INTERNATIONAL STANDARD

ISO 19118

Second edition
2011-10-15

Geographic information — Encoding

Information géographique — Codage

Reference number
ISO 19118:2011(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 19118 was prepared by Technical Committee ISO/TC 211, *Geographic information/Geomatics*.

This second edition cancels and replaces the first edition (ISO 19118:2005), which has been technically revised.

Introduction

This International Standard specifies the requirements for defining encoding rules used for interchange of geographic data within the set of International Standards known as the “ISO 19100 series”. An encoding rule allows geographic information defined by application schemas and standardized schemas to be coded into a system-independent data structure suitable for transport and storage. The encoding rule specifies the types of data being coded and the syntax, structure and coding schemes used in the resulting data structure. The resulting data structure can be stored on digital media or transferred using transfer protocols. It is intended that the data be read and interpreted by computers, but data can be in a form that is human readable.

The choice of one encoding rule for application-independent data interchange does not exclude application domains and individual nations from defining and using their own encoding rules that can be platform dependent or more effective with regard to data size or processing complexity. XML is a subset of ISO/IEC 8879 and has been chosen because it is independent of computing platform and interoperable with the World Wide Web.

This International Standard is divided into three logical sections. The requirements for creating encoding rules based on UML schemas are specified in Clauses 6 to 9. The requirements for creating encoding service are specified in Clause 10, and the requirements for XML-based encoding rules are specified in Annex A.

The XML-based encoding rule is intended for use as a neutral data interchange. It relies on the Extensible Markup Language (XML) and the ISO/IEC 10646 character set standards.

The geographic information standards are organized within the set of International Standards known as the “ISO 19100 series”. The background and the overall structure of this series of International Standards and the fundamental description techniques are defined in ISO 19101, ISO/TS 19103 and ISO/TS 19104.

Users of this International Standard can develop application schemas to formally describe geographic information. An application schema is compiled by integrating elements from other standardized conceptual schemas (e.g. ISO 19107). How this integration takes place is described in ISO 19109. The set of International Standards known as the “ISO 19100 series” also defines a set of common services that are available when developing geographic information applications. The common services are generally defined in ISO 19119 and cover access to, and processing of, geographic information according to the common information model. This International Standard covers implementation issues.

Geographic information — Encoding

1 Scope

This International Standard specifies the requirements for defining encoding rules for use for the interchange of data that conform to the geographic information in the set of International Standards known as the “ISO 19100 series”.

This International Standard specifies

— requirements for creating encoding rules based on UML schemas,

— requirements for creating encoding services, and

— requirements for XML-based encoding rules for neutral interchange of data.

This International Standard does not specify any digital media, does not define any transfer services or transfer protocols, nor does it specify how to encode inline large images.

2 Conformance

2.1 Introduction

Two sets of conformance classes are defined for this International Standard.

2.2 Conformance classes related to encoding rules

All encoding rules shall pass all test cases of the abstract test suite in B.1. All encoding rules shall pass all test cases of the abstract test suite in B.2 and/or B.3.

Table 1 — Conformance classes related to encoding rules

Conformance class	Subclause of the abstract test suite
All encoding rules	B.1
Encoding rule with instance conversion	B.2
Encoding rule with schema conversion	B.3

2.3 Conformance classes related to encoding services

All encoding services shall pass all test cases of the abstract test suite in B.4. Depending on the capabilities of the encoding service, it shall pass all test cases of additional conformance classes in accordance with Table 2.

Table 2 — Conformance classes related to encoding services

Conformance class	Subclause of the abstract test suite
All encoding services	B.4
Generic encoding service	B.5
Service that encodes data	B.6
Service that decodes data	B.7
Service that generates an output data structure schema	B.8

3 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 8601:2004, *Data elements and interchange formats — Information interchange — Representation of dates and times*

ISO/IEC 10646:2011, *Information technology — Universal Coded Character Set (UCS)*

ISO/TS 19103:2005, *Geographic information — Conceptual schema language*

ISO 19109:2005, *Geographic information — Rules for application schema*

Extensible Markup Language (XML) 1.0, W3C Recommendation. Available at <http://www.w3.org/TR/REC-xml>

4 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

4.1
application schema
conceptual schema (4.5) for **data** (4.8) required by one or more applications

[ISO 19101:2002, 4.2]

NOTE An application schema describes the content, the structure and the constraints applicable to **information** (4.22) in a specific application domain.

4.2
character
member of a set of elements that is used for the representation, organization, or control of **data** (4.8)

[ISO/IEC 2382-1:1993, 01.02.11]

4.3
code
representation of a label according to a specified scheme

4.4
conceptual model
model (4.27) that defines concepts of a **universe of discourse** (4.33)

[ISO 19101:2002, 4.4]

4.5
conceptual schema
formal description of a **conceptual model** (4.4)

[ISO 19101:2002, 4.5]

4.6
conceptual schema language
formal language based on a conceptual formalism for the purpose of representing **conceptual schemas** (4.5)

[ISO 19101:2002, 4.6]

EXAMPLES UML, EXPRESS, IDEF1X.

NOTE A conceptual schema language may be lexical or graphical.

4.7
conversion rule
rule for converting instances in the input **data** (4.8) structure to instances in the output data structure

4.8
data
reinterpretable representation of **information** (4.22) in a formalized manner suitable for communication, interpretation, or processing

[ISO/IEC 2382-1:1993, 01.01.02]

4.9
data interchange
delivery, receipt and interpretation of **data** (4.8)

4.10
data transfer
movement of **data** (4.8) from one point to another over a **medium** (4.26)

NOTE Transfer of **information** (4.22) implies transfer of data.

4.11
data type
specification of a **value domain** (4.34) with operations allowed on values in this domain

[ISO/TS 19103:2005, 4.1.5]

EXAMPLES Integer, Real, Boolean, String and Date.

NOTE A data type is identified by a term, e.g. Integer. Values of the data types are of the specified value domain, e.g. all integer numbers between –65537 and 65536. The set of operations can be +, -, * and / and is semantically well defined. A data type can be simple or complex. A simple data type defines a value domain where values are considered atomic in a certain context, e.g. Integer. A complex data type is a collection of data types that are grouped together. A complex data type may represent an object and can, thus, have identity.

4.12
dataset
identifiable collection of **data** (4.8)

[ISO 19115:2003, 4.2]

4.13
encoding
conversion of **data** (4.8) into a series of **codes** (4.3)

4.14
encoding rule
identifiable collection of **conversion rules** (4.7) that define the **encoding** (4.13) for a particular **data** (4.8) structure

EXAMPLES XML, ISO 10303-21, ISO/IEC 8211.

NOTE An encoding rule specifies the types of data being converted as well as the syntax, structure and **codes** (4.3) used in the resulting data structure.

4.15
encoding service
software component that has an **encoding rule** (4.14) implemented

4.16
feature
abstraction of real world phenomena

[ISO 19101:2002, 4.11]

NOTE A feature may occur as a type or an instance. Feature type or feature instance is used when only one is meant.

4.17
file
named set of records stored or processed as a unit

[ISO/IEC 2382-1:1993, 01.08.06]

4.18
geographic data
data (4.8) with implicit or explicit reference to a location relative to the Earth

[ISO 19109:2005, 4.12]

4.19
geographic information
information (4.22) concerning phenomena implicitly or explicitly associated with a location relative to the Earth

[ISO 19101:2002, 4.16]

4.20
identifier
linguistically independent sequence of **characters** (4.2) capable of uniquely and permanently identifying that with which it is associated

[ISO 19135:2005, 4.1.5]

4.21
identification convention
set of rules for creating **identifiers** (4.20)

4.22
information
knowledge concerning objects, such as facts, events, things, processes, or ideas, including concepts, that within a certain context has a particular meaning

[ISO/IEC 2382-1:1993, 01.01.01]

4.23
instance model
representation **model** (4.27) for storing **data** (4.8) according to an **application schema** (4.1)

4.24
interface
⟨UML⟩ named set of operations that characterize the behaviour of an element

[ISO/IEC 19501]

4.25
interoperability
capability to communicate, execute programs, or transfer **data** (4.8) among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units

[ISO/IEC 2382-1:1993, 01.01.47]

4.26
medium
substance or agency for storing or transmitting **data** (4.8)

EXAMPLES Compact disc, internet[1], radio waves, etc.

4.27
model
abstraction of some aspects of reality

[ISO 19109:2005, 4.14]

4.28
schema
formal description of a **model** (4.27)

[ISO 19101:2002, 4.25]

4.29
schema model
representation **model** (4.27) for storing **schemas** (4.28)

EXAMPLE Representation model for a schema repository.

4.30
stereotype
⟨UML⟩ new type of modelling element that extends the semantics of the metamodel

[ISO/IEC 19501]

NOTE It is necessary that stereotypes be based on certain existing types or classes in the metamodel. Stereotypes may extend the semantics, but not the structure, of pre-existing types and classes. Certain stereotypes are predefined in the UML, others may be user-defined. Stereotypes are one of three extensibility mechanisms in UML; the others are constraint and tagged value.

4.31
transfer protocol
common set of rules for defining interactions between distributed systems

4.32
transfer unit
collection of **data** (4.8) for the purpose of a **data transfer** (4.10)

NOTE A transfer unit does not have to be identifiable like a **dataset** (4.12).

4.33
universe of discourse
view of the real or hypothetical world that includes everything of interest

[ISO 19101:2002, 4.29]

4.34
value domain
set of accepted values

[ISO/TS 19103:2005, 4.1.15]

EXAMPLE The range 3-28, all integers, any character, enumeration of all accepted values (green, blue, white).

5 Symbols and abbreviated terms

DCE	Distributed computing environment
DUID	Domain unique identifier
HTML	Hypertext markup language
MODIS	Moderate resolution imaging spectroradiometer
POSC	Petroleum Open Standards Consortium
TIFF	Tagged image file format
UCS	Universal multiple-octet coded character set
UML	Unified modelling language
UTF	UCS Transfer format
UUID	Universally unique identifier
XML	Extensible markup language

6 Fundamental concepts and assumptions

6.1 Concepts

The purpose of the set of International Standards known as the “ISO 19100 series” is to enable interoperability between heterogeneous geographic information systems. To achieve interoperability between heterogeneous systems, it is necessary to determine two fundamental issues. The first issue is to define the semantics of the content and the logical structures of geographic data. This shall be done in an application schema. The second issue is to define a system- and platform-independent data structure that can represent data corresponding to the application schema.

The fundamental concepts of data interchange, i.e. the procedure based on the application schema for encoding, delivery, receipt and interpretation of geographic data, are described in 6.2 to 6.6. An overview of the data interchange process is described in 6.2; 6.3 introduces application schemas that allow interpretation of geographic data; 6.4 describes the importance of the encoding rule for producing system-independent data structures; 6.5 describes a software component, called the encoding service, for executing the encoding rule; and 6.6 describes the procedure for delivery and receipt, called the transfer service.

6.2 Data interchange

An overview of a data interchange is shown in Figure 1. System A wants to send a dataset to system B. To ensure a successful interchange, it is necessary that A and B decide on three things: i.e. a common application schema **I**, which encoding rule R to apply, and what kind of transfer protocol to use. The application schema is the basis of a successful data transfer and defines the possible content and structure of the transferred data, whereas the encoding rule defines the conversion rules for how to code the data into a system-independent data structure.

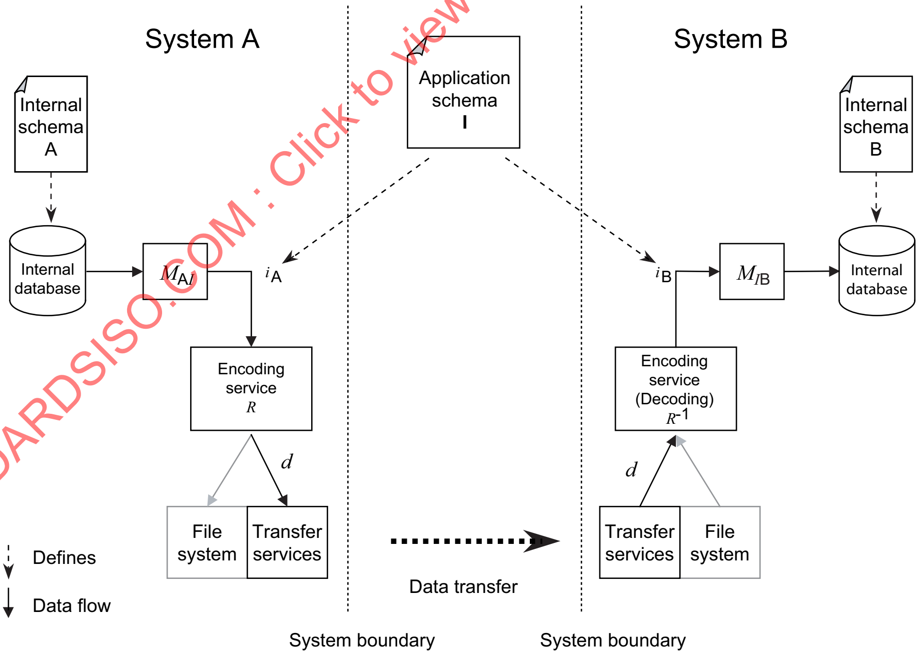


Figure 1 — Overview of data interchange between two systems

Both systems, A and B, store data in an internal database according to an internal schema, but the schemas are usually different, i.e. schema A is not equal to schema B. It is necessary to take the following logical steps in order to transfer a dataset from A's internal database to B's internal database.

a) The first step for system A is to translate its internal data into a data structure that is in accordance with the common application schema I. Here, this is done by defining a mapping from the concepts of the internal schema to the concepts defined in the application schema and by writing appropriate mapping software to translate the data instances. In Figure 1, this mapping is denoted as $M_{\mathrm{A}I}$. The result is an application-schema-specific data structure, i_{A}. The data structure is stored in memory or on an intermediate file and is system-dependent and, thus, is not suitable for transfer.

b) The next step is to use an encoding service, which applies the encoding rule R to create a data structure that is system independent and, therefore, suitable for transfer. This encoded dataset is called d and may be stored in a file system or transferred using a transfer service.

c) System A then invokes a transfer service to send the encoded dataset d to system B. The transfer service follows a transfer protocol for how to do packaging and how the actual transportation over an on-line or off-line communication medium should take place. It is necessary that both parties agree on the transfer protocol used.

d) The transfer service on system B receives the transferred dataset, and according to the protocol the dataset is unpacked and stored as an encoded dataset d, e.g. on an intermediate file.

e) In order to get an application-schema-specific data structure i_{B}, system B applies the inverse encoding rule R^{-1} to interpret the encoded data.

f) To use the dataset, it is necessary that B translate the application-schema-specific data structure i_{B} into its internal database. This is done by defining a mapping from the application schema into its internal schema and by writing software that does the actual translation. In Figure 1 this mapping is denoted $M_{I\mathrm{B}}$.

This International Standard specifies only the requirements for creating encoding rules and the encoding services and not the whole data interchange process. Thus, only steps b) and e) are standardized. Steps a), c), d) and f) use general information technology services.

