Committee Draft ISO/IEC CD	
Date: **2004-08-06**	Reference number: ISO/JTC 1/SC 32**N1164**
Supersedes document SC 32Nxxx	

THIS DOCUMENT IS STILL UNDER STUDY AND SUBJECT TO CHANGE. IT SHOULD NOT BE USED FOR REFERENCE PURPOSES.

ISO/IEC JTC 1/SC 32 Data Management and Interchange Secretariat: USA (ANSI)	Circulated to P- and O-members, and to technical committees and organizations in liaison for voting (P-members only) by: **2004-11-07** Please return all votes and comments in electronic form directly to the SC 32 Secretariat by the due date indicated.

ISO/IEC CD 20944-20:200x(E)

Title: Information technology — Metadata Interoperability & Bindings (MDIB) Part 20: Common Provisions for Coding bindings

Project: 1.32.17.01.20.00

Introductory note: The attached document is hereby submitted for a three-month letter ballot to the National Bodies of ISO/IEC JTC 1/SC 32. The ballot starts 2004-08-06.

Medium: E

No. of pages: 20

Address Reply to: Douglas Mann, Secretariat, ISO/IEC JTC 1/SC 32, Pacific Northwest National Laboratory, 13667 Legacy Circle Apt H, Herndon, VA, 20171, United States of America

Telephone: +1 202-566-2126; Facsimile; +1 202-566-1639; E-mail: MannD@battelle.org

Reference number of working document: ISO/IEC JTC1 SC32 N1164

Date: 2004-08-04

Reference number of document: ISO/IEC CD1 20944-20
[Release Sequence #7]

Committee identification: ISO/IEC JTC1 SC32 WG2

SC32 Secretariat: US

Information technology — Metadata Interoperability and Bindings (MDIB) — Part 20: Common provisions for coding bindings

Warning

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: International standard
Document subtype: if applicable
Document stage: (30) Committee
Document language: E

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 20944-20 was prepared by Technical Committee ISO/IEC JTC1, *Information Technology*, Subcommittee SC32, *Data Management and Interchange*.

ISO/IEC 20944 consists of the following parts, under the general title *Information technology — Metadata Interoperability and Bindings (MDIB)*:

— *Part 01: Framework*

— *Part 02: Common vocabulary*

— *Part 03: Common provisions for conformance*

— *Part 04: Generic usage*

— *Part 05: Common data structures and services*

— *Part 06: Semi-structured aggregation*

— *Part 20: Common provisions for coding bindings*

— *Part 21: XML coding binding*

— *Part 22: DIVP coding binding*

— *Part 23: ASN.1 coding binding*

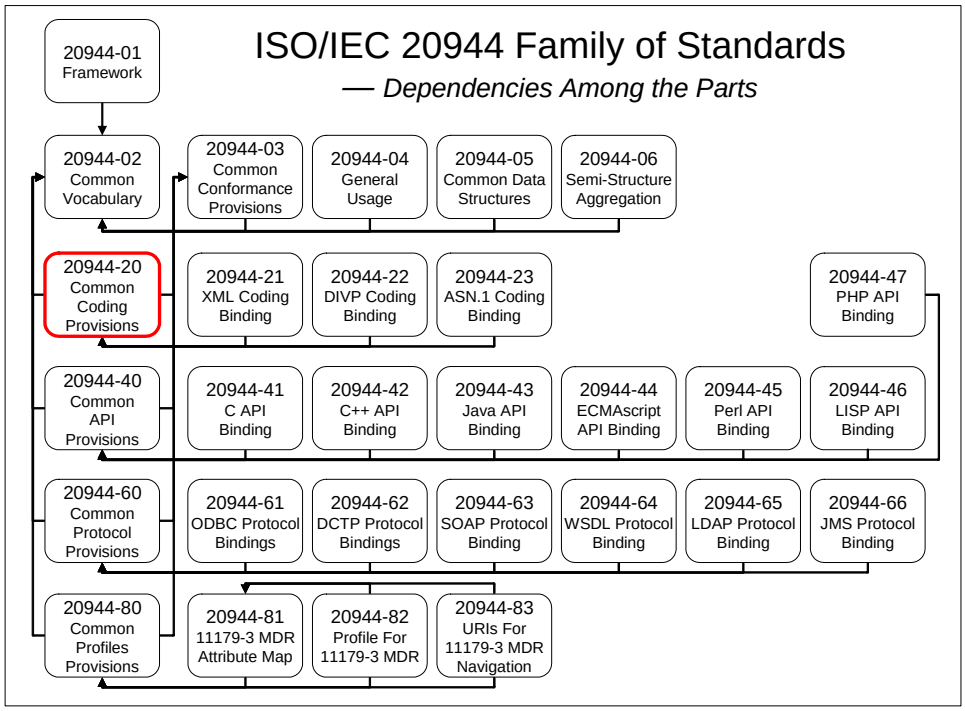
— *Part 40: Common provisions for application programming interface (API) bindings*

