INTERNATIONAL STANDARD

ISO 21090

First edition
2011-02-15

Health informatics — Harmonized data types for information interchange

Informatique de santé — Types de données harmonisées pour une interchangeabilité d'informations

Reference number
ISO 21090:2011(E)

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

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) 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 21090 was prepared by Technical Committee ISO/TC 215, *Health informatics*.

Introduction

Assistance from the Infrastructure and Messaging Committee in HL7 and the support of Connecting for Health have been instrumental in the preparation of this International Standard, which is a shared document between Health Level Seven (HL7) and ISO, and has been produced according the terms of the agreement between HL7, CEN and ISO (JIC, see http://www.global-e-health-standards.org/), which ensures that the content is fully available through ISO, CEN and HL7 publication channels.

Health informatics — Harmonized data types for information interchange

1 Scope

This International Standard

— provides a set of datatype definitions for representing and exchanging basic concepts that are commonly encountered in healthcare environments in support of information exchange in the healthcare environment;

— specifies a collection of healthcare-related datatypes suitable for use in a number of health-related information environments;

— declares the semantics of these datatypes using the terminology, notations and datatypes defined in ISO/IEC 11404, thus extending the set of datatypes defined in that standard;

— provides UML definitions of the same datatypes using the terminology, notation and types defined in Unified Modelling Language (UML) version 2.0;

— specifies an XML (Extensible Mark-up Language) based representation of the datatypes.

The requirements which underpin the scope reflect a mix of requirements gathered primarily from HL7 Version 3 and ISO/IEC 11404, and also from CEN/TS 14796, ISO 13606 (all parts) and past ISO work on healthcare datatypes.

This International Standard can offer a practical and useful contribution to the internal design of health information systems, but is primarily intended to be used when defining external interfaces or messages to support communication between them.

2 Normative references

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

ISO/IEC 4217, *Codes for the representation of currencies and funds*

ISO/IEC 8601, *Data elements and interchange formats — Information interchange — Representation of dates and times*

ISO/IEC 8824 (all parts), *Information technology — Abstract Syntax Notation One (ASN.1)*

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

ISO/TS 22220, *Health Informatics — Identification of subjects of health care*

IETF RFC 1738, *Uniform Resource Locators (URL)*

IETF RFC 1950, *ZLIB Compressed Data Format Specification version 3.3*

IETF RFC 1951, *DEFLATE Compressed Data Format Specification version 1.3*

IETF RFC 1952, *GZIP file format specification version 4.3*

IETF RFC 2045, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*

IETF RFC 2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*

IETF RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*

IETF RFC 3066, *Tags for the Identification of Languages*

IETF RFC 3966, *The tel URI for Telephone Numbers*[1)]

FIPS PUB 180-1, *Secure Hash Standard*

FIPS PUB 180-2, *Secure Hash Standard*[2)]

Open Group, CDE 1.1, *Remote Procedure Call specification, Appendix A*

HL7 V3 Standard, *Data Types — Abstract Specification (R2)*

Regenstrief Institute, Inc. and the UCUM Organization, *The Unified Code for Units of Measure*[3)]

W3C Recommendation, *XML Signature Syntax and Processing*[4)]

3 Terms and definitions

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

3.1
attribute
characteristic of an object that is assigned a name and a type

NOTE The value of an attribute can change during the lifetime of the object.

3.2
class
descriptor for a set of objects with similar structure, behaviour and relationships

3.3
code
concept representation published by the author of a code system as part of the code system, being an entity of that code system

1) Revision of IETF RFC 2806.

2) Revision of FIPS PUB 180-1.

3) Regenstrief Institute, Inc. and the UCUM Organization, Indianapolis, Indiana, USA [viewed 2010-08-23]. Available from: http://aurora.regenstrief.org/ucum.

4) World Wide Web Consortium (W3C) [viewed 2010-08-23]. Available from: http://www.w3.org/TR/xmldsig-core/.

3.4
code system
managed collection of concept identifiers, usually codes, but sometimes more complex sets of rules and references

NOTE They are often described as collections of uniquely identifiable concepts with associated representations, designations, associations and meanings.

EXAMPLES ICD-9, LOINC and SNOMED

3.5
concept
unitary mental representation of a real or abstract thing; an atomic unit of thought

NOTE 1 It should be unique in a given code system.

NOTE 2 A concept can have synonyms in terms of representation and it can be a primitive or compositional term.