6.3 Application schema

An application schema is a conceptual schema for applications with similar data requirements. The application schema is the basis of a successful data interchange and defines the possible content and structure of the data. It is also the basis for implementing application-schema-specific data structures for local storage of data.

The application schema used for encoding in compliance with this International Standard shall be written in the UML conceptual schema language, in accordance with ISO/TS 19103 and ISO 19109. These International Standards specify a framework for how to write application schemas. The rules include specifications on how to use standardized schemas to define feature types. It is necessary that both a sender and a receiver of data have access to the application schema.

The application schema shall be accessible to both ends of a data interchange to ensure a successful result. It is necessary that the application schema be transferred before data interchange takes place, so that both the receiver and sender can prepare their systems by implementing mappings and data structures according to the application schema. It may be transferred together with the dataset, or it may be stored in a public place and referenced from the dataset.

The application schema may be interchanged by paper- or electronic-based methods.

6.4 Encoding rule

6.4.1 Concept

An encoding rule is an identifiable collection of conversion rules that defines the encoding for a particular data structure. The encoding rule specifies the data types being converted, as well as the syntax, structure and coding schemes used in the resulting data structure. An encoding rule is applied to application-schema-specific data structures to produce system-independent data structures suitable for transport or storage. In order to define an encoding rule, it is necessary that three important aspects be specified: the input data structure, the output data structure and the conversion rules between the elements of the input and the output data structures. Both the input and output data structures are written using a conceptual schema language and the concepts in the languages are used to define the encoding rule.

6.4.2 Input data structure

The input data structure is an application-schema-specific data structure. The data structure can be thought of as a set of data instances, i.e. $\mathbf{i} = \{i_1, ..., i_p\}$; see Figure 1. Each data instance, i_k, is an instance of a concept, I_l, defined in an application schema. The application schema defines a set of concepts defined in the application schema $\mathbf{I} = \{I_1, ..., I_m\}$.

The application schema is a conceptual schema, **c**, written in a conceptual schema language, **C**. The conceptual schema defines a set of concepts $\mathbf{c} = \{c_1, ..., c_m\}$ by instantiating the concepts of the conceptual schema language $\mathbf{C} = \{C_1, ..., C_r\}$. Since the application schema is a conceptual schema, $\mathbf{c} = \mathbf{I}$.

6.4.3 Output data structure

The output data structure is defined by a schema, $\mathbf{D} = \{D_1, ..., D_s\}$. D is the schema for the output structure and is not shown in Figure 1. The output data structure can be thought of as a set of data instances, i.e. $\mathbf{d} = \{d_1, ..., d_q\}$ where each data instance, d_k, is an instance of a concept, D_l.

The schema, **D**, defines the syntax, structure and coding schemes of the output data structure.

6.4.4 Conversion rules

A conversion rule specifies how a data instance in the input data structure shall be converted to zero, one, or more instances in the output data structure. The conversion rules are defined and based on the concepts of the conceptual schema language, **C**, and on the concepts of the output data structure schema, **D**. It is necessary to specify a conversion rule, R_i, for each of the legal combinations of concepts in the conceptual schema language. The set of conversion rules are $\mathbf{R} = \{R_1, ..., R_n\}$, where R_i is the i-th conversion rule and C_i is the i-th legal combination of instances from the schema language. A conversion table for all possible C_i can be set up, where each C_i maps to a production of instances in the output data structure, **D**. Figure 2 shows the relationship between the input and output conceptual schema language and the encoding rule.

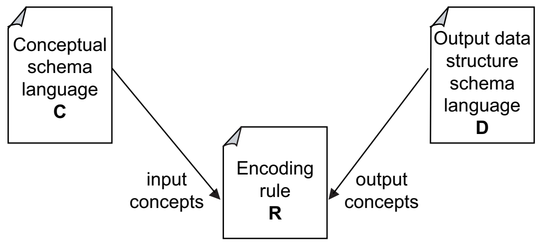

Conceptual schema language C
Output data structure schema language D
input concepts
Encoding rule R
output concepts

Figure 2 — The encoding rule defines conversion rules from input concepts to output concepts

NOTE The conversion rules are defined based on the two schema languages and not on any particular application schema. This is a generic approach that allows developers to write application-schema-independent encoding services, which can be used for different application schemas as long as the schemas are defined in the same conceptual schema language.

6.5 Encoding service

An encoding service is a software component that has implemented the encoding rule and provides an interface to encoding and decoding functionality. It is an integrated part of data interchange.

Figure 3 presents the details of an encoding service and its relationships to important specification schemas. The encoding service shall be able to read the input data structure and convert the instances to an output data structure and *vice versa*. It shall also be able to read the application schema declarations and write the corresponding output data structure schema. The input data structure is defined by an application schema. The application schema is defined using concepts of the conceptual schema language. The output data structure is also described with a schema, called the data structure schema, which defines the possible content, structure and coding schemes of the output data structure. The data structure schema is described with a schema language. The encoding rule specifies conversion rules at two levels: the first is at the schema level and the second is at the instance level. At the schema level, the conversion rules define a mapping for each of the concepts defined in the application schema to corresponding concepts in the data structure schema. At the instance level, the conversion rules define a mapping for each of the instances in the input data structure to corresponding instances in the output data structure. The instance conversion rules are normally deduced from the schema conversion rules.

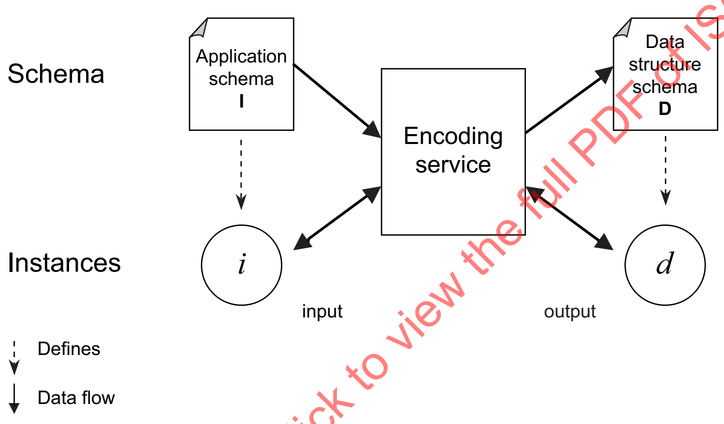


Figure 3 — Overview of the encoding process

An encoding service shall at least provide interfaces for encoding and decoding functionality. Examples of such interfaces are for encoding **d** = encode (**i, I**) and for decoding **i** = decode (**d, I**). Here, **i** is a reference to an application-schema-specific data structure; **I** is a reference to the application schema; and **d** is a reference to the system independent data structure.

6.6 Transfer service

A transfer service is a software component that has implemented one or more transfer protocols that allows data transfer between distributed information systems over off-line or on-line communication media. To successfully transfer data between two systems, it is necessary that the sender and receiver agree on the transfer protocol being used.

Different transfer protocols can be defined. One example is off-line transfer protocols where data are stored on optical or magnetic media and delivered using postal services or other dedicated delivery services. Another example is on-line transfer protocols where data are compressed and included as an email attachment, delivered using a file transfer protocol or transferred using other distributed information technology services which rely on an underlying network service.

This International Standard does not prescribe any preferred transfer protocols.

7 Character repertoire

ISO/IEC 10646 defines an internationally recognized repertoire of characters called the Universal Character Set (UCS) and its character-encoding schemes. The international character set standards defined in ISO/IEC 10646 shall be used in implementing this International Standard.

The character-encoding schemes that can be supported by international profiles of this International Standard are the following:

a) 8-bit variable size UCS Transfer Format UTF-8;

b) 16-bit variable size UCS Transfer Format UTF-16;

c) 16-bit fixed size Universal Character Set UCS-2 (deprecated);

d) 32-bit fixed size Universal Character Set UCS-4.

International encoding rules that claim conformance with this International Standard shall support one or more of these character-encoding schemes. Within national profiles and system implementations, different character-encoding schemes may be used. The fixed-size character-encoding schemes are often used in database implementations and the variable-size is often used for data interchange purposes.

ISO/IEC 10646 specifies only the repertoire of characters and gives no indication of which language is actually used.

NOTE 1 In cases where it is important to distinguish between different languages in text strings, special mechanisms to indicate the language used can be used.

ISO/IEC 10646 defines mechanisms for creating composite characters. Composite characters are characters produced by superimposing one or more additional characters on a base character. ISO/IEC 10646 defines a set of precomposed characters and their defined decomposition. Since mixing composite characters with their precomposed equivalents can lead to interpretation problems, the use of a composite character if a precomposed character exists is deprecated, i.e. the precomposed character shall always be used.

To summarize, an encoding rule shall

— support one or more character-encoding schemes, and

— not use composite characters if equivalent precomposed characters exist.

EXAMPLE The precomposed character ö has the defined decomposition o¨.

NOTE 2 For a more detailed description of character normalization, see http://www.unicode.org/reports/tr15/ and http://www.w3.org/TR/charmod-norm/.

NOTE 3 UTF-16, UCS-2 and UCS-4 require information on how to deal with byte ordering, see http://www.unicode.org/faq/utf_bom.html.

8 Generic instance model

8.1 Introduction

A generic instance model is defined in Clause 8. The instance model is a convenient common representation of data when developing encoding services. The instance model is capable of representing data described by application schemas expressed in UML. The instance model represents the application-schema-specific data structure defined in Clause 6 (data structures i_A and i_B in Figure 1). The instance model consists of a dataset (IM_Dataset) that contains a sequence of objects (IM_Object), where an object consists of a sequence of properties (IM_Property). Properties in this context are either attributes or associations; operations are not included in the general instance model. Each property is encoded according to its data type. The instance model is shown in Figures 4 and 5.

The application schema defines a number of classes and their attributes and associations and it is the basis for generating data representations. A data representation (dataset) contains one or more objects that are structured and encoded according to their class definitions. Clause 8 describes the principles of how to represent objects, their attributes and associations between objects.

The basic unit of information in a dataset is the object. An object shall be an instance of a single concrete class. There are no instances of abstract classes and classes stereotyped as interface. Thus, properties defined by such classes are encoded as part of concrete classes inheriting or realizing them. Each class shall have a unique name within the application schema. The application schema may refer to or use classes defined in standardized schemas or other application schemas. The declaration of these classes shall either be included in the UML model that contains the application schema or accompany the application schema as a separate file.

An object shall contain a set of property values. The object's class defines the properties and they can either be inherited through the “class” supertypes or defined within the class itself. In order to differentiate between the different properties, each property shall have a name that is unique within its class. The property's data type governs the possible values and the multiplicity statement indicates the number of instances of the attribute in an instantiated object.

An object has a corresponding class, defined in an application schema or standardized schema, which defines the possible attributes and associations that are necessary to represent the state of the object. An IM_Object refers to its class by the “class” attribute, it shall be identified within the context of a dataset by its unique identifier “id”, and may be universally uniquely identified within a defined universe, application domain or name space, by its “duid” attribute.

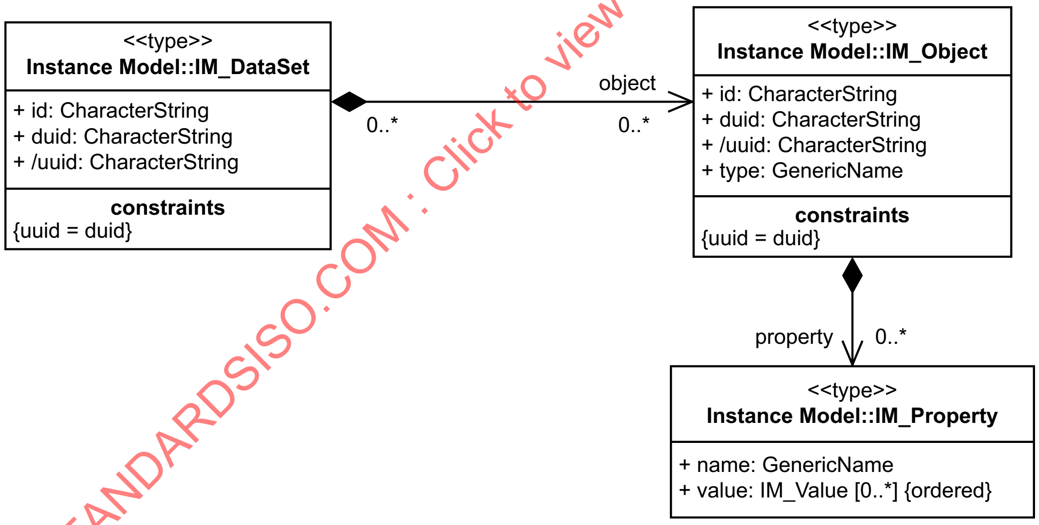


Figure 4 — Instance model — Dataset, object and property

The attributes defined by the class and the association ends navigable from the class are mapped to a set of properties. A property (IM_Property) represents a name with an ordered collection of values. It can represent an attribute or association end. The property name shall correspond to the attribute name or the target role name of an association. A value (IM_Value) represents a property value.

Null values may be given either explicitly or implicitly. An explicit null value shall be indicated by an instance of the corresponding IM_Property with a given nilReason value. An implicit null value is indicated if the corresponding IM_Property instance is missing.

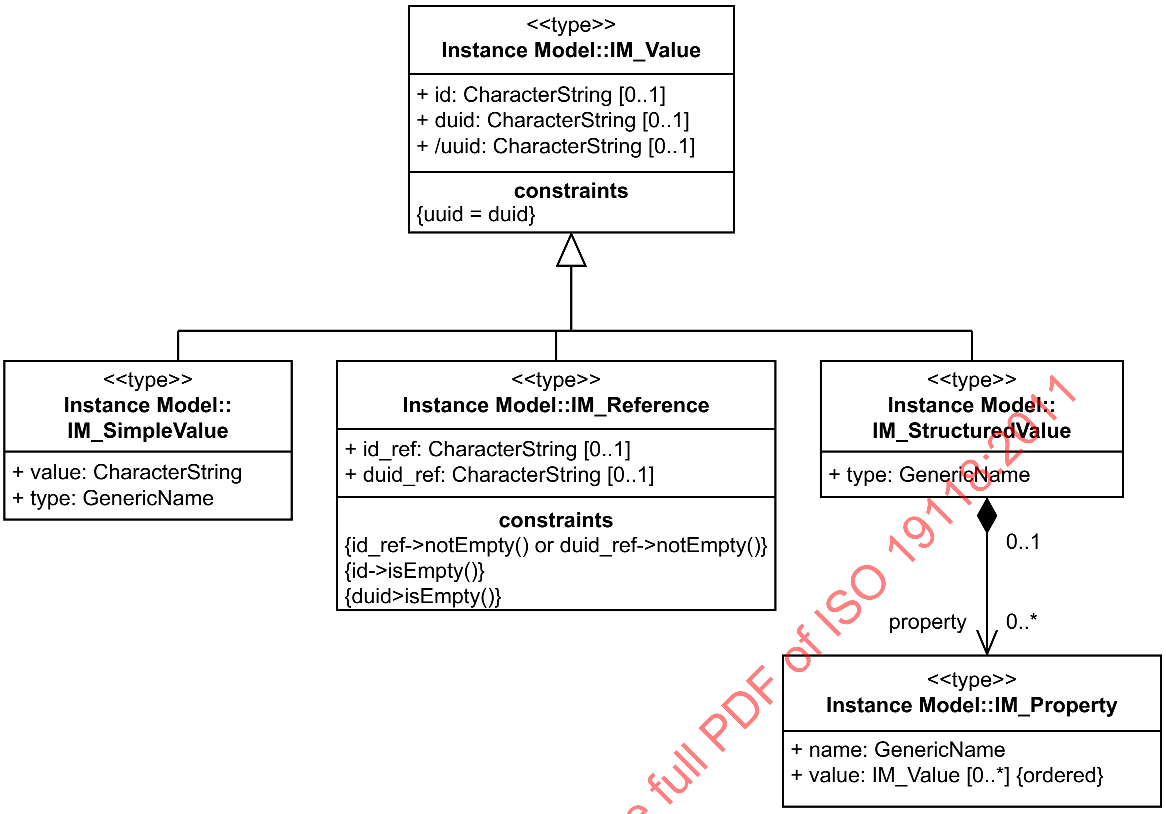


Figure 5 — Instance model — Value types

The three value types are defined as follows.