— *Part 41: C API binding*

- Part 42: C++ API binding
- Part 43: Java API binding
- Part 44: ECMAscript API binding
- Part 45: Perl binding
- Part 46: LISP binding
- Part 47: PHP binding
- Part 60: Common provisions for protocol bindings
- Part 61: ODBC protocol binding
- Part 62: DCTP protocol binding
- Part 63: SOAP protocol binding
- Part 64: WSDL protocol binding
- Part 65: LDAP protocol binding
- Part 66: JMS protocol binding
- Part 80: Common provisions for profiles
- Part 81: Attribute mapping for 11179-3 metadata registry metamodel
- Part 82: Profile for 11179-3 metadata registry metamodel
- Part 83: Uniform Resource Identifier (URI) suffixes for 11179-3 metadata registry metamodel navigation

Introduction

The following diagram shows the organization of the ISO/IEC 20944 family of standards with this Part highlighted.



Organization of ISO/IEC 20944 family of standards.

This Part of ISO/IEC 20944 concerns provisions that are common to the coding bindings, i.e., Parts 20 to 39. The coding bindings have commonality in their conceptualization of data instances and their internal structures. For example, common features include:

— using datatypes to characterize the nature and operations upon data
— using ISO/IEC 11404 to define and declare datatypes
— using common aggregate structures, such as array and record, to describe sets of data
— using common navigation descriptions to reference components within a set of data

The individual coding bindings (Parts 21 to 39) each incorporate a mapping of common data semantics to their individual binding requirements.

This Part is intended to be normatively referenced by

— the individual coding bindings (i.e., Parts 21 to 39)
— application-specific data models, e.g., using 20944 for their coding, API, and protocol bindings
— coding, API, and protocol bindings for data (metadata) interchange for ISO/IEC 11179-3

Information technology — Metadata Interoperability and Bindings (MDIB) — Part 20: Common provisions for coding bindings

Editor's Note: Each part of 20944 is marked with a common sequence number ("**[Release Sequence #N]**") to indicate they are synchronized and harmonized among themselves. The mark "**[Release Sequence #N]**" does *not* imply that there are a complete set of N-1 prior drafts for any particular Part.

1 Scope

This Part of this International Standard specifies provisions that are common across coding bindings for the 20944 family of standards. This Part is incorporated, via normative reference, into the individual coding bindings.

2 Normative references

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

ISO/IEC 11404:—[1], *Information technology — General Purpose Datatypes (GPD)*

ISO/IEC 20944-02:—[2], *Information technology — Metadata Interoperability and Bindings (MDIB) — Common vocabulary*[3]

3 Terms and definitions

For the purposes of this document, the following terms and definitions given in Part 02 and the following apply[4].

---

[1] To be published, under revision (prior edition: 1996). For current status, see "http://www.dkuug.dk/jtc1/sc22/wg11".

[2] To be published.

[3] The international standards, technical reports, and drafts of the 11179, 20943, and 20944 series are available at

http://metadata-standards.org/11179
http://metadata-standards.org/20943
http://metadata-standards.org/20944

3.1
referenced data interchange specification
data model that is being used for a defined interoperability binding

NOTE The term *referenced data interchange specification*, defined in 20944-02, is used throughout the 20944 family of standards to reference the data model that is being used for the bindings. The *referenced data interchange specification* is tied to the bindings via normative reference, e.g., some other standard defines a data model and uses 20944, via normative reference, to provide some coding, API, or protocol bindings. For Part 82, the *referenced data interchange specification* refers to the 11179-3 metamodel. Part 04 of this International Standard, explains how other standards and specifications may use or re-use portions of the 20944 family of standards.