3.6
conformance
fulfillment of a specified requirement; adherence of an information processing entity to the requirements of one or more specific specifications or standards

3.7
datatype
set of distinct values, characterized by properties of those values, and by operations on those values

3.8
enumeration
datatype whose instances are a set of user-specified named enumeration literals

NOTE The literals have a relative order, but no algebra is defined on them.

3.9
generalization
taxonomic relationship between a more general class, interface or concept and a more specific class, interface or concept

NOTE 1 Each instance of the specific element is also an instance of the general element. Thus, the specific element has all the features of the more general element.

NOTE 2 The more specific element is fully consistent with the more general element and contains additional information.

NOTE 3 An instance of the more specific element can be used where the more general element is allowed.

3.10
information processing entity
anything that processes information and contains the concept of datatype, including other standards, specifications, data handling facilities and services

3.11
inheritance
mechanism by which more specific elements incorporate structure and behaviour of more general elements

3.12
interface
specifier for the externally-visible operations of class, without specification of internal structure

3.13
invariant
rule about the features of a class which must always be true

3.14
operation
service that an instance of the class may be requested to perform

NOTE An operation has a name and a list of arguments with assigned names and types, and returns a value of the type specified.

3.15
specialization
taxonomic relationship between a more general class, interface or concept and a more specific class, interface or concept where the more specific entity adds new features or redefines existing features by constraining their possible behaviours

3.16
string character set
character set used in all string content throughout this International Standard

3.17
valueSet
that which represents a uniquely identifiable set of valid concept representations, where any concept representation can be tested to determine whether or not it is a member of the value set

NOTE A concept representation can be a single concept code or a post-coordinated combination of codes.

4 Abbreviated terms

For the purposes of this document, the following abbreviated terms apply.

CEN Comité Européen de Normalisation (European Committee for Standardization)

CNE Coded no exceptions

CWE Coded with exceptions

GPD General-Purpose Datatypes

HL7 Health Level Seven, Inc.

IETF Internet Engineering Task Force

OID Object Identifier

OMG Object Management Group

UML Unified Modelling Language

W3C World Wide Web Consortium

XML Extensible Mark-up Language

5 Conformance

5.1 General

An information processing product, system, element or other entity may conform to this International Standard either directly, by utilizing datatypes specified in this International Standard in a conforming manner, or indirectly, by means of mappings between internal datatypes used by the entity and the datatypes specified in this International Standard.

NOTE The term "information processing entity" is used as defined in 3.10, which is consistent with how it is used in ISO/IEC 11404:2007, Clause 4. Specifically, this definition includes applications and also other standards and specifications.

5.2 Direct conformance

5.2.1 Direct conformance definition

An information processing entity which conforms directly to this International Standard shall:

a) specify which of the datatypes specified in Clause 7 are provided by the entity and which are not;

b) define the value spaces of the healthcare datatypes used by the entity to be identical to the value spaces specified by this International Standard;

c) specify to what extent the value spaces of the datatypes are constrained for use within its own context;

d) to the extent that the entity provides operations other than movement or translation of values, define operations on the healthcare datatypes which can be derived from, or are otherwise consistent with, the characterizing operations specified by this International Standard;

e) represent these datatypes using the Extensible Mark-up Language (XML) representation described herein, when the datatypes are represented in XML;

f) optionally, publish a formal conformance profile making these statements clear, or reference one published by some other information processing entity.

The above-mentioned requirements prohibit the use of a type-specifier defined in this International Standard to designate any other datatype (but, see 6.3 concerning the scope of the datatype names). They make no other limitation on the definition of additional datatypes in a conforming entity. For instance, a directly conforming information processing entity could continue to use ISO/IEC 11404 general-purpose datatypes in addition to these healthcare datatypes.

Requirement d) does not require all characterizing operations to be supported and permits the provision of additional operations. The intention is to permit the addition of semantic interpretation to the datatypes, as long as it does not conflict with the interpretations given in this International Standard. A conflict arises only when a given characterizing operation could not be implemented or would not be meaningful, given the entity provided operations on the datatype.

Examples of entities that could conform directly are language definitions or healthcare specifications whose datatypes, and the notation for them, are those defined herein. In addition, the verbatim support by a software tool or application package of the datatype syntax and definition facilities herein should not be precluded.

Information processing entities claiming direct conformance with this International Standard do not always need to use the datatypes defined in this International Standard to represent the concepts, i.e. simply because an address datatype is defined does not mean that the address datatypes must always be used for representing addresses. However, the type defined within this International Standard shall be used where the context is interoperability using these datatypes.

Information processing entities claiming direct conformance with this International Standard may further constrain the value domain of any of the datatypes within their context of use. The conformance statement shall make clear how constraints are applied within the information processing entity and how values that do not conform to the imposed constraints are handled.