— IM_SimpleValue represents a value of simple content.

EXAMPLE An integer or a character string.

— IM_Reference represents a link or reference to a target object. The target object may be located in the same transfer unit or another one. A unique identifier (id_ref) targets an object located within the same transfer unit. A domain unique identifier (duid_ref) targets an object located within the context of an application domain.

— IM_StructuredValue represents a data type value with complex content [a sequence of properties (IM_Property)].

An object may, through its associations, be linked (or refer) to one or more objects. UML defines three different types of associations, called association, aggregation and composition, with different semantics; see ISO/TS 19103 for further details. There are, in general, two strategies for representing associations: either encapsulated as a part of the objects or detached from the objects as separate association objects or relational tables.

The encapsulated representation strategy splits an association into a source object property and a target object property. These two link properties then each point to the other object. The link property contains references to its target objects or, in the case of a composition, the target objects themselves. A link property is identified by the role name close to the target class and has a corresponding multiplicity. If the role name is missing or if the association is not navigable from the source object, then there shall be no link property. The two link property values should be consistent in that the referential integrity constraint is enforced. That is, if a source object is referring to another target object through a link property and the target object has a bi-directional association to the source object, the target object shall have a corresponding link property that is referring back to the source object. If a link property is encoded as a reference or an embedded object or not encoded at all, it is defined by a concrete encoding rule (such as Annex A).

8.2 Relation between UML and the instance model

Table 3 gives a summary of the relation between UML and the instance model.

Objects based on classes that have supertypes shall contain all the properties and association ends of their class and of their supertypes. Thus, all attributes and association ends shall be copied from the supertypes and are considered to be a part of the object. Attribute and association end names shall be the way of accessing the values of the attributes and they shall therefore be unique within the class.

Operations and constraints shall not be mapped to the instance model.

Table 3 — Summary of relationship between UML and the instance model

UML concept	Instance model
Package	N/A[a]
Class	
Stereotype	
<<Interface>>	IM_Object
<<DataType>>	IM_SimpleValue or IM_StructuredValue
<<Union>>	IM_StructuredValue
<<Enumeration>>	IM_SimpleValue
<<CodeList>>	IM_SimpleValue or IM_StructuredValue
<<Type>>	IM_Object
<<FeatureType>>	IM_Object
NONE	IM_Object
any other stereotype as defined by the UML profile used	as defined by the encoding rule specification
Attribute	IM_Property with IM_Value according to attribute type; either IM_SimpleValue, IM_Reference or IM_StructuredValue
Association	IM_Property with IM_Reference
Aggregation	IM_Property with IM_Reference
Composition	IM_Property with IM_Value according to target type; either IM_Reference (target type is a class) or IM_SimpleValue or IM_StructuredValue (target type is a data type)
Generalization	The object according to the sub-class carries all the properties that the sub-class inherits from the super-classes.
Operation	N/A
Constraint	N/A

[a] N/A stands for not applicable.

9 Encoding rules

9.1 Introduction

The requirements for specifying encoding rules are defined in 9.2 to 9.6. An encoding rule describes conversion rules for transforming data from an input data structure to an output data structure. Schemas shall be described for both the input and the output data structure. The schema for the input data structure is called an application schema.

An encoding rule shall in general specify the following:

a) general encoding requirements (9.2):

   1) application schemas and schema language,

   2) order of bits within each byte, and bytes within a word (where applicable),

   3) character repertoire and encoding,

   4) necessary exchange metadata,

   5) dataset and object identification convention;

b) input data structure (9.3):

   1) data structure used to pass data according to an application schema (data structures i_A and i_B in Figure 1) to the encoding service, called the instance model,

   2) how the instance model is related to the application schema;

c) output data structure used, called the exchange format (9.4);

d) conversion rules, called the mapping, for converting data in the instance model to the exchange format (9.5):

   1) conversion rules for encoding,

   2) if necessary, conversion rules for decoding;

e) sufficient examples of abstract data, application of conversion rules and encoded data (9.6).

9.2 General encoding requirements

9.2.1 Application schema and schema language

The encoding rule shall specify the schema language used to define application schemas and describe how an application schema is organized.

9.2.2 Bit and byte ordering

If the encoding rule specifies a binary encoding, it shall specify the order of the bits within each byte, and the order and number of bytes within any multi-byte structure (word).

NOTE These are general rules that apply to text and binary encoding rules. Even text-based encodings (such as UTF-16, UCS-2, UCS-4) require a specification of byte ordering (see http://www.unicode.org/faq/utf_bom.html).

9.2.3 Character repertoire and encoding

The character repertoire defines the characters that are allowed. The character repertoire doesn't define the language of the data. The same character repertoire can be encoded in different ways.

If more than one language is required, the application schema shall model an attribute so that the actual language of encoded text can be identified.

NOTE See class LocalizedCharacterString in ISO/TS 19103.

The encoding rule shall specify the character repertoire used and the encoding of the characters.

9.2.4 Exchange metadata

Exchange metadata are metadata about the encoded data structure. They may describe the originator of the dataset, refer to metadata information about the dataset, refer to the application schema and provide information about the encoding rule applied.

The encoding rule shall specify the exchange metadata and how it accompanies the encoded data structure.

Figure 6 illustrates an example of an ExchangeMetadata class and a corresponding EncodingRule class. The CI_Citation class is defined in ISO 19115. An exchange file normally contains one instance of the ExchangeMetadata class. The "datasetCitation" attribute contains information about the originator of the dataset, the "metadataCitation" attribute may contain information about where to find metadata about the dataset, and the "applicationSchemaCitation" attribute may hold a reference to the application schema. The EncodingRule's "encodingRuleCitation" attribute identifies the encoding rule applied and a description of how the rule was applied in this particular case. It may also contain information about the encoding service tool used to encode the dataset, where "toolName" and "toolVersion" indicate the name of the encoding service tool and its version number.

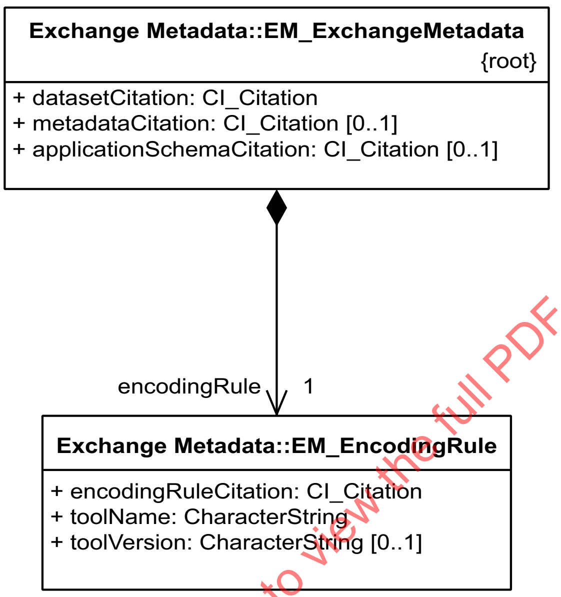

Exchange Metadata::EM_ExchangeMetadata
{root}
+ datasetCitation: CI_Citation
+ metadataCitation: CI_Citation [0..1]
+ applicationSchemaCitation: CI_Citation [0..1]
encodingRule
1
Exchange Metadata::EM_EncodingRule
+ encodingRuleCitation: CI_Citation
+ toolName: CharacterString
+ toolVersion: CharacterString [0..1]

Figure 6 — Example of exchange metadata

9.2.5 Transfer unit

9.2.5.1 Granularity and structure

It is important to define the granularity and the structure of a transfer unit so that it can be encoded and decoded efficiently. An object is here considered as the basic unit of information and it is necessary that the different types of objects be identified. The objects in a transfer unit may be structured sequentially and/or hierarchically. An object may be internally structured as a sequence of attributes, it may contain references to other objects and it may also be composed of other objects in a hierarchical manner.

An object may be split into different fragments, or its properties may be merged with other objects. Whatever changes in structure an encoding rule specifies, the instance conversion shall be unambiguously reversible.

The encoding rule shall specify the following:

— what an object is and the different types of object;

— the structure of an object;

— the structure of a dataset.

9.2.5.2 Object identification

Objects may be assigned identifiers that allow their unique identification within a particular context. Two different contexts should be considered.

a) The first context is a transfer unit. Here the object identifiers are unique within a particular unit of data transfer. These identifiers allow objects to refer to other objects within the context of that one transfer unit. The identifiers may be assigned to the objects when they are inserted into the transfer unit and are transient in nature.

b) The second context is an application domain. An application domain defines a universe and an identification convention called domain unique identifiers (DUIDs). A DUID is assigned to an object when it is created and is stable over the entire life span of the object. A DUID of a deleted object shall not be used again. A DUID is mostly used for implementation reasons. DUIDs are required for long term distributed data management and for realizing update mechanisms. These identifiers are also called persistent identifiers. A special name server may be used to resolve persistent identifiers. The identifiers shall be unique within a well-defined, limited universe defined by an application domain.

An application domain may use additional properties to identify the real-world phenomena represented by the object (e.g. a parcel number). These additional properties are not considered object identifiers in the scope of this specification.

An encoding rule may change the layout of the data, for example factoring out common character strings. To support this, types without identity, according to the application schema, may get a fragment identifier. These fragment identifiers are in any case transient (only valid inside the transfer unit), and are never a DUID.

The encoding rules shall specify the following:

— the different object identification mechanisms used;

— their internal structure.

9.3 Input data structure

The input data structure is called the instance model and it is an instrument for reasoning about application data, for defining conversion rules and for expressing examples. The instance model shall be capable of representing data according to the specification in the application schema. It may be specific to a particular application schema or capable of representing data according to any schema. It may be abstract in that it does not have to be implemented in order to realize an encoding service.

The encoding rule shall reference the general instance model as defined in Clause 8 or specify an instance model and its relation to the application schema.

This International Standard does not mandate a specific implementation of the instance model or how data are passed from/to the encoding service.

9.4 Output data structure

The output data structure defines how data is structured and represented in an exchange file. A schema may accompany the output data structure.

The encoding rule shall specify the output data structure and, if present, the output data structure schema.

9.5 Conversion rules

A conversion rule specifies how a data instance in the input data structure is converted to a data instance in the output data structure. Two sets of conversion rules may exist. The first one is the set of schema conversion rules that define a mapping from the UML schema to the schema of the output data structure. The second one is the set of instance conversion rules that define a mapping from instances in the instance model to instances in the resulting data model. Figure 7 shows the different conversion rules.

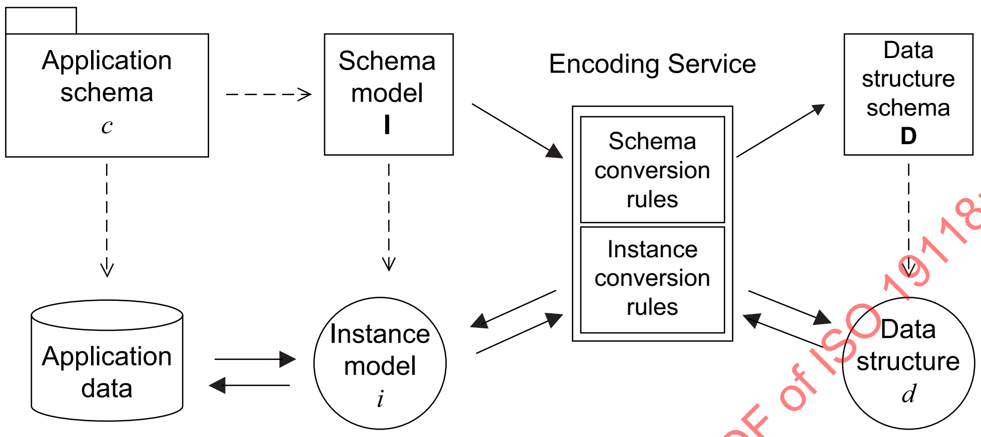


Figure 7 — Conversion rules

The encoding rule shall specify the following:

— schema conversion rules;

— instance conversion rules.

9.6 Examples

Examples are important for understanding the conversion rules and for testing encoding services.

The encoding rule shall provide the following:

— examples that illustrate conversion rules;

— test data that can be used in implementation of encoding services.

10 Encoding service

An encoding service is a software component that implements the encoding rule and provides an interface to its functionality.

An encoding service shall provide an interface to its functionality through one or more interface specifications.

An encoding service shall provide one or more of the following:

— capability to encode data according to the instance conversion rules;

— capability to decode data according to the instance conversion rules;

— capability to create an output data structure schema according to the schema conversion rules.

This International Standard does not mandate a specific interface to the encoding service.

An encoding service may provide only encoding or decoding capabilities.

Figure 8 is an example of an XML-based encoding service.

NOTE Figure 8 shows an encoding service with one interface specification. It supports three operations: generateXMLSchema, encode and decode. The “generateXMLSchema” operation can be used to generate an XML Schema file. This operation takes a schema model as input parameter and produces an XMLStream object as a result. The “encode” operation can be used to generate an XML document. It takes a schema model and an instance model as input parameters and returns an XMLStream object. The “decode” operation can be used to interpret an XML document. It takes a schema model and an XMLStream object as input and returns an instance model.