3.1
coding-independent representation
CIR
representation of data that is independent of storage representation

3.1
coding-specific representation
CSR
representation of data based upon a defined storage representation

4 Functional capabilities

Bindings concern the mapping of one standard (or framework) into another standard (or framework).[5]

Coding bindings concern the mapping of instances of data models to code elements (representations of data).

More than one standard (or framework) may be used to complete the mapping.[6]

The 20944 family of standards uses at least three tiers of mappings. The first tier concerns the main kind of mapping for the binding: coding bindings, API bindings, and protocol bindings. The second tier of mapping concerns the mapping of data model instances to a coding-independent representation (CIR) of data — the ISO/IEC 11404 standard specifies the syntax and semantics of this CIR. The third tier of mapping concerns the mapping of CIR to a coding-specific representation (CSR) — Parts 21 to 39 describe the coding-specific mappings. Additional tiers of mapping are possible, such as specifications of encoding mappings.[7]

The purpose of a common CIR of data is to support common semantics and interoperability among coding bindings and other bindings, such as API and protocol bindings.

---

[4] Users and implementers of this International Standard may find it useful to reference additional terms and definitions from 20944-02.

[5] For example, an ASN.1 binding of "technical specification XYZ" implies mapping the features and requirements of "technical specification XYZ" to the features and capabilities of the ASN.1 standard.

[6] When viewed as a series of layers (e.g., data model = Standard XYZ, coding binding = ASN.1, encoding = ASN.1 Basic Encoding Rules (BER)), bindings may also be viewed as "layered standards" or a "layering of standards".

[7] The XML, ASN.1, and DIVP coding bindings all have additional tiers for encoding. XML has two additional encoding tiers: character encoding (e.g., ASCII vs. UTF-8), and a lower level byte-ordering encoding for multi-octet representations (e.g., little endian and big endian orderings for UTF-16 and UTF-32). ASN.1 has an additional layer of encoding, Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER). DIVP may have encoding rules specified by its outer wrapper, but they are not part of the DIVP coding binding.

EXAMPLE A CIR is developed for a data model. Based upon this CIR, it is possible to transform one coding binding (e.g., XML coding binding) to another coding binding (e.g., ASN.1 coding binding) while sharing common semantics (the CIR) among the coding bindings. Likewise, one coding-specific representation (e.g., XML) can be transformed into another coding-specific representation (e.g., ASN.1).

5 Conceptual model

The coding bindings have commonality in their conceptualization of data instances and their internal structures. For example, common features include:

— using datatypes to characterize the nature and operations upon data
— using ISO/IEC 11404 to define and declare datatypes
— using common aggregate structures, such as array and record, to describe sets of data
— using common navigation descriptions to reference components within a set of data

The individual coding bindings (Parts 021 to 039) each incorporate a mapping of common data semantics to their individual binding requirements.

5.1 Overview of data objects

The conceptual model of data objects is divided into two parts: the data structuring model and the navigation model. The data structuring model describes the "logical" structure of data as it is transferred. The navigation model describes the mapping of the logical structure to a navigation hierarchy.

NOTE The use of hierarchical naming does not imply a hierarchical data model or metadata model.

5.2 Referenced data interchange specification

The 20944 family of standards is organized by an implicit data interchange specification. This data interchange specification is external to the coding, API, and protocol bindings. (See footnote 2.)

5.3 Data structuring model

5.3.1 Data objects

5.3.1.1 General

A unit or collection of data is called an "data object".[8] Data objects are structured data characterized by following.

5.3.1.2 Atomic data objects

An object may be an "atomic data object" if the data object has values which are intrinsically indivisible, e.g., a basic type such as integer, real, character.

EXAMPLE **`17`**, **`-1.7E8`**, **`0D19980831235959`**, **`"hello world"`**.

[8] Not to be confused with "objects" of "object-oriented" analysis, design, and programming.

5.3.1.3 Aggregates

An aggregate is a collection of data objects (atomic or not). Each member of the aggregate is called a "component". The components may be ordered or unordered, named or unnamed, typed or untyped. An aggregate may be nested, i.e., a component of aggregate may be an aggregate itself.

