 1. Extensions and extended data elements raise conformance issues.**

A standard must balance the needs of interoperability with an industry's need for products that exceed the requirements of a standard. Because products with higher functionality (e.g., exceeding the requirements of a standard) also have lower interoperability (e.g., proprietary extensions), it is not enough to state that "an implementation conforms to the standard".

A strictly conforming implementation has higher interoperability (with respect to the standard), but lower functionality. An implementation that is merely conforming (but not strictly conforming) may have higher functionality but lower interoperability with respect to this Guideline.

Because implementations are labeled "strictly conforming" or merely "conforming", the consumer and the marketplace can make appropriate choices among varying qualities of implementation.

### 11.2.3 Separate phases of translation

The separation of translation phases of the data readers into consumption and interpretation, and data writers into generation and production allow certain bindings to better handle extensions.

In terms of an implementation's behavior, a consistent framework can be described by data generation and data interpretation, yet certain bindings allow the "behavior" to be relaxed by producing less undefined behavior

Example 1: XML coding bindings can ignore unknown tags in data consumption.

Example 2: Other bindings can handle (ignore, diagnose, etc.) extension prefixes, e.g., identifiers that begin with "X-", "x-", or "\_\_" (double underscores).

### 11.2.4 Standards lifecycle for incorporating extensions

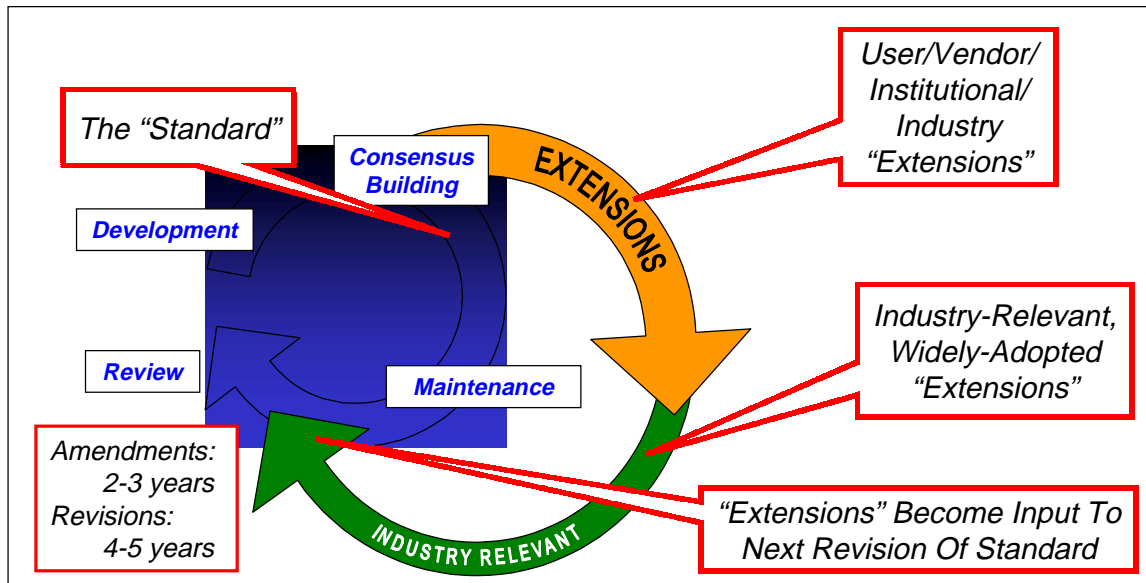


Figure 2. Building standards via increments: amendments and revisions to editions of a standard.

Extensions and extended data elements are techniques that industries use to "try out" new features and "let the market decide". When certain extensions become industry relevant, they may be incorporated into an amendment (typically, 2-3 years after a standard is approved) or into a revision (typically, 4-5 years after a standard is approved).

It is easier to build consensus around features that have been widely adopted than those that have not yet been widely adopted. This incremental approach may help "grow" a standard's data model over time as an industry increasingly adopts improvements into the data model.

## 12 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.

### 12.1 Revision history

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

### 12.2 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.

### 12.3 Resolved issues

The following issues have been resolved:

- None yet.

### 12.4 Open issues

The following issues are outstanding:

- Need to write corresponding PAR.

### 12.5 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)

Frank Farance

2001-02-06

P1484.14.1D1

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