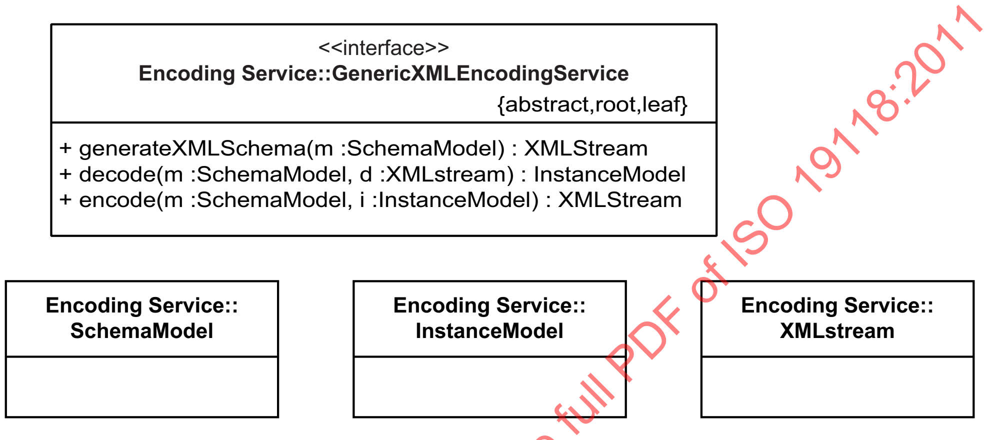

<<interface>>
Encoding Service::GenericXMLEncodingService
{abstract,root,leaf}
+ generateXMLSchema(m :SchemaModel) : XMLStream
+ decode(m :SchemaModel, d :XMLstream) : InstanceModel
+ encode(m :SchemaModel, i :InstanceModel) : XMLStream
Encoding Service:: SchemaModel
Encoding Service:: InstanceModel
Encoding Service:: XMLstream

Figure 8 — Example encoding service interface

Annex A
(normative)

XML-based encoding rule

A.1 The XML encoding in this International Standard shall be in accordance with the Extensible Markup Language (XML) 1.0. There isn't a fixed set of encoding requirements enabling a single XML-based encoding rule for all the schemas. As a result, this International Standard does not specify a schema conversion rule. Every XML-based encoding rule shall

— specify or identify the input data structure, typically the profile of UML used by the encoding rule,

— specify the XML output data structure,

— specify the schema conversion rules between the input and output data structures, and

— conform to all requirements specified in this annex.

A.2 Different use cases result in different encoding rules. Currently, the set of International Standards known as the “ISO 19100 series” specifies two XML-based encoding rules:

— ISO 19136:2007, Annex E, specifies an XML-based encoding rule for ISO 19109-conformant application schemas that can be represented using a restricted profile of UML that allows for a conversion to XML Schema. The encoding rule has mainly been developed for the purpose of application schemas specifying feature types and their properties. The encoding rule uses XML Schema for the output data structure schema.

— ISO/TS 19139 specifies an XML-based encoding rule for conceptual schemas specifying types that describe geographic resources, e.g. metadata according to ISO 19115 and feature catalogues according to ISO 19110. The encoding rule supports the UML profile as used in the UML models commonly used in the standards developed by Technical Committee ISO/TC 211. The encoding rule uses XML Schema for the output data structure schema.

A.3 Different XML-based encoding rules may be required and specified by an information community. Examples for such requirements include, but are not limited to, the following:

— support for the XML-based encoding rule specified in Annex C (this encoding rule is in use by communities);

— support for a different UML profile not covered by an existing XML-based encoding rule;

— support for an output data structure schema other than XML Schema (e.g. Relax NG);

— support for new XML technologies or new versions of existing XML technologies;

— support for specific conversions to optimize the use of the capabilities of XML;

— support for other XML-related requirements that are established in a community.

Annex B
(normative)

Abstract test suit

B.1 Test cases for an encoding rule

B.1.1 General

All encoding rules shall pass all test cases of the abstract test suite in B.1 to B.3, according to the conformance classes defined in clause 2.2.

B.1.2 Documentation of conversion rules

a) Test purpose: Verify that the encoding rule defines instance or schema conversion rules.

b) Test method: Inspect the encoding rule documentation.

c) Reference: 9.5.

d) Test type: Basic.

B.1.3 Consistent instance and schema conversion rules

a) Test purpose: Verify that, if the encoding rule defines schema and instance conversion rules, they are consistent.

b) Test method: Inspect the encoding rule documentation.

c) Reference: 9.5.

d) Test type: Capability.

B.1.4 Bit and byte ordering

a) Test purpose: Verify that the encoding rule specifies bit and byte ordering or defines metadata to specify it at runtime.

b) Test method: Inspect the encoding rule documentation.

c) Reference: 9.2.2, 9.2.4.

d) Test type: Capability.

B.1.5 Character set and encoding

a) Test purpose: Verify that the encoding rule specifies character set and encoding or defines metadata to specify it at runtime.

b) Test method: Inspect the encoding rule documentation.

c) Reference: 9.2.3, 9.2.4.

d) Test type: Capability.

B.1.6 Identification convention

a) Test purpose: Verify that the encoding rule specifies an identification convention or defines metadata to specify it at runtime.

b) Test method: Inspect the encoding rule documentation.

c) Reference: 9.2.4, 9.2.5.2.

d) Test type: Capability.

B.1.7 Encoding metadata

a) Test purpose: Verify that the encoding rule defines metadata to specify encoding aspects at runtime that are not fixed in the encoding rule.

b) Test method: Inspect the encoding rule documentation.

c) Reference: 9.2.2, 9.2.3, 9.2.4, 9.2.5.2, 9.5.

d) Test type: Capability.

B.2 Test cases for instance conversion rules

B.2.1 Documentation of instance conversion

a) Test purpose: Verify that the encoding rule defines how instances of the generic instance model are mapped to the transfer format.

b) Test method: Inspect the encoding rule documentation.

c) Reference: 9.5.

d) Test type: Capability.

B.2.2 Completeness of instance conversion

a) Test purpose: Verify that the encoding rule defines instance conversion rules for the complete generic instance model.

b) Test method: Inspect the encoding rule documentation.

c) Reference: Clause 8 and 9.5.

d) Test type: Capability.

B.2.3 Unambiguous instance conversion

a) Test purpose: Verify that the encoding rule defines instance conversion rules that map an instance from the generic instance model to the transfer format and back again without loss of information.

b) Test method: Inspect the encoding rule documentation.

c) Reference: 9.2.5.1.

d) Test type: Capability.

B.3 Test cases for schema conversion rules

B.3.1 Documentation of schema conversion

a) Test purpose: Verify that the encoding rule defines how an application schema is mapped to the transfer format schema.

b) Test method: Inspect the encoding rule documentation.

c) Reference: 9.5.

d) Test type: Capability.

B.3.2 Completeness of schema conversion

a) Test purpose: Verify that the encoding rule defines schema conversion rules for the complete UML profile defined by ISO/TS 19103 or a profile of it.

b) Test method: Inspect the encoding rule documentation.

c) Reference: 9.5.

d) Test type: Capability.

B.3.3 Unambiguous instance conversion

a) Test purpose: Verify that the encoding rule defines schema conversion rules that result in a mapping of an instance from the generic instance model to the transfer format and back again without loss of information.

b) Test method: Inspect the encoding rule documentation.

c) Reference: 9.2.5.1.

d) Test type: Capability.

B.4 Test cases for an encoding service

B.4.1 Documentation of service interface

a) Test purpose: Verify that the encoding service provides a documented interface.

b) Test method: Inspect the encoding service documentation.

c) Reference: Clause 10.

d) Test type: Basic.

B.4.2 Reference to encoding rule

a) Test purpose: Verify that the encoding service documentation references the encoding rule that it implements.

b) Test method: Inspect the encoding service documentation.

c) Reference: Clause 10.

d) Test type: Basic.

B.4.3 Implementation of specified encoding rule

a) Test purpose: Verify that the encoding service implements the referenced encoding rule.

b) Test method: Inspect the encoding service implementation.

c) Reference: Clause 10.

d) Test type: Capability.

B.5 Support of any application schema

a) Test purpose: Verify that the encoding service supports any application schema as specified by the UML profile used by the encoding rule.

b) Test method: Inspect the encoding service interface to see if it supports the generic instance model.

c) Reference: Clause 8.

d) Test type: Capability.

B.6 Encoding data

a) Test purpose: Verify that the encoding service provides functionality to write data.

b) Test method: Inspect the encoding service interface to see if it provides functionality for writing data.

c) Reference: Clause 10.

d) Test type: Capability.

B.7 Decoding data

a) Test purpose: Verify that the encoding service provides functionality to read data.

b) Test method: Inspect the encoding service interface to see if it provides functionality for reading data.

c) Reference: Clause 10.

d) Test type: Capability.

B.8 Schema generation

a) Test purpose: Verify that the encoding service provides functionality to generate a format schema.

b) Test method: Inspect the encoding service interface to see if it provides format schema generation functionality.

c) Reference: Clause 10.

d) Test type: Capability.

Annex C
(informative)

XML-based encoding rule in use by communities

C.1 Introduction

This annex introduces an example of the XML-based encoding rule for neutral data interchange in some communities. The encoding rule is compatible with the Unified Modelling Language (UML) and defines an encoding rule based on the Extensible Markup Language (XML).

NOTE ISO 19118:2005, Annex A, specified an XML-based encoding rule. The rule has been used in some communities. This Annex C is the revision of the rule for the communities in compliance with this International Standard.

This annex follows the requirements in Clause 9 and specifies the following:

a) the general encoding requirements in C.2;

b) the input data structure in C.3;

c) the output data structure in C.4;

d) the conversion rules in C.5 and C.6.

Examples are given in C.5.8 and C.6.4.

The conversion rules are based on the idea that the class definitions in the application schema are mapped to type declarations in XML Schema, and that the objects in the instance model are mapped to corresponding element structures in the XML document. Figure C.1 depicts the two types of conversion rule.

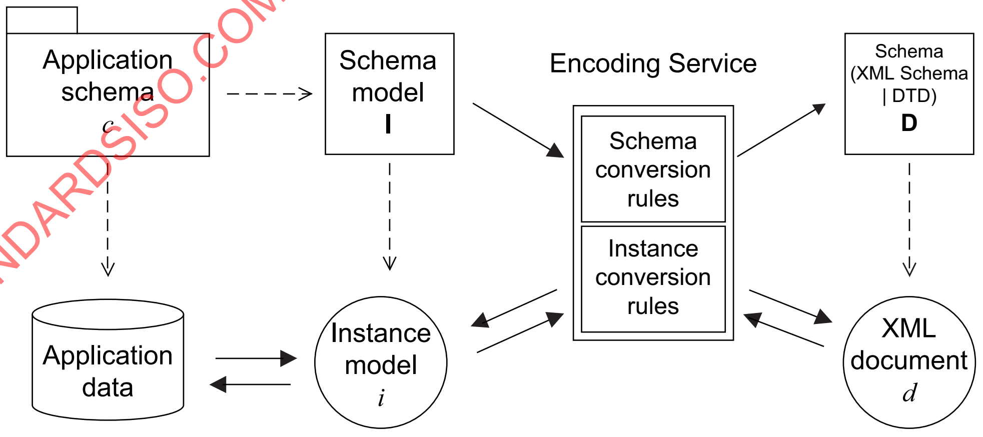


Figure C.1 — XML-based conversion rules

C.2 General encoding requirements

C.2.1 Application schema

C.2.1.1 Introduction

The application schema shall be expressed in UML schema language according to the rules specified in ISO/TS 19103 and ISO 19109. An application schema consists of application-defined concepts expressed as classes and associations. Some of these classes may be imported from the standardized schemas in other standards.

C.2.1.2 Class

The fundamental modelling concept in UML is the class. It is identified by a name and may have a stereotype. The class may have attributes, operations and constraints and participate in associations. A class defines a type that can be used as a building block to define other classes. Types are the fundamental building blocks out of which all forms of data can be composed and encoded, e.g. numbers, coordinates, text strings, dates, and objects. A type defines the legal value domain and the operations on values of that domain.

Two main categories of types are the simple data types and the complex data types. A type is a simple data type if there is a canonical encoding defined for the type. This canonical encoding may define how to represent values of the type as bits in a memory location or as characters in a textual encoding. Examples of simple types are integer, float and string. A type is of a complex data type when there is no canonical encoding defined for the type. Examples are object types, structured types, records, and collections. A complex type consists of a structured collection of basic and complex attributes that can be encoded using a combination of basic types and special structuring primitives. A third category is the external data types that are defined outside the set of International Standards known as the "ISO 19100 series".

a) Simple data types – Fundamental types for representing values:

   1) basic data types: CharacterString, Integer, Binary, Boolean, Date, Time, etc.;

   2) enumerated data types and code lists: A list of legal values, where each value is a word or a code with associated semantics.

b) Complex data types – Types for representing more complex collections of values:

   1) collection data types: Template types for representing multiple occurrences of other types – Set, Bag, Sequence, Dictionary, etc.;

   2) structured data types: Types that define attribute groups;

   3) object types: Types whose instances are objects; often defined in application schemas or standardized schemas – GM_Point, Building, etc.;

   4) interfaces: Types whose instances are service components.

c) External data types – Basic or complex types with a well-defined encoding that are not defined within the set of International Standards known as the "ISO 19100 series"; examples are image formats such as NASA MODIS, TIFF, etc. Special referencing mechanisms shall be specified that allow references, usually stored in separate files, to external data types.

ISO/TS 19103 defines the modelling concepts applied, including a number of simple and complex data types together with their semantics. The application developers are free to specify user-defined data types and object types using the stereotype extension mechanism of UML. Other International Standards within the set of International Standards known as the "ISO 19100 series" define more specialized data types. It can occasionally be necessary for application schema developers to use external data types. External data types are basic or structured data types that are defined outside this series of International Standards. It is necessary that they have a well-defined encoding; examples are image formats such as NASA MODIS and TIFF. Objects of external data types can be stored in separate files and special referencing mechanisms that allow references to external data types shall be specified.

The stereotypes allowed on classes are described in Table C.1.

Table C.1 — Stereotypes on classes

Stereotype	Description
<<BasicType>>	Defines a basic data type that has defined a canonical encoding
<<DataType>>	Defines a structured data type. The instances are not considered objects and shall therefore not have an identity
<<Union>>	Defines a union data type
<<Enumeration>>	Defines an enumerated data type
<<CodeList>>	Defines an extendable enumerated data type, consisting of code and value pairs
<<Interface>>	Defines a service interface and shall not be encoded
<<Type>>	Defines an object type. Instances shall have identity
<<FeatureType>>	Defines a feature type. Instances shall have identity
NONE	Defines an object type. Instances shall have identity

An object is considered the fundamental unit of interchange. Only aspects that are essential for capturing an object's state shall be considered for data interchange purposes. Attributes and associations shall be encoded. Operations and constraints shall not be considered further. An encoding rule shall specify how instances of classes are represented, including how attributes and associations are structured and represented.

C.2.1.3 Attributes

An attribute is identified by a name. It may have a multiplicity statement and shall always have a type. The multiplicity statement shall indicate the number of legal value occurrences of a particular attribute.

Combinations of multiplicity and use of collection data types allows nesting of values.

There shall be mechanisms to handle null values. The type shall define how a null value is represented. See ISO/TS 19103 for the definition of the basic and collection data types.

C.2.1.4 Associations

Associations define relationships between classes that involve connections between their instances. An instantiation of an association is called a link. A link contains an ordered list of references to objects. UML defines three different types of associations called association, aggregation and composition.

— Association defines general relationships between classes.

— Aggregation defines weak part-whole relationships between classes.

— Composition defines strong part-whole relationships between classes.

The three association types have different semantics. ISO/TS 19103 gives further details.