EXAMPLE The list **`{ X: 17 Y: { 18 19 20 } Z: 0D19980831232359 }`** contains three named data elements **`X`**, **`Y`**, and **`Z`**. The first component (**`X`**) is an atomic data object of type integer. The second component (**`Y`**) is an aggregate of three unnamed component. The third component (**`Z`**) is an atomic data object of type date-and-time. The difference between an aggregate and a C programming language structure is: a C structure is always ordered, named, and typed; whereas an aggregate may be ordered, named, and/or typed.

5.3.1.4 Hierarchical organization

The nesting implies a hierarchical organization only from the perspective of data access, not from data implementation. For example, a multi-dimensional array may use the same access method as a nested list. As an analogy, when comparing the C programming language two-dimensional array char x[10][20] to the nested list char *y[10], both types are accessed by the same hierarchical method: x[a][b] and y[a][b].

5.3.1.5 Datatypes

Data objects may have a datatype. The ISO/IEC 11404 General Purpose Datatypes (GPD) Standard is supported. GPD includes basic types (e.g., boolean, integer, real, date-time, string), type parameters (e.g., precision), and type operators (e.g., lists, records, sets).

5.3.2 Properties

5.3.2.1 General

Properties can be associated with data objects. Properties are attributes of the data object.

5.3.2.2 Property lists

Each data object may have an associated property list. A property list is an ordered, named list of data objects.

EXAMPLE The data object **`{ 1 2 3 }`** might have the property list **`{ read_write_access: read_only size: 123 }`**.

5.3.2.3 System and user properties

Some properties are created by the underlying system, e.g., **`_type`**, **`_shape`**, **`_last_modified`**. Some properties are created by the user (or application), e.g., **`read_write_access`**, **`size`**. Designation conventions (e.g., a leading underscore for system properties) help avoid collisions between user/application properties and system properties.

5.3.2.4 Static and dynamic properties

Static properties exist for the duration of the object (unless explicitly removed) and can be listed, e.g., **`_type`**, color. Dynamic properties might not be enumerable and might not exist for the duration of the object.

5.3.2.5 Stored and calculated properties

Some properties have explicit storage associated with the object, e.g., color and (possibly) **_type**. Some properties may be implicit or calculated and have no storage, e.g., **_type** and **_shape**.

5.3.2.6 Memory and volatile properties

Some properties behave the like "memory", i.e., "reading" the property more than once always returns the same value, and "writing" the same value more than once has the same affect as writing it once.

EXAMPLE The properties color and price are "memory"-like properties because they remain the same. The property **_last_modified** is "volatile"-like because the value may change; the property **usage_payment** is "volatile"-like because setting its value more than once (e.g., paying a usage fee of 0.50 USD more than once) might have a different effect than setting the value only once.

5.3.2.7 Properties of the Datatype

Some properties are inherent to the datatype itself, such as "ordered" vs. "unordered". GPD specifies some of these properties. Other properties may exist.

5.4 Designations, identifiers, and navigation

5.4.1 General

Non-atomic objects imply more than one component. Designation methods facilitate navigation of structured data in objects.

5.4.2 Identifiers for navigation

An identifier is a designation that is used for referencing.[9]

5.4.2.1 Designations for indexes

Elements in ordered lists may be accessed by number, e.g., the element numbered **2** of the list **{ 15 16 17 }** is **17**. Elements in lists may be accessed by identifier, e.g., the element number **Z** of the list **{ X: 15 16 Z: 17 }** is **17**. Labeled elements (i.e., elements with an identifier) of ordered lists may be accessed by identifier or number, e.g., the third element of **{ X: 15 16 Z: 17 }** may be accessed by the identifier **Z** or the number **2**.

5.4.2.2 Hierarchical identifiers

Identifiers are hierarchical because the objects they navigate have hierarchical naming.

EXAMPLE The names **C/Z**, **C/2**, **2/Z**, and **2/2** all designate the element **17** in the data object **{ A: 10 B: 11 C: { X: 15 Y: 16 Z: 17 } }**.

[9] As used in this context, the term "identifier" is a designation used for navigation.

5.4.2.3 Merged navigation identifiers

The object, property, link, control, etc., navigation identifiers are merged with the data object navigation identifiers to simplify the number of methods of access, e.g., no need for separate **`getvalue`** and **`getpropery`** operations.

