INTERNATIONAL STANDARD

ISO/IEC 20944-2

First edition
2013-01-15

Information technology — Metadata Registries Interoperability and Bindings (MDR-IB) —

Part 2: Coding bindings

Technologies de l'information — Interopérabilité et liaisons des registres de métadonnées (MDR-IB) —
Partie 2: Liaisons de codage

Reference number
ISO/IEC 20944-2:2013(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national 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 and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 20944-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

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

— *Part 1: Framework, common vocabulary, and common provisions for conformance*

— *Part 2: Coding bindings*

— *Part 3: API bindings*

— *Part 4: Protocol bindings*

— *Part 5: Profiles*

Introduction

The ISO/IEC 20944 series of International Standards provides the bindings and their interoperability for metadata registries, such as those specified in the ISO/IEC 11179 series of International Standards.

This part of ISO/IEC 20944 contains provisions that are common to coding bindings (Clauses 4-10) and the coding bindings themselves (Clause 11 onward). 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 each incorporate a mapping of common data semantics to their individual binding requirements.

Information technology — Metadata Registries Interoperability and Bindings (MDR-IB) —

Part 2:
Coding bindings

1 Scope

The ISO/IEC 20944 series of International Standards describes codings, application programming interfaces (APIs), and protocols for interacting with an ISO/IEC 11179 metadata registry (MDR).

This part of ISO/IEC 20944 specifies provisions that are common across coding bindings for the ISO/IEC 20944 series. This part of ISO/IEC 20944 includes the individual coding bindings themselves.

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 11404:2007, *Information technology — General-Purpose Datatypes (GPD)*

ISO/IEC 20944-1:2013, *Information technology — Metadata Registries Interoperability and Bindings (MDR-IB) — Framework, common vocabulary, and common provisions for conformance*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 20944-1 apply.

4 Intended use of this part of ISO/IEC 20944

Bindings concern the mapping of one standard (or framework) into another standard (or framework).[1]

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.[2]

The ISO/IEC 20944 series of International 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

[1] 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.

[2] 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".

second tier of mapping concerns the mapping of data model instances to a coding-independent representation (CIR) of data — ISO/IEC 11404 specifies the syntax and semantics of this CIR. The third tier of mapping concerns the mapping of CIR to a coding-specific representation (CSR) — Clauses 11 and onward describe the coding-specific mappings. Additional tiers of mapping are possible, such as specifications of encoding mappings.[3]

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.

NOTE XML bindings are useful for integration with other XML technologies. The XML binding provides the requirements of XML interchange without specifying the XML technologies (e.g., XML Schema) to implement the features. DIVP bindings are useful for integration with name-value pair technologies, such as scripting systems, E-mail, and web servers.

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 Abstract 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 (Clauses 11 and onward) 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 ISO/IEC 20944 series of International 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.)

[3] 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.

5.3 Data structuring model

5.3.1 Data objects

5.3.1.1 General

A unit or collection of data is called a "data object".[4] Data objects are structured data characterized by the 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"`.

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. he 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. ISO/IEC 11404 includes basic types (e.g., boolean, integer, real, date-time, string), type parameters (e.g., precision), and type operators (e.g., lists, records, sets).

Implementations shall support the ISO/IEC 11404 datatypes [5] . Additional supported datatypes are implementation-defined.

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 }`.

---

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

5 ISO/IEC 11404 includes a description of datatype conformity.

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.[6] The meaning of the navigation is defined by the URI syntax, as overlaid with indexes and hierarchies.

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`**. Index numbers begin at zero.

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

5.4.2.2 Hierarchical identifiers

Identifiers are hierarchical because the objects they navigate have hierarchical naming. These hierarchical naming conventions, as well as all other forms of external data identifiers that permits hierarchical navigation are also called hierarchical navigation paths.

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 } }`.

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 ISO/IEC 20944 datatypes are based upon ISO/IEC 11404.

6.2 Inherent structure

Objects contain data that is navigated by hierarchical navigation identifiers.[7]

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

[7] This does not imply hierarchical data.