The end of an association is identified by its role name and its target class, and is further described by a multiplicity statement. The role name shall be used to represent a link. If the role name is missing, then there is no link.

C.2.2 Exchange structure and exchange metadata

The exchange structure shall be divided into three parts. The first part is the exchange metadata, which shall be described according to 9.2.4. The second part is the dataset that shall contain XML elements that correspond to independent objects. The third part is optional and shall contain an update section of the update primitives as described in C.2.5.

C.2.3 Character repertoire and language

The underlying exchange structure shall support the language and character repertoire:

— language (see C.5.6 for language tagging);

— character repertoire (see ISO/IEC 10646).

C.2.4 Dataset and object identification

The two ways of specifying object identification as described in 9.2.5.2 shall be as follows:

— unique dataset identifiers according to XML's ID mechanism;

— universal unique identifiers according to the application domain's specification.

NOTE The term “universal unique identifier” has the same meaning as “domain unique identifier”. This annex uses it and “UUID” as its abbreviation to keep the compatibility for the communities in use. The application of UUIDs in this International Standard is not limited to a particular mathematical basis, like ISO/IEC 11578 or ISO/IEC 9834-8. The “universal unique identifier” is not actually qualified by application domains, in contrast to the “domain unique identifier”, which is qualified by the application domain.

EXAMPLE An application domain can want to use a two-component identifier. The first component is the domain name and the next component is an integer instance number. The components can be separated by a colon “:”. The instance numbers can be encoded in hex. There are no restrictions on the size of the instance number. Examples of two UUIDs in a domain called “example” are: “example:F23C30” and “example:FFFFFF12345A”.

C.2.5 Update mechanism

An update mechanism allows previously exchanged data to be brought up to date without the requirement for reissuing a complete new dataset. Policies and procedures for update shall be defined by the specific application. Three basic update primitives are usually defined: *add*, *modify* and *delete*. These primitives work on the object level, but may also be defined to work on the attribute or association level. Any object that has previously been transmitted with a UUID may be modified or deleted. An update dataset contains an ordered sequence of update primitives. The basic primitives are described as follows.

a) *add*: A new object has been added to the source dataset and shall be added into the target dataset. An add primitive shall contain information about the new object to be added and may contain information about where it is inserted in the target dataset.

b) *modify*: An existing object has been modified in the source dataset and shall be modified in the target dataset. A modify primitive shall contain information that identifies the target object and the actual modifications. Examples of modification information could range from a complete object to just an updated attribute.

c) *delete*: An existing object has been deleted in the source dataset and shall therefore be deleted in the target dataset. The delete primitive shall contain information that identifies the target object to be deleted.

The users may extend the list of primitives.

C.3 Input data structure

C.3.1 Instance model

The instance model is capable of representing data described by application schemas expressed in UML. As the input data structure, this encoding rule uses an instance model which is based on the generic instance model defined in Clause 8. The instance model is defined in Figures C.2 and C.3.

Figure C.2 shows IM_Dataset, IM_Object and IM_Property classes that are basically the same as the ones defined in 8.1 and in Figure 4.

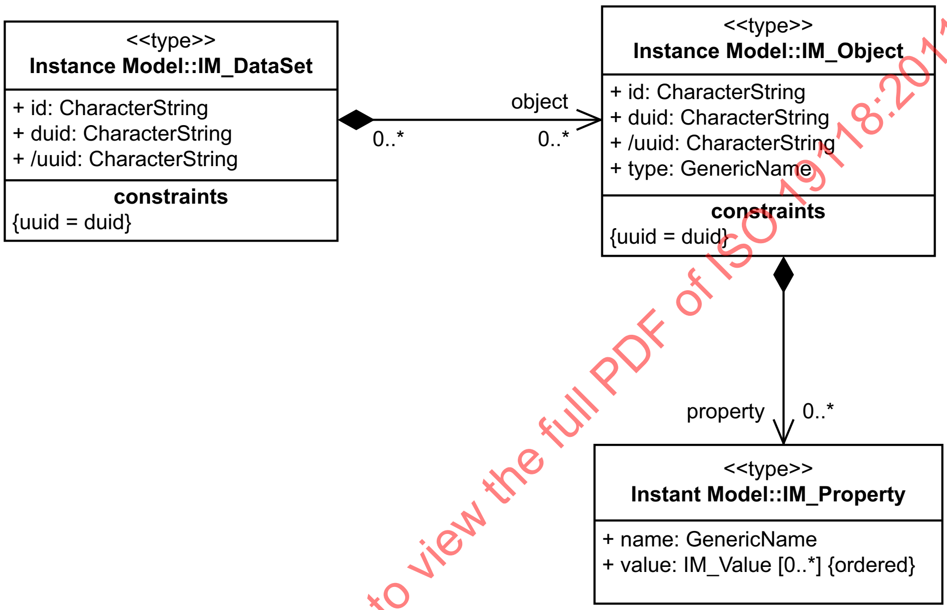


Figure C.2 — Instance model — Dataset, object and property

The instance model consists of a dataset represented by IM_Dataset. A dataset contains a sequence of objects represented by IM_Object. An object consists of a sequence of properties represented by IM_Property.

There is an additional definition of attribute uuid in IM_Dataset and IM_Object. The derived attribute uuid is defined as the alternative name of attribute duid.

NOTE 1 The definition of attribute uuid is to keep the compatibility for the communities in use.

Value types in the instance model are defined in Figure C.3, which is based on Figure 5.

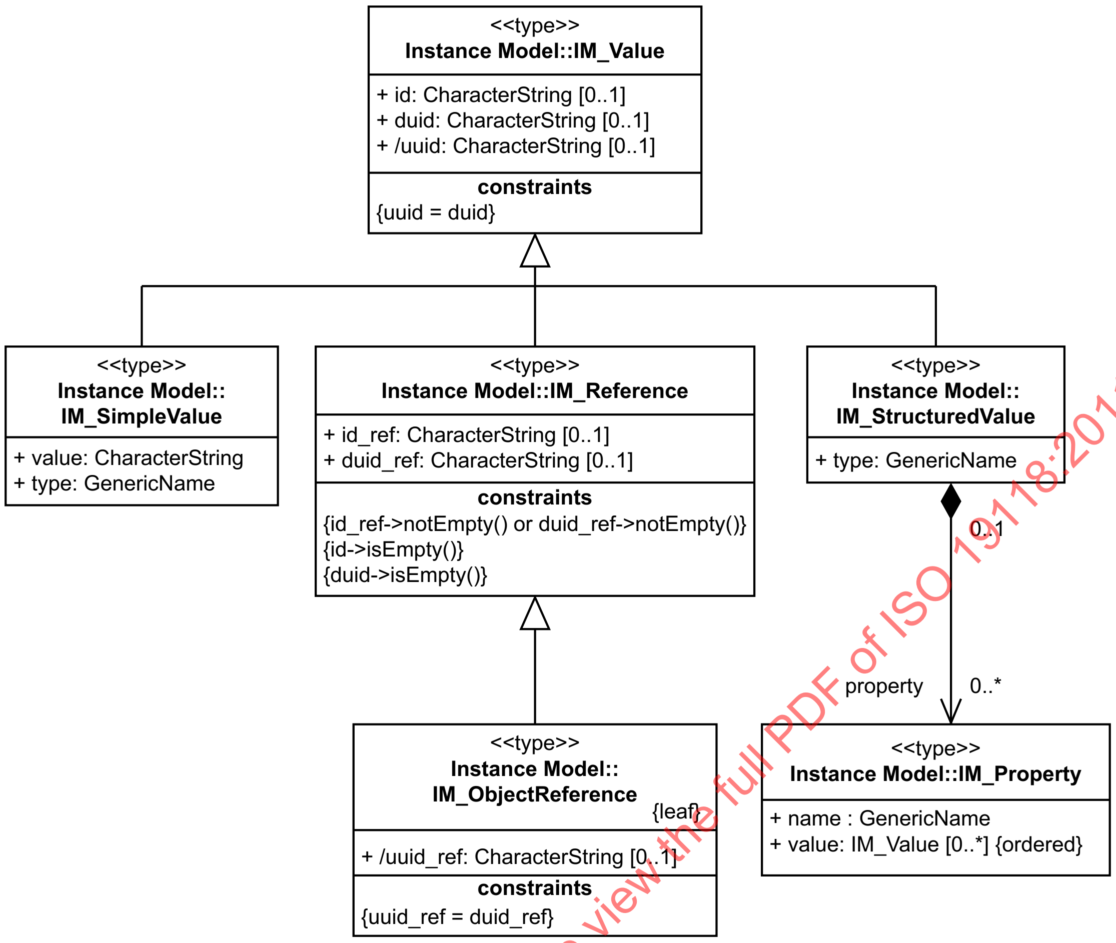


Figure C.3 — Instance model — Value types

IM_Value is the superclass of four value types defined as follows.

— IM_SimpleValue represents a value of a basic type.

EXAMPLE An integer or a character string.

— IM_Reference represents a link or reference to a target object. The target object may be located in the same or other datasets.

— IM_ObjectReference is a subtype of IM_Reference. A unique identifier (id_ref) targets an object located within the same dataset. A universal unique identifier (uuid_ref and duid_ref) targets an object located within the context of an application domain.

— IM_StructuredValue represents a data type value with complex content [a sequence of properties (IM_Property)].

The difference between this instance model and the generic instance model defined in Clause 8 is as follows.

a) IM_Value contains the derived attribute uuid as the alternative name of attribute duid.

b) IM_ObjectReference is defined and it has an additional derived attribute uuid_ref which defines the alternative name of attribute duid_ref.

Other classes defined in Figure C.3 are identical to the ones defined in 8.1 and Figure 5.

NOTE 2 The definition of the class IM_ObjectReference and the attribute uuid in IM_Value and IM_ObjectReference are to keep the compatibility for the communities in use.

C.3.2 Relation between UML and the instance model

Tables C.2 and C.3 give a summary of the relation between UML and the instance model. Thus an abstract class shall not be instantiated.

Objects based on classes that have supertypes shall contain all the properties, associations and compositions of their class and of their supertypes. Thus, all attributes and associations shall be copied from the supertypes and are considered to be a part of the object. Attribute and association names shall be the way of accessing the values of the attributes and they shall therefore be unique within the class.

Operations and constraints shall not be mapped to the instance model.

Table C.2 — Summary of relationship between UML and the instance model

UML concept	Instance model
Package	N/A[a]
Class Stereotype <<Interface>> <<BasicType>> <<DataType>> <<Union>> <<Enumeration>> <<CodeList>> <<Type>> or NONE Abstract class Sub-class	 N/A IM_SimpleValue IM_StructuredValue IM_StructuredValue IM_SimpleValue IM_SimpleValue IM_Object N/A attributes and associations shall be copied from super-classes
Attribute	IM_Property with IM_Value according to data type (either IM_SimpleValue or IM_StructuredValue)
Association	IM_Property with IM_Value of IM_ObjectReference
Aggregation	IM_Property with IM_Value according to either IM_StructuredValue or IM_ObjectReference
Composition	IM_Property with IM_Value of IM_StructuredValue
Operation	N/A
Constraint	N/A

[a] N/A stands for not applicable.

Table C.3 — Mapping of attributes with multiplicity and collection type

Attribute	Instance model
a1 [0..*] : Integer is the same as a1 : Sequence<Integer>[a]	IM_Property with multiple value occurrences
a2 : Sequence<T>	IM_Property with multiple occurrences of value type T
a3 : Dictionary<T1, T2>	IM_Property with multiple occurrences of IM_StructuredValue with two elements of value type T1 and T2
a4 [0..*] : Sequence<Integer>	IM_Property with multiple occurrences of IM_StructuredValue with integer elements

[a] This is also valid for any other basic type.

Attribute and association name clashes can cause problems when using inheritance. A simple way to avoid this is to ensure that all attributes and associations shall be prefixed with their appropriate class name; alternatively, the method of avoiding name clashes is left to the user.

Attribute and association redeclaration can also cause problems when using inheritance. Redeclaration happens when an attribute or association declared in a supertype gets redeclared in a subtype with a new or restricted type. Many object-oriented programming languages cannot handle redeclarations and it should be carefully considered whether redeclarations should be deprecated.

C.3.3 Application schema and instance model — Example

C.3.3.1 Application schema

This example defines an application schema, gives some data and shows the mapping from the data into the instance model.

Figure C.4 defines an example application schema that defines four classes and their relationships. Here, class C0 has three attributes. Attribute “a1” has a multiplicity of zero or one, which means it is optional, and a basic data type “Real”. Attribute “a2” has a multiplicity of zero or many. Attribute “a3” is of a structured data type. The association between “C0” and “C1” has two role names. “role1” belongs to C0 and names the association to C1 in the context of C0. Class C1 has an association to C0 which is called “main”. Notice that class C2 has an attribute named “pos” with a class data type Point.

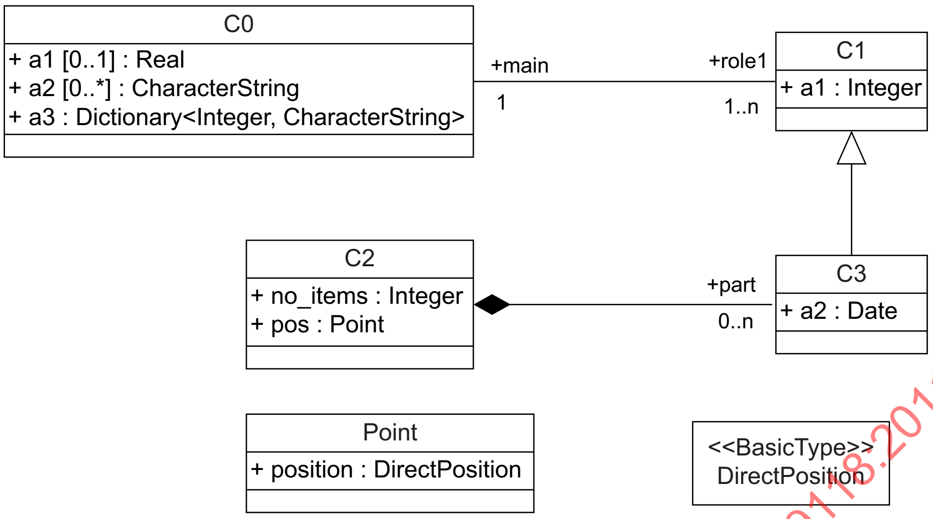


Figure C.4 — Application schema — Example

C.3.3.2 Data — Example

The example data is shown in Figure C.5 in an object diagram.