EXAMPLE The property **`Y`** of the object **`X`** is accessed via the name **`X/.Y`**. For the soft link of **`L`** that points to **`D`**: (1) getting the value **`L`** "chases" the link and returns the value of **`D`**, (2) getting the value **`L/`** refers to the link itself, i.e., the reference to **`D`**, not the value of **`D`**, (3) getting the value **`L/.`** chases the link and retrieves **`D`**.

5.4.2.4 Designation conventions

Several designation conventions are used to increase interoperability and reduce integration cost. The user (application) uses external identifiers. The system (implementation) uses internal identifiers. External identifiers are mapped to/from logical identifiers. A logical identifiers is a sequence of pathname separators and pathname segments. Each pathname segment consists of words separated by word separators. Logical identifiers are mapped to/from internal identifiers.

EXAMPLE The external designation conventions use MIME names, while the internal identifiers use C programming language conventions. The external MIME name **`Abc-Def-Ghi/3/Jkl-Mno`** is mapped to the logical pathname: the first path segment is the words **`Abc`**, **`Def`**, and **`Ghi`**; the second path segment is the word **`3`**; the third path segment is the words **`Jkl Mno`**. The logical identifier is mapped to the internal C programming language identifier **`Abc_Def_Ghi[3].Jkl_Mno`** or, possibly, **`Abc_Def_Ghi[3]->Jkl_Mno`**.

5.4.2.5 Hard links

A data object may be accessed by more than one identifier. A hard link is a first class reference to an data object. A data object exists until all its hard links have been removed, then the data object is destroyed.

5.4.2.6 Soft links

An alternate identifier may be created for an data object. A soft link is a second class reference: the link may still exist after the target data object is destroyed. Soft links are implicitly "chased" (automatically "dereferenced").

EXAMPLE If the soft link **`L`** points to **`D`**, then (1) getting the value **`L`** "chases" the link and returns the value of **`D`**, (2) getting the value **`L/`** refers to the link itself, i.e., the reference to **`D`**, not the value of **`D`**, (3) getting the value **`L/.`** chases the link and retrieves **`D`**.

5.4.2.7 Reference

An identifier or pointer to a data object. A reference is a second class name: the link may still exists after the target object is destroyed. References must be explicitly "chased" (or "dereferenced").

5.4.2.8 Views

A view represents a subset of structure data. A simple view might be a subtree of structure data. A complex view may be an SQL select query to describe to subset.

Views describe subsets of information. A view may be used to address the need of creating functional subsets.

6 Semantics

6.1 Datatypes

The 20944 datatypes are based on ISO/IEC 11404.

6.2 Inherent structure

Objects contain data the is navigated by hierarchical navigation identifiers.[10]

EXAMPLE

```
struct
{
    int id;
    char name[100];
    time_t datetime;
};
```

A C programming language structure is represented conceptually as:

```
   Name        Value
+-----------+-------------+
| id        | 11223344    |
+-----------+-------------+
| name      | John Doe    |
+-----------+-------------+
| datetime  | 19980102    |
+-----------+-------------+
```

The same conceptual structure may be used for XML:

```
<NAMEINFO>
    <ID>11223344</ID>
    <NAME>John Doe</NAME>
    <DATETIME>19980102</DATETIME>
</NAMEINFO>
```

6.3 Hierarchical naming

Data is "structured" and accessed via a hierarchical navigation identifier system.

EXAMPLE

```
struct
{
    int id;
```

[10] This does not imply hierarchical data.

```
    struct
    {
        char last[40];
        char first[30];
        char middle[30];
        char title[10];
    } name;
    time_t datetime;
};
```

The nested C structure may be represented conceptually as:

```
   Name        Value
+-----------+-------------+
| id        | 11223344    |
+-----------+-------------+
| name      |last  | Doe  |
+-----------+-------------+
| name      |first | John |
+-----------+-------------+
| name      |middle| Q.   |
+-----------+-------------+
| name      |title | Dr.  |
+-----------+-------------+
| datetime  | 19980102    |
+-----------+-------------+
```