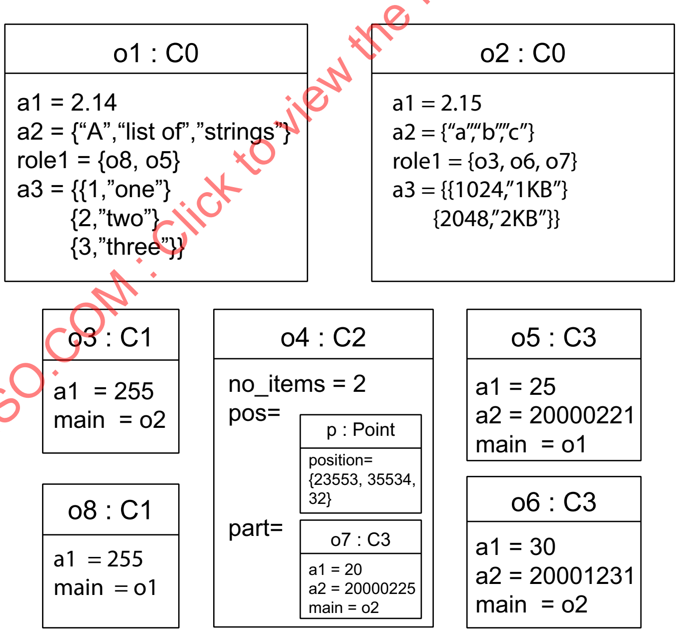


Figure C.5 — Data — Example

C.4 Output data structure

C.4.1 XML document

This encoding rule is based on the XML Recommendation 1.0. XML is a text format and it is necessary that the values of all data types be character encoded. Data shall be encoded using XML elements and the rules given in the XML recommendation. The basic units of encoding in XML are XML elements. An element may have attributes and content. This enables a hierarchical structure and combined with XML's linking facilities a network structure can be created. The exchange structure shall consist of a sequence of elements corresponding to the objects in the instance model.

The schema for the output data structure that governs the structure of the exchange format shall be a valid XML Schema.

C.4.2 XML Schema

An XML Schema Document (XSD) defines a number of complex types, simple types and element declarations that define the allowable structure and data instances of an XML document. The XML Schema conversion rules are defined in C.5.

C.5 Schema conversion rules

C.5.1 XML Schema

The schema conversion rules define how to produce an XML Schema Document (XSD) according to an application schema expressed in UML. The main purpose of the XSD is to ensure that XML documents produced using the data conversion rules are valid.

The XSD shall contain type definitions and attribute and element declarations that correspond to the classes defined in the application schema. The elements shall be organized in an exchange structure. The XSD shall adhere to XML Schema Part 1: Structures and Part 2: Datatypes.

The XSD may physically be represented in a single schema document or divided into several separate (sub) schema documents. Logically, it shall be referred to as a single schema utilizing the import or include mechanisms of XML Schema. There are no restrictions on the use of namespaces in an XSD.

A number of general rules are defined in C.5.2 to C.5.8. A class shall in general be converted to a type definition according to C.5.2, it may be converted to an element declaration according to C.5.4 and it may be a member of the exchange structure.

Exceptions to the general rules are allowed as long as they are documented.

NOTE In the following the namespace “xs:” is used to refer to the namespace of XML Schema, which is "http://www.w3.org/2001/XMLSchema".

C.5.2 Types

C.5.2.1 <<BasicType>>

C.5.2.1.1 General rule

A class stereotyped <<BasicType>> shall be converted to a simpleType declaration in XML Schema. Any of the data types defined in XML Schema can be used as building blocks to define user-defined basic types. The encoding of the basic types shall follow the canonical representation defined in XML Schema Part 2: Datatypes.

Modelling rules for basic types are defined in ISO/TS 19103. The basic types defined in ISO/TS 19103 are converted in C.5.2.1.2 to C.5.2.1.15.

Users are permitted to restrict the basic types further using the restriction mechanisms defined in XML Schema.

NOTE The different types are not clearly defined in ISO/TS 19103 and neither is the <<BasicType>> stereotype used. The following declarations, therefore, follow a subset of the data type definitions in XML Schema Part 2: Datatypes.

C.5.2.1.2 Number

A Number is defined in ISO/TS 19103 as an abstract type and a supertype of Integer, Real and Decimal. However, it is much used in the different standardized schemas. XML Schema does not define an abstract number type, but defines decimal to be a supertype of integer.

NOTE A number data type is declared as a simple data type based on the decimal data type.

```
<xs:simpleType name="Number">
    <xs:restriction base="xs:decimal"/>
</xs:simpleType>
```

C.5.2.1.3 Integer

An Integer shall be based on the XML Schema integer data type. The value domain may be restricted.

```
<xs:simpleType name="Integer">
    <xs:restriction base="xs:integer"/>
</xs:simpleType>
```

C.5.2.1.4 Decimal and Real

The Decimal and Real types are both based on the XML Schema decimal type.

```
<xs:simpleType name="Decimal">
    <xs:restriction base="xs:decimal"/>
</xs:simpleType>

<xs:simpleType name="Real">
    <xs:restriction base="xs:decimal"/>
</xs:simpleType>
```

NOTE ISO/TS 19103 does not define the conceptual difference between a Decimal and a Real.

C.5.2.1.5 Vector

A Vector is defined as a sequence of numbers. The list construct defines a list of decimal values.

```
<xs:simpleType name="Vector">
    <xs:list itemType="xs:decimal"/>
</xs:simpleType>
```

C.5.2.1.6 Character

A Character is represented as an XML Schema string restricted to contain only one character.

```
<xs:simpleType name="Character">
    <xs:restriction base="xs:string">
        <xs:length value="1" fixed="true"/>
    </xs:restriction>
</xs:simpleType>
```

C.5.2.1.7 CharacterString

CharacterString is based on XML Schema string that can represent any ISO/IEC 10646 string.

```
<xs:simpleType name="CharacterString">
    <xs:restriction base="xs:string"/>
</xs:simpleType>
```

NOTE For language identification, see C.5.6.

C.5.2.1.8 Date

Date is based on XML Schema date, which has a canonical encoding according to ISO 8601.

```
<xs:simpleType name="Date">
    <xs:restriction base="xs:date"/>
</xs:simpleType>
```

C.5.2.1.9 Time

Time is based on XML Schema time, which has a canonical encoding according to ISO 8601.

```
<xs:simpleType name="Time">
    <xs:restriction base="xs:time"/>
</xs:simpleType>
```

C.5.2.1.10 DateTime

DateTime is based on XML Schema dateTime, which has a canonical encoding according to ISO 8601.

```
<xs:simpleType name="DateTime">
    <xs:restriction base="xs:dateTime"/>
</xs:simpleType>
```

C.5.2.1.11 Boolean

Boolean is based on XML Schema boolean. The values are “0” or “false”, which represent logical false, and “1” and “true”, which represent logical true.

```
<xs:simpleType name="Boolean">
    <xs:restriction base="xs:boolean"/>
</xs:simpleType>
```

C.5.2.1.12 Logical

Logical defines three values: true, maybe and false. It is represented as a union between XML Schema boolean and two enumerated values, “0.5” and “maybe”, which represent the maybe value.

```
<xs:simpleType name="Logical">
    <xs:union>
        <xs:simpleType>
            <xs:restriction base="xs:boolean"/>
        </xs:simpleType>
        <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
                <xs:enumeration value="maybe"/>
                <xs:enumeration value="0.5"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:union>
</xs:simpleType>
```

C.5.2.1.13 Probability

Probability is a decimal number between 0 and 1.0.

```
<xs:simpleType name="Probability">
    <xs:restriction base="xs:decimal">
        <xs:minInclusive value="0.0"/>
        <xs:maxInclusive value="1.0"/>
    </xs:restriction>
</xs:simpleType>
```

C.5.2.1.14 Binary

XML Schema defines two binary data types base64Binary and hexBinary. Two types are defined that shall be used as binary data types in UML: BinaryBase64 and BinaryHex. A special choice type called Binary is defined that contains either a BinaryBase64 or a BinaryHex.

```
<xs:simpleType name="BinaryBase64">
        <xs:restriction base="xs:base64Binary"/>
    </xs:simpleType>

    xs:simpleType name="BinaryHex">
        <xs:restriction base="xs:hexBinary"/>
    </xs:simpleType>

    <xs:complexType name="Binary">
        <xs:choice>
            <xs:element name="BinaryBase64" type="BinaryBase64"/>
            <xs:element name="BinaryHex" type="BinaryHex"/>
        </xs:choice>
    </xs:complexType>
```

NOTE The Binary data type is not defined in ISO/TS 19103.

C.5.2.1.15 UnlimitedInteger

UnlimitedInteger is a basic type that has a value domain from 0 to infinity. The symbol “*” is defined to represent the infinite value.

```
<xs:simpleType name="UnlimitedInteger">
        <xs:union>
            <xs:simpleType>
                <xs:restriction base="xs:nonNegativeInteger"/>
            </xs:simpleType>
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="*"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:union>
</xs:simpleType>
```

C.5.2.2 <<DataType>>

C.5.2.2.1 General rule

A class stereotyped <<DataType>> shall be converted to a complexType definition in XML Schema. The attributes and possible associations shall be declared as XML attributes or local XML elements in a sequence construct according to C.5.3. The order of the property elements is, therefore, given in the complex type definition.

The data types defined in ISO/TS 19103 are converted in C.5.2.2.2 and C.5.2.2.3.

NOTE The different types are not clearly defined in ISO/TS 19103 and neither are all classes that are structured data types defined using the <<DataType>> stereotype. The declarations in C.5.2.2.2 and C.5.2.2.3, therefore, interpret the data type definitions in ISO/TS 19103.

C.5.2.2.2 Multiplicity

A Multiplicity class is defined as a multiplicity range from lower to upper in ISO/TS 19103. Here it is interpreted as a <<DataType>> and defined as follows.

```
<xs:complexType name="Multiplicity">
    <xs:sequence>
        <xs:element name="range" type="MultiplicityRange" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="MultiplicityRange">
    <xs:sequence>
        <xs:element name="lower" type="xs:nonNegativeInteger"/>
        <xs:element name="upper" type="UnlimitedInteger"/>
    </xs:sequence>
</xs:complexType>
```

C.5.2.2.3 Units of measure

The units of measure types defined in ISO/TS 19103 are intended for use as definitions of local and international measurement systems. Quantities are then measured according to a unit system probably published in a units of measure dictionary. The models are unfortunately difficult to understand, no examples are provided and the description is vague. The POSC (Petroleum Open Standards Consortium) Units of Measure Recommendation is an alternative to ISO/TS 19103. A UML diagram that illustrates some of the basic ideas in the specification is shown in Figure C.6. An instance of a UnitOfMeasure defines a measure system, gives it a name and, if necessary, provides information about how to convert quantities to a base unit using the general formula $Y = (A + BX)/(C + DX)$, where X is the value of the unit to be converted and Y is in the base unit. If $A = D = 0$ and $C = 1$, then B becomes a conversion factor. If $A = D = 0$, the conversion factor is described by a fraction. Otherwise, it is described by the four parameters. An instance of a Measure is then a decimal value with a reference to the unit of measure.

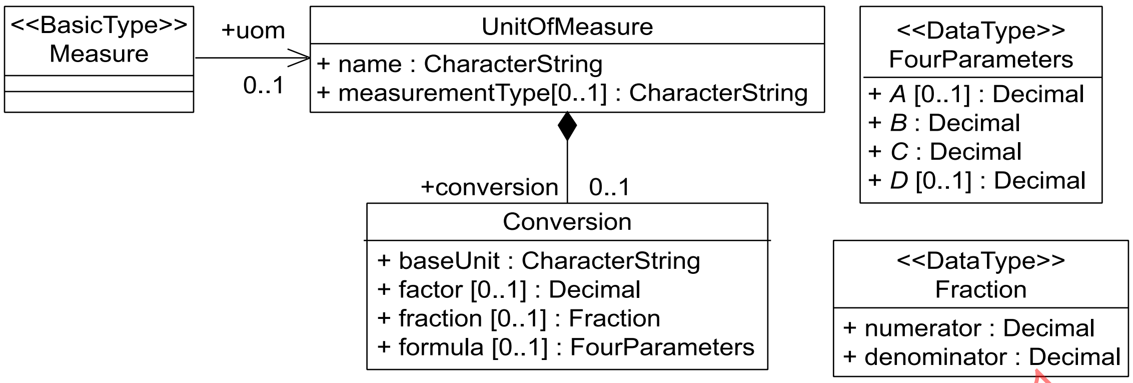


Figure C.6 — Units of measure

It is recommended to use the XML Schema definitions as defined in the POSC Units of Measure Recommendations.

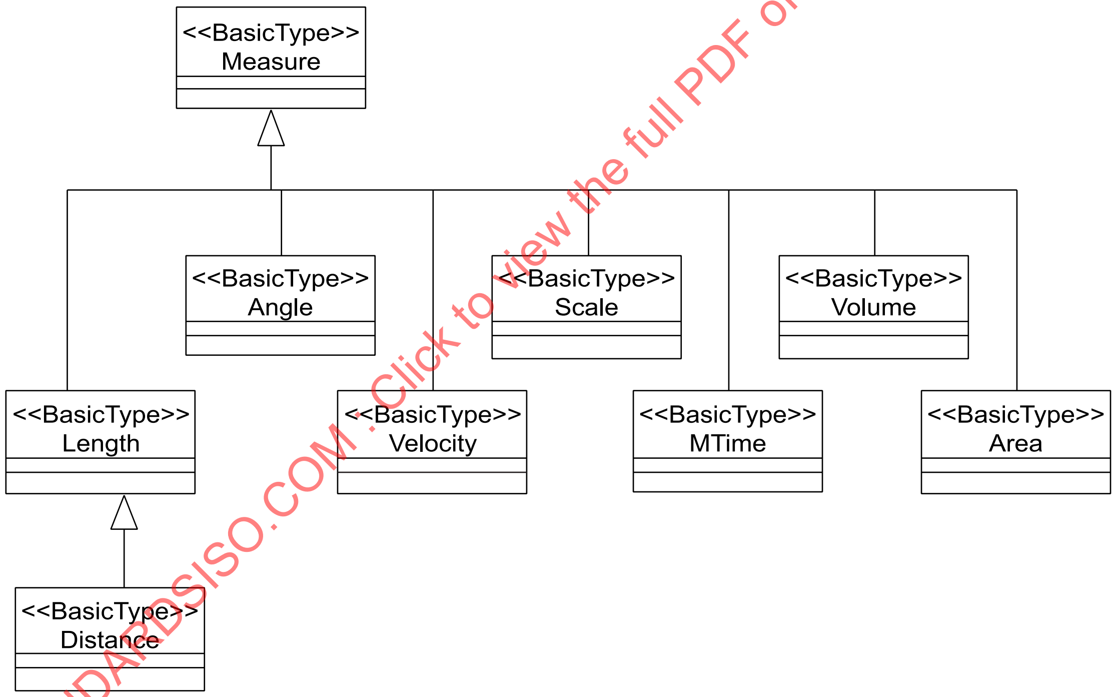


Figure C.7 — Measure types

The measure types defined by ISO/TS 19103 (Length, Angle, Velocity, Scale, MTime, Area and Volume) shall be defined as follows.

```
<xs:complexType name="Measure">
        <xs:simpleContent>
            <xs:extension base="Decimal">
                <xs:attribute name="uom" type="URI"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
```

```
<xs:complexType name="Length">
    <xs:simpleContent>
        <xs:extension base="Measure"/>
    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="Angle">
    <xs:simpleContent>
        <xs:extension base="Decimal"/>
    </xs:simpleContent>
</xs:complexType>

xs:complexType name="Scale">
    <xs:simpleContent>
        <xs:extension base="Measure"/>
    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="Area">
    <xs:simpleContent>
        <xs:extension base="Measure"/>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="Velocity">
    <xs:simpleContent>
        <xs:extension base="Measure"/>
    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="MTime">
    <xs:simpleContent>
        <xs:extension base="Measure"/>
    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="Distance">
    <xs:simpleContent>
        <xs:extension base="Measure"/>
    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="Volume">
    <xs:simpleContent>
        <xs:extension base="Measure"/>
    </xs:simpleContent>
</xs:complexType>
```

C.5.2.3 <<Enumeration>>

A class stereotyped as <<enumeration>> shall be converted to a simple type that restricts a text string to a number of enumerated values.

EXAMPLE An example is the Sign enumeration, which is based on a string and restricts its value to either “+”, “positive”, “–” or “negative”.

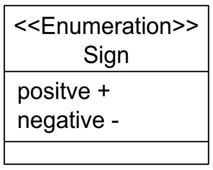


Figure C.8 — Example of <<Enumeration>>

```
<xs:simpleType name="Sign">
    <xs:restriction base="xs:string">
        <xs:enumeration value="+"/>
        <xs:enumeration value="-"/>
        <xs:enumeration value="positive"/>
        <xs:enumeration value="negative"/>
    </xs:restriction>
 </xs:simpleType>
```

C.5.2.4 <<CodeList>>

A class stereotyped as <<codelist>> shall not be converted to the output schema but may instead be mapped to a dictionary that stores the code and value pairs defined in the code list. The dictionary shall be made publicly available and its Web address shall be given as a URI.

An attribute of a code list type shall be encoded as a string value.

See also Figure C.22 for class CodeListExtraction and Figure C.27 for the representation of codelists.

EXAMPLE Figure C.9 shows a code list called BorderCL that is mapped to a dictionary in XML. Note that instead of listing the code-value pairs as attributes to the BorderCL, a comment box is used.

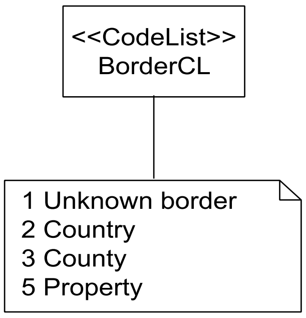


Figure C.9 — Example of <<CodeList>>

```
<codelist name="BorderCL">
    <codevalue code="1" value="Unknown border"/>
    <codevalue code="2" value="Country"/>
    <codevalue code="3" value="County"/>
    <codevalue code="5" value="Property"/>
</codelist>
```

C.5.2.5 <<Union>>

A class stereotyped <<Union>> lists a number of attributes, and the semantics is that only one of the attributes can be present at any time. It shall be converted to a complex type definition with the attributes as elements in a choice construction.

EXAMPLE The GM_Position union is taken from ISO 19107. It is mapped to a complex type with the appropriate local element declarations in a choice construction.

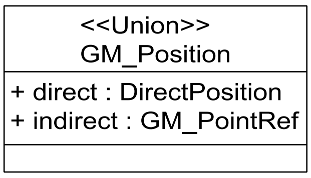


Figure C.10 — Example of <<Union>>

```
<xs:complexType name="GM_Position">
    <xs:choice>
        <xs:element name="direct" type="DirectPosition"/>
        <xs:element name="indirect" type="GM_PointRef"/>
    </xs:choice>
</xs:complexType>
```

C.5.2.6 Object types

C.5.2.6.1 General rule

A class with no stereotype or stereotyped <<Type>> shall be mapped to a complex type definition with the same name as the class. The complex type definition shall include identification attributes, either inherited from IM_Object or through a reference to the IM_ObjectIdentificaton attribute group.

C.5.2.6.2 Record types

A few record types are defined in ISO/TS 19103. They are remodelled and interpreted in Figure C.11. The AttributeName and TypeName are modelled as basic types based on CharacterString.

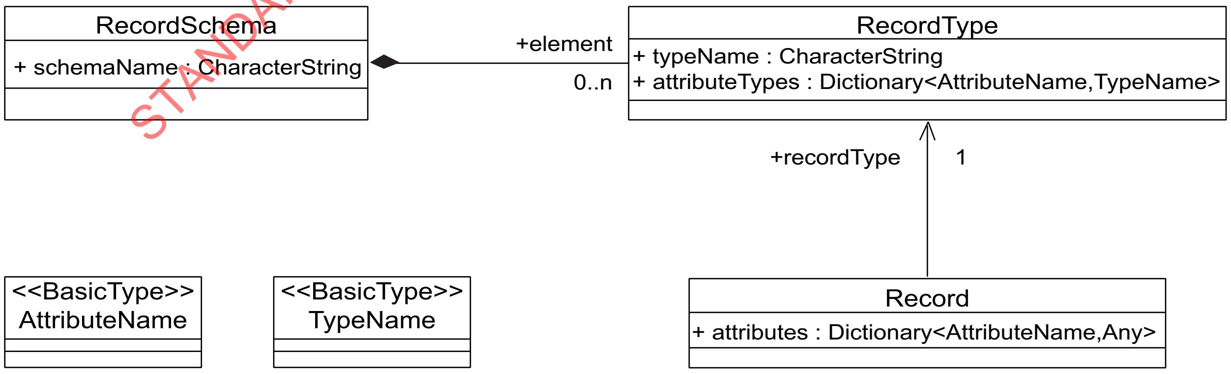

RecordSchema
+ schemaName : CharacterString
+element
0..n
RecordType
+ typeName : CharacterString
+ attributeTypes : Dictionary<AttributeName,TypeName>
+recordType
1
<<BasicType>>
AttributeName
<<BasicType>>
TypeName
Record
+ attributes : Dictionary<AttributeName,Any>

Figure C.11 — Record types

```
<xs:complexType name="RecordSchema">
        <xs:sequence>
            <xs:element name="schemaName" type="CharacterString"/>
            <xs:element name="element" type="RecordType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attributeGroup ref="IM_ObjectIdentification"/>
</xs:complexType>

<xs:complexType name="RecordType">
        <xs:sequence>
            <xs:element name="typeName" type="CharacterString"/>
            <xs:element name="attributeTypes" type="Dictionary_AttributeName_TypeName_"/>
        </xs:sequence>
        <xs:attributeGroup ref="IM_ObjectIdentification"/>
</xs:complexType>

<xs:complexType name="Record">
        <xs:sequence>
            <xs:element name="attributes" type="Dictionary_AttributeName_Any_"/>
            <xs:element name="recordType" type="ref_RecordType"/>
        </xs:sequence>
        <xs:attributeGroup ref="IM_ObjectIdentification"/>
</xs:complexType>
```

C.5.2.7 Bound template type

A bound template type is a type where the parameters are bound to actual argument values. ISO/TS 19103 defines five different template types: Set<T>, Bag<T>, Sequence<T>, CircularSequence<T> and Dictionary<K,V>. The first four take one parameter whereas Dictionary takes two. These types are usually bound in attribute declarations; see the "attributeTypes" attribute of the RecordType defined in Figure C.11.

— A bound template type shall be converted to a complex type definition that corresponds to the template type. The type declaration shall be named. The name may be constructed by concatenating the template name with the argument names separated with underscore "_" characters. The less-than "<", comma "," and greater-than ">" characters cannot be used in the name.

    — A bound Set, Bag, Sequence or CircularSequence template type shall be mapped to a complex type definition that consists of a sequence construct of unbounded multiplicity containing one element named and typed according to the single parameter type. If the parameter type is a basic type, then a simple type definition may be used instead, utilizing the list construct of XML Schema.

    — A bound template type of Dictionary shall be mapped to a complex type definition that consists of a sequence construct of unbounded multiplicity with two elements named and typed according to the two parameters.

EXAMPLE 1 The direct position data type defined in ISO 19107 is shown in Figure C.12.

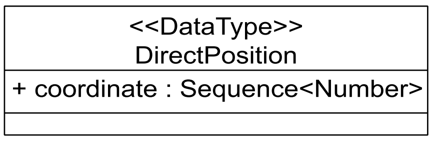


Figure C.12 — Example of bounded template type

The Sequence<Number> defines a bound template type and, according to the general rule, maps to the following.

```
<xs:complexType name="Sequence_Number_">
    <xs:sequence maxOccurs="unbounded">
        <xs:element name="Number" type="Number"/>
    </xs:sequence>
</xs:complexType>
```

Since the argument type is of a basic type, it may also be mapped to the following:

```
<xs:simpleType name="Sequence_Number_">
    <xs:list itemType="Number"/>
</xs:simpleType>
```

EXAMPLE 2 The bound template types defined in Figure C.11 are mapped as follows.

```
<xs:complexType name="Dictionary_AttributeName_Any_">
        <xs:sequence maxOccurs="unbounded">
            <xs:element name="AttributeName" type="CharacterString"/>
            <xs:element name="Any" type="xs:anyType"/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="Dictionary_AttributeName_TypeName_">
        <xs:sequence maxOccurs="unbounded">
            <xs:element name="AttributeName" type="CharacterString"/>
            <xs:element name="TypeName" type="CharacterString"/>
        </xs:sequence>
</xs:complexType>
```

C.5.2.8 Inheritance

C.5.2.8.1 General

The inheritance mechanism in UML allows a subtype to inherit its supertypes' attributes and associations. In single inheritance, a type can inherit only from a single supertype, whereas in multiple inheritance a type can inherit from more than one type. UML allows both single and multiple inheritance. XML Schema only supports single inheritance. Therefore, it is necessary to simulate multiple inheritance.

Inheritance shall be realized either

— by the XML Schema extension or restriction mechanism called *single inheritance* (C.5.2.8.2), or

— by copying attributes and associations from supertypes into the target type called *multiple inheritance* (C.5.2.8.3).

In case of multiple inheritance, the attributes and associations shall be copied into the target type.

C.5.2.8.2 Single inheritance

The general rule shall be to use XML Schema's extension mechanism for complex types. But if an attribute or association is redefined the restriction mechanism shall be used.

EXAMPLE Figure C.13 shows a supertype S1 with subtypes S2 and S3. S1 is an abstract class. S4 and S5 are subtypes of S2. Note that S5 redefines “attr1” and that it is necessary, therefore, to use the restriction mechanism.

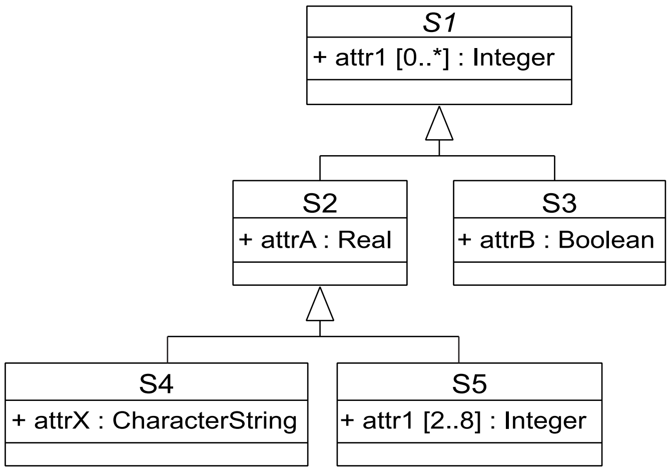


Figure C.13 — Example of single inheritance

```
<xs:complexType name="S1" abstract="true">
    <xs:complexContent>
        <xs:extension base="IM_Object">
            <xs:sequence>
                <xs:element name="attr1" type="Integer" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="S2">
    <xs:complexContent>
        <xs:extension base="S1">
            <xs:sequence>
                <xs:element name="attrA" type="Real"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="S3">
    <xs:complexContent>
        <xs:extension base="S1">
            <xs:sequence>
                <xs:element name="attrB" type="Boolean"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="S4">
    <xs:complexContent>
        <xs:extension base="S2">
            <xs:sequence>
                <xs:element name="attrX" type="CharacterString"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

```
<xs:complexType name=”S5”>
    <xs:complexContent>
        <xs:restriction base="S2">
            <xs:sequence>
                <xs:element name=”attr1" type="Integer" minOccurs="2" maxOccurs="8"/>
                <xs:element name="attrA" type="Real"/>
            </xs:sequence>
        </xs:restricton>
    </xs:complexContent>
</xs:complexType>
```

C.5.2.8.3 Multiple inheritance

The procedure for copying the attributes/associations is to start with the left supertype and copy its attributes and associations, then continue with the next supertype to the right until the rightmost supertype is reached. The subtype's attributes are added last. A conflict occurs if a supertype or subtype defines an attribute or association with the same name as previously copied. In case of a name conflict, the latter attribute or association shall take precedence and replace the previously copied one.

EXAMPLE Figure C.14 defines four types: T1, T2, T3 and T4. T4 is a subtype of T1, T2 and T3.

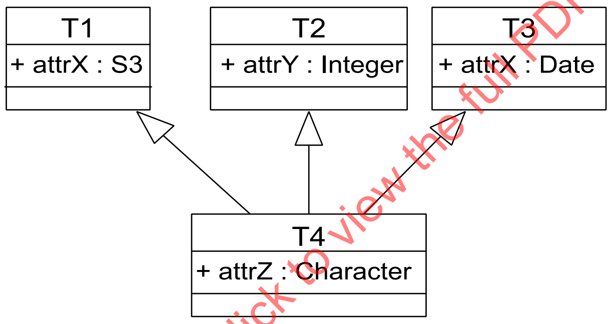


Figure C.14 — Example of multiple inheritance

```
<xs:complexType name=”T1”>
    <xs:sequence>
        <xs:element name=”attrX" type="S3"/>
    </xs:sequence>
    <xs:attributeGroup ref="IM_ObjectIdentification"/>
</xs:complexType>

<xs:complexType name=”T2”>
    <xs:sequence>
        <xs:element name=”attrY" type="Real"/>
    </xs:sequence>
    <xs:attributeGroup ref="IM_ObjectIdentification"/>
</xs:complexType>

<xs:complexType name=”T3”>
    <xs:sequence>
        <xs:element name=”attrX" type="Date"/>
    </xs:sequence>
    <xs:attributeGroup ref="IM_ObjectIdentification"/>
</xs:complexType>
```

```
<xs:complexType name="T4">
    <xs:sequence>
        <xs:element name="attrX" type="Date"/>
        <xs:element name="attrY" type="Integer"/>
        <xs:element name="attrZ" type="Character"/>
    </xs:sequence>
    <xs:attributeGroup ref="IM_ObjectIdentification"/>
</xs:complexType>
```

C.5.2.9 Substitution types

The use of a supertype as an attribute type means that an instance of the attribute can be of one of the concrete subtypes defined by the inheritance hierarchy of the supertype. XML Schema does not support this dynamic type mechanism directly.

Three alternative approaches may be used.

a) Declare a standard element declaration with type corresponding to the supertype. In the exchange file, the xsi:type attribute shall be used to indicate the required type. See Example 1.

b) Define global elements with a substitution group that matches the inheritance hierarchy of the supertype. The global element shall be referred to within an element declaration. See Example 2.

c) Define choice groups for each supertype that contains a choice of element declarations for each of the concrete types in the inheritance hierarchy of the supertype. The choice group shall be referred to within an element declaration. See Example 3.

If the copy mechanism described in C.5.2.8.3 is used, only approach c) shall be used.

EXAMPLE 1 Approach a): A class Ex2 defines an attribute of type S1, see Figure C.15. S1 is the abstract supertype with an inheritance hierarchy defined in Figure C.13.

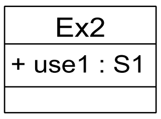


Figure C.15 — Example attribute of a supertype

```
<xs:complexType name="Ex2">
    <xs:sequence>
        <xs:element name="use1" type="S1"/>
    </xs:sequence>
</xs:complexType>
```

Use of xsi:type in the exchange file:

```
<Ex1>
    <use1 xsi:type="S3">
        <attr1>42</attr1><attr1>43</attr1><attr1>44</attr1>
        <attrB>true</attrB>
    </use1>
</Ex1>
```

EXAMPLE 2 Approach b): Use of global substitution elements.