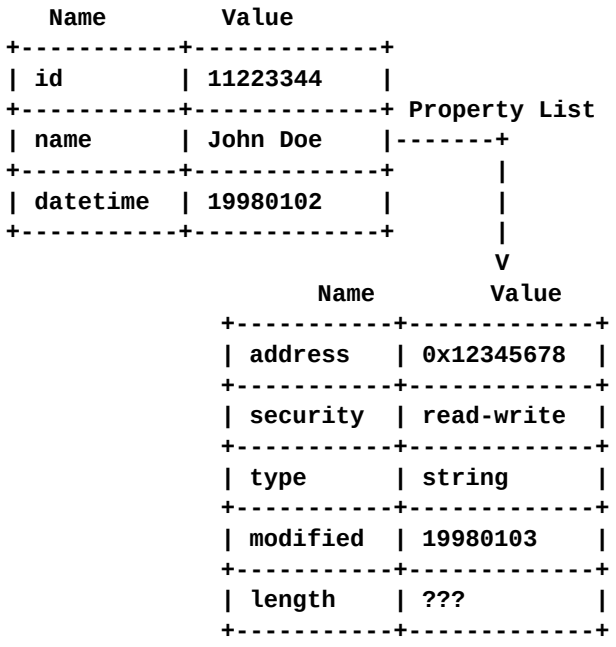
The most important feature is the hierarchical navigation identifier. Using C syntax as an example, **`id`**, **`name.last`**, **`name.first`**, **`name.middle`**, **`name.title`**, **`datetime`**. Note that the naming is the key feature, not the implementation of the structure, so the following is an equivalent conceptual model from a MDIB perspective:

```
   Name        Value
+-----------+-------------+
| id        | 11223344    |
+-----------+-------------+
| name      |     *-------|-------+
+-----------+-------------+       |
| datetime  | 19980102    |       |
+-----------+-------------+       |
                                  V
                       Name       Value
                  +-----------+-------------+
                  | last      | Doe         |
                  +-----------+-------------+
                  | first     | John        |
                  +-----------+-------------+
                  | middle    | Q.          |
                  +-----------+-------------+
                  | title     | Dr.         |
                  +-----------+-------------+
```

The above conceptual data structures could be represented by UNIX filesystems designation convention (e.g., **id**, **name/last**, **name/first**, **name/middle**, **name/title**, **datetime**) or by Windows Registry designation convention (e.g., **id**, **name\last**, **name\first**, **name\middle**, **name\title**, **datetime**).

6.4 Associated properties

Data objects can have properties associated with them. Properties are, conceptually, list of identifier-value pairs. Properties may be system-defined (e.g., pointer address) or user-defined (e.g., security), static (e.g., type) or dynamic (e.g., last modified time), require storage (e.g., security) or calculated on demand (e.g., length).



6.5 Merged navigation identifiers for properties

The navigation identifiers of properties are merged with the navigation identifiers of data objects to simplify access to information, i.e., only a "getvalue" is needed rather than requiring both "get value" and "get property value" operations. Assuming designation conventions of **/** as a pathname separator and **.** as a property introduction, the **_type** property of the data object navigation identifier would be accessed via **name/._type**.

6.6 External, logical, and internal naming conventions

The designation conventions may vary. Conceptually, the "external identifier" is the name used in information interchange, e.g., the navigation identifier used in the "get" service. The "external identifier" is mapped to the "logical identifier". A "logical identifier" is comprised of a list of path name segments, each segment is a list of word segments. The "logical identifier" is mapped to the "internal identifier" within the implementation of the structured data.

Examples of external identifiers with varying designation conventions:

```
Abc-Def/Ghi-Jkl/Mno-Pqr-Stu/123/456    # MIME
abc_def.ghi_jkl.mno_pgr_stu[123][456]  # C language
AbcDef\GhiJkl\MnoPqrStu\123\456        # Windows Registry
```

```
((abc def) (ghi jkl) (mno pqr stu) 123 456)  # LISP
```

The above external names map into the following "logical identifier":

```
path segment #1:
        word #1: abc
        word #2: def
path segment #2:
        word #1: ghi
        word #2: jkl
path segment #3:
        word #1: mno
        word #2: pqr
        word #3: stu
path segment #4:
        word #1: 123
path segment #5:
        word #1: 56
```

The above logical identifier might map into any of the following "internal identifiers"

```
Abc-Def/Ghi-Jkl/Mno-Pqr-Stu/123/456    # MIME
abc_def.ghi_jkl.mno_pgr_stu[123][456]  # C language
AbcDef\GhiJkl\MnoPqrStu\123\456        # Windows Registry
((abc def) (ghi jkl) (mno pqr stu) 123 456)  # LISP
```

The designation conventions for external navigation identifiers are independent of and automatically mapped to the internal designation conventions, e.g., a MIME naming convention might be used externally but a C language convention is used internally.