```
<xs:element name="S1" type="S1" abstract="true"/>
<xs:element name="S2" type="S2" substitutionGroup="S1"/>
<xs:element name="S3" type="S3" substitutionGroup="S1"/>
<xs:element name="S4" type="S4" substitutionGroup="S2"/>
<xs:element name="S5" type="S5" substitutionGroup="S2"/>

<xs:complexType name="Ex2">
    <xs:sequence>
        <xs:element name="use1">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="S1"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<Ex1>
    <use1>
        <S3>
            <attr1>42</attr1><attr1>43</attr1><attr1>44</attr1>
            <attrB>true</attrB>
        </S3>
    </use1>
</Ex1>
```

EXAMPLE 3 Approach c): Use of choice groups.

```
<xs:group name="S1">
    <xs:choice>
        <xs:element name="S2" type="S2"/>
        <xs:element name="S3" type="S3"/>
        <xs:element name="S4" type="S4"/>
        <xs:element name="S5" type="S5"/>
    </xs:choice>
</xs:group>

<xs:complexType name="Ex2">
    <xs:sequence>
        <xs:element name="use1">
            <xs:complexType>
                <xs:group ref="S1"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

The instance becomes exactly the same as the instance in Example 2. The benefit of this approach is that there are no global elements.

C.5.2.10 <<Abstract>> or abstract class

An abstract class may not be represented as a complexType declaration if the copy down mechanism is used. If single inheritance is used, an abstract class shall be converted to a complexType definition according to C.5.2.6 that has the “abstract” attribute set to true.

C.5.2.11 <<ExternalType>>

All classes stereotyped <<ExternalType>> shall either be mapped to a simple type that restricts the XML Schema type anyURI or to a NOTATION type.

EXAMPLE

```
<xs:simpleType name="Modis">
    <xs:restriction base="xs:anyURI"/>
</xs:simpleType>
```

C.5.3 Property element declarations

C.5.3.1 Attribute

An attribute defines a characteristic of a class. An attribute shall be converted to an *element declaration* or an *attribute declaration* in the class' complex type declaration. An *element declaration* is the default rule. If an attribute is of basic type with multiplicity of zero or one it may be converted to an *attribute declaration*. All other attributes shall be converted to an *element declaration*.

The default rule is to convert all attributes and derived attributes.

An *attribute declaration* defines an attribute with a name and a basic data type. The multiplicity of an attribute shall be according to Table C.4. The default multiplicity of an attribute declaration in XML Schema is optional.

Table C.4 — Multiplicity mapping for attributes

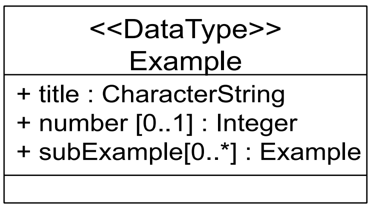
UML	Optional	Necessary attribute declaration
1 (default)	false	optional="false"
0..1	true (default)	—

An *element declaration* defines an element with a name and a type. The multiplicity shall be according to Table C.5. The default values are indicated in the table, and it is not necessary that they be declared. Alternatively, both minimum and maximum values may be given.

Table C.5 — Multiplicity mapping for content elements

UML	minOccurs	maxOccurs	Necessary element declaration
1 (default)	1 (default)	1 (default)	—
0..1	0	1 (default)	minOccurs=”0”
0..*	0	unbounded	minOccurs=”0” maxOccurs=”unbounded”
1..*	1 (default)	unbounded	maxOccurs=”unbounded”
2..8	2	8	minOccurs=”2” maxOccurs=”8”

EXAMPLE A data type Example is declared in UML. It has three attributes “title”, “number” and “subExample”, see Figure C.16. Both the “title” and “number” attribute can be converted to an attribute declaration, whereas “subExample” is of a complex type and it is necessary that it be mapped to an element declaration.



<<DataType>>
Example
+ title : CharacterString
+ number [0..1] : Integer
+ subExample[0..*] : Example

Figure C.16 — Example attribute

The default element declaration gives the following:

```
<xs:complexType name="Example" >
    <xs:sequence>
        <xs:element name="title" type="CharacterString"/>
        <xs:element name="number" type="Integer" minOccurs="0"/>
        <xs:element name="subExample" type="Example" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
```

Here “title” and “number” are converted according to the attribute declaration rule.

```
<xs:complexType name="Example" >
    <xs:sequence>
        <xs:element name="subExample" type="Example" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="title" type="CharacterString" optional="false"/>
    <xs:attribute name="number" type="Integer"/>
</xs:complexType>
```

C.5.3.2 Association

An association defines a general relationship between two classes. In the following, one of the classes is called a source class and the other is called a target class. Source objects store references to target objects and vice versa.

— The complex type corresponding to the source class shall contain an element declaration if the association is navigable and the target class is identified by a role name. The name of the element shall be the role name identifying the target class, and the type shall be either *IM_ObjectReference*, or a type that is based on or has the attributes defined in *IM_ObjectReference*. The element declaration shall have multiplicity according to Table C.5.

— The complex type corresponding to the target class shall contain an element declaration if the association is navigable and the source class is identified by a role name. The name of the element shall be the role name identifying the source class, and the type shall be either *IM_ObjectReference*, or a type that is based on or has the attributes defined in *IM_ObjectReference*. The element declaration shall have multiplicity according to Table C.5.

EXAMPLE An association between class A and B is defined, see Figure C.17. Only A knows about B since only one role name is defined.

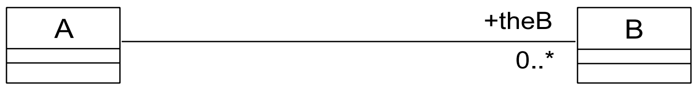


Figure C.17 — Example association

The XML Schema declarations are as follows:

```
<xs:complexType name="A">
    <xs:sequence>
        <xs:element name=”theB” type=”ref_B” minOccurs=”0” maxOccurs=”unbounded”/>
    </xs:sequence>>
</xs:complexType>

<xs:complexType name="ref_B">
    <xs:attributeGroup ref="IM_ObjectReference"/>
</xs:complexType>
```

C.5.3.3 Aggregation

An aggregation defines a weak whole-part relationship between an aggregate (whole) and a constituent part. The ownership is weak in that parts can be members of more than one aggregate at the same time. Thus, a part object may be shared by more than one aggregate object. An aggregate can, therefore, in general store references only to its parts but may, in case of complete ownership, contain the respective parts.

— The complex type corresponding to the aggregate class shall contain an element declaration where the name corresponds to the role name identifying the part class. The multiplicity of this element shall be according to Table C.5. The type of the element shall be based on an *IM_ObjectReference* and may contain zero or one element of a type that corresponds to the part class.

— The complex type corresponding to the part class shall contain an element declaration if the association is navigable and the target class is identified by a role name. The name of the element shall correspond to the role name identifying the aggregate class and type shall be based on an *IM_ObjectReference*. The element declaration shall have multiplicity according to Table C.5.

EXAMPLE An aggregation between the aggregate C and the part D is defined, see Figure C.18. C identifies D by the role name “theD” and D identifies C by the role name “theC”.

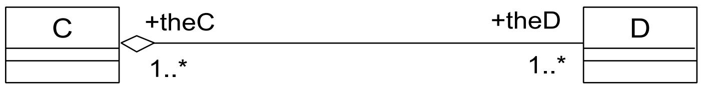


Figure C.18 — Example aggregation

```
<xs:complexType name="C">
    <xs:sequence>
        <xs:element name="theD" minOccurs="1" maxOccurs="unbounded">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="D" type="D" minOccurs="0" maxOccurs="1"/>
                </xs:sequence>
                <xs:attributeGroup ref="IM_ObjectReference"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="D">
    <xs:sequence>
        <xs:element name="theC" type="IM_ObjectReference" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
```

C.5.3.4 Composition

A composition defines a strong whole-part relationship between a composite (whole) and a constituent part. The ownership is strong in that a part can be a member of exactly one composite object. A composite shall, therefore, contain its respective parts.

— The complex type corresponding to the composite class shall contain an element declaration where the name corresponds to the role name identifying the part class and the type corresponds to the type of the part class. The element declaration shall have multiplicity according to Table C.5.

— The complex type corresponding to the part class shall not contain any element declaration, even if a role name identifies the composite class. This is implicit because a part is always contained within a composite class.

EXAMPLE A composition between classes E and F is defined, see Figure C.19. E identifies F by a target role name “theF”.

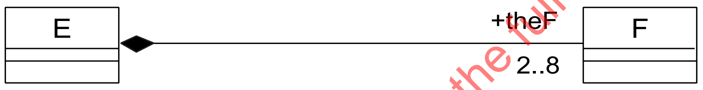


Figure C.19 — Example composition

```
<xs:complexType name="E">
    <xs:sequence>
        <xs:element name="theE" type="F" minOccurs="2" maxOccurs="8"/>
    </xs:sequence>
</xs:complexType>
```

C.5.4 Element declarations

C.5.4.1 Document structure

The GI element shall be the root element of the exchange file. It contains three elements: exchangeMetadata, dataset and update; see Figure C.20. The dataset and update elements may have identity whereas the exchange metadata shall not.

The attributes of the GI element are as follows:

— version: CharacterString = "1.0";

— timeStamp: DateTime;

— exchangeMode [0..1] : CharacterString.

The **version** attribute is required to be set to “1.0”. This indicates that the exchange file conforms to this version of this International Standard. Revised versions of this International Standard will have another number associated with them. The **timestamp** attribute indicates the date and time of when the data was encoded. That is, when the exchange file was produced. The **exchangeMode** attribute is user-defined and its value may indicate the context or mode of the exchange file.

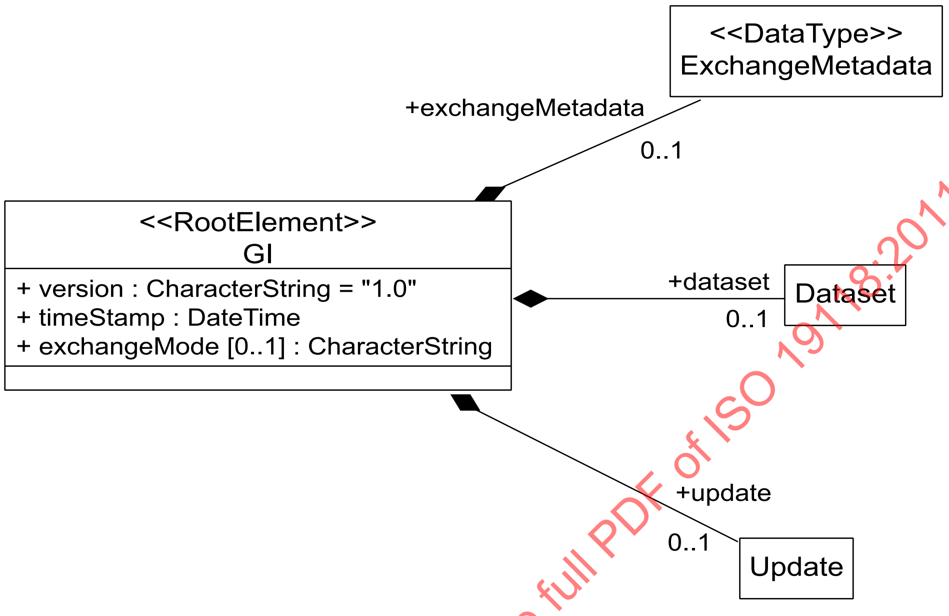


Figure C.20 — Document structure

The declaration of the GI element is as follows.

```
<xs:element name="GI">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="exchangeMetadata" type="ExchangeMetadata" minOccurs="0"/>
            <xs:element name="dataset" type="Dataset" minOccurs="0"/>
            <xs:element name="update" type="Update" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="version" type="CharacterString" use="required" fixed="1.0"/>
        <xs:attribute name="timeStamp" type="DateTime" use="required"/>
        <xs:attribute name="exchangeMode" type="CharacterString"/>
    </xs:complexType>
</xs:element>
```

C.5.4.2 Dataset and object elements

A dataset contains one or more elements that encode objects, called object elements, which shall be declared.

— All complex type definitions with identification attributes are candidates for object element declaration. Some might not be considered as independent objects and might not, therefore, be defined.

— The name of the element shall be the same as the type name or it may be a tag name defined for this type. An object element may be declared either as a local or as a global element. The default is as a local element.

— All object elements shall be grouped in a choice group that may be named **Object**. Thus the choice group either refers to global declared elements or declares the object elements locally. This choice group shall be used to restrict the legal objects in a dataset.

EXAMPLE The Object group defines the object elements of the example in C.5.2.8.3. The dataset type refers to an unbounded sequence of the object group (Figure C.21). This means that the dataset can contain only the four elements T1, T2, T3 and T4.

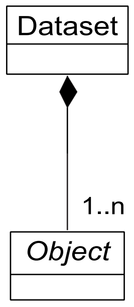


Figure C.21 — Dataset contains objects

```
<xs:complexType name="Dataset">
    <xs:sequence maxOccurs="unbounded">
        <xs:group ref="Object"/>
    </xs:sequence>
    <xs:attributeGroup ref="IM_ObjectIdentification"/>
</xs:complexType>

<xs:group name="Object">
    <xs:choice>
        <xs:element name=”T1” type=”T1”/>
        <xs:element name=”T2” type=”T2”/>
        <xs:element name=”T3” type=”T3”/>
        <xs:element name=”T4” type=”T4”/>
    </xs:choice>
</xs:group>
```

C.5.4.3 Exchange metadata

The exchange metadata types are defined in Figure C.22. The CI_Citation type is imported and reused from ISO 19115.

An ExchangeMetadata shall contain information that describes the dataset. The “datasetCitation” attribute describes the originator of the dataset. The “metadataCitation” attribute refers to relevant metadata for the dataset, the “applicationSchemaCitation” refers to the application schema used, and the “configFileCitation” describes the configuration file used. The “encodingRule” composition describes the encoding rule used to produce the dataset. If the dataset contains attributes of code lists, it shall indicate which code lists are used and their validity. This is done by the “codeLists” composition.