NOTE The above designation conventions are only examples. The designation conventions are defined via parametric specification.

6.7 The _value property

The **`_value`** property of an data object is identical to the value of the data object, e.g., reading or writing the value of "object" is identical to reading or writing the value of the property **`_value`** associated with the object, i.e., getting **`object`** is the same getting **`object/._value`**.

6.8 Keywords

Keywords are identifiers indicated by the prefix **`"._"`**. Keywords have special meaning, as defined below.

- **`._typeas`**: Returns the value converted to a specific type, e.g., **`"zipcode/._typeas/int"`**.
- **`._value`**: The **`"value"`** property.
- **`._prop`**: A list of properties.
- **`._type`**: The type of the object.
- **`._label`**: The label-only portion of the object.

7 Bindings

This Part describes the common conceptual model and common semantics across all coding bindings. A conforming coding binding shall conform to the requirements described in Clauses 4, 5, 6, 8, and 9 of this Part.

8 Administration

8.1 Use of registry-defined datatypes

A conforming implementation that uses datatypes that are defined by a registry shall conform to those requirements of that (dependent) registry. The data upon which dependent registry updates takes effect shall be implementation-defined.

EXAMPLE For an implementation that uses a data element whose value domain is based upon a registry, that implementation shall document how the expected schedule for applying registry changes to the implementation (e.g., if a value domain is based on ISO 3166-1 country codes and a new country code is added, what is the expected schedule for incorporating the country code into the implementation?).

9 Conformance

9.1 Coding conformance paradigm

The conformance paradigm for 20944 consists of the following conformance roles: data instance, data reader, data writer, data repository.

9.2 Data instance conformance

9.2.1 Data instance

Data instance conformance is independent of binding.

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

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

Conformity assessment of data instances shall be performed by (1) rendering the data instance in ISO/IEC 11404 notation, and (2) verifying the requirements described by the referenced data interchange specification.

9.2.2 Bound data instance

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

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

NOTE The difference between a SC/C data instance, a SC/C coding, and a SC/C bound data instance is: (1) a data instance 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 bound 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 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.

9.3 Data application conformance

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

A strictly conforming data application is a data application that strictly conforms to the referenced data interchange specification.

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

A conforming data applications is a data application that conforms to the referenced data interchange specification.

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

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

9.3.1 Data reader

A strictly conforming data reader shall interpret data that strictly conforms to (1) the referenced data interchange specification, and (2) at least one binding of the referenced data interchange specification.

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

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

A conforming data reader shall interpret data that conforms to (1) the referenced data interchange specification, and (2) at least one binding of the referenced data interchange specification.

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

9.3.2 Data writer

A strictly conforming data writer shall generate data that strictly conforms to (1) the referenced data interchange specification, and (2) at least one binding of the referenced data interchange specification.

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

A conforming data writer shall generate data that conforms to (1) the referenced data interchange specification, and (2) at least one binding of the referenced data interchange specification.

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

9.3.3 Data repository

A strictly conforming data repository shall:

— receive data instances for subsequent retrieval;
— use strictly conforming data interpretation for receiving data instances;
— store data instances in persistent storage so that data extensions may not persist;
— send, on request, previously stored data instances;
— use strictly conforming data generation for sending data instances;
— strictly conform to at least one coding binding; and
— strictly conform to at least one API binding or 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 Annex A for more information on the storage of extensions in strictly conforming data repositories.

A conforming data repository shall:

— receive data objects for subsequent retrieval;
— use conforming data interpretation for receiving data instances;
— store data instances in persistent storage so that data extensions may persist;
— send, on request, previously stored data objects;
— use conforming data generation for sending data instances;
— conform to at least one coding binding; and
— conform to at least one API binding or 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 instances.