Consistency of characterizing operations specified by conforming entities may be assessed by these criteria. Where operations have the same name as the operation defined within this International Standard, they are consistent if the operation can be invoked with the same parameters to return the same result. Other parameters may be defined, but shall have default values or be defined using additional definitions of operations with the same name but other parameter lists.

Information processing entities claiming direct conformance are not required to call any or all of the types defined in this International Standard "types". Other terms such as "data structures" may be used.

5.2.2 Conformance statements

When an information processing entity claims direct conformance with this International Standard, it should make a conformance statement.

It is anticipated that other standards bodies would make conformance statements with regard to this International Standard both in a general sense and in the sense of adopting these datatypes for a particular standard. In addition, it is anticipated that certain countries publish profiles of these datatypes on either an advisory or a normative basis. Finally, vendors and purchasers of healthcare applications may well find use in creating, sharing and publishing these conformance statements.

This International Standard makes no rules about either the form of the statement or how it is published, but it should be clearly and formally presented and made available to all interested parties associated with the scope of the information processing entity.

In addition to specifying that conformance statements shall contain formal statements pertaining to 5.2.1 a) to d), this International Standard makes additional rules about what they shall or should say or may choose to do.

5.2.2.1 Direct conformance statements shall:

a) define which character set and encoding applies; the default is Unicode (see 6.7.5);

b) *if* an alternative mechanism for providing history and audit data is provided, define how it maps to the history and audit information on datatypes (see 7.1.3);

c) make clear how attribute and collection cardinality are specified (see 7.1.5);

d) define how the attributes nullFlavor, updateMode and flavorId on ANY are managed (see 7.3.3);

e) *if* quantities are used, make clear exactly how and when the QTY attributes expression, originalText, uncertainty and uncertaintyType are used;

f) make clear what methods may be used to provide alternative definitions for discrete set uniqueness (see 7.9.3);

g) *if* the structured documents types are used, document the scope of the document context and clearly define how references within this document context are resolved (see 7.12);

h) specify to what degree the XML format is adopted and define the namespace that is used (see A.1).

5.2.2.2 Direct conformance statements should:

a) define defaulting rules for language (see 7.4.2.3.7);

b) declare what languages are supported in the QTY.expression property (see 7.8.2.3.1);

c) describe which codes may be used in QSC.code (see 7.10.8.3);

d) *if* the structured documents types are used, define how version tracking works in the contexts where it is used (see 7.12.12.2.1).

5.2.2.3 Direct conformance statements may also:

a) define additional datatype flavors or additional authorities for the definition of flavors (see 6.7.6);

b) make additional arrangements for the use of derived data and the DER NullFlavor (see 7.1.4);

c) define how the controlInformationRoot and controlInformationExtension properties on HXIT are used (see 7.3.2.3.4);

d) clarify how telecommunication and postal addresses are selected for particular purposes (see 7.6.2.3.2);

e) define the code systems to which different name and address part types are bound (see 7.7.3.6 and 7.7.5.6).

5.3 Indirect conformance

5.3.1 Indirect conformance definition

An information processing entity which conforms indirectly to this International Standard shall:

a) provide mappings between its internal datatypes and the healthcare datatypes conforming to the specifications of Clause 7;

b) specify for which of the datatypes in Clause 7 an inward mapping is provided, for which an outward mapping is provided and for which no mapping is provided;

c) specify whether the XML representation described in this International Standard is used when the datatypes are represented in XML, or whether it is used optionally to provide an alternative namespace for the XML representation;

d) optionally, publish a formal conformance profile making these statements clear or reference one published by some other information processing entity.

Examples of entities which could conform indirectly are healthcare specifications, applications, software engineering tools and other interface specifications, and many other entities that have a concept of datatype and an existing notation for it.

Standards for existing healthcare specifications yet to be proposed as International Standards are expected to provide for indirect conformance rather than direct conformance.

Information processing entities claiming indirect conformance with this International Standard do not always need to use the datatypes defined in this International Standard to represent the concepts, i.e. simply because an address datatype is defined does not mean that the address datatypes must always be used for representing addresses. However the type defined within this International Standard shall be used where the context is interoperability using these datatypes.

Information processing entities claiming indirect conformance with this International Standard may further constrain the value domain of any of the datatypes within their context of use. The conformance statement must make clear how constraints are applied within the information processing entity and how values that do not conform to the imposed constraints are handled.

Information processing entities claiming indirect conformance are not required to call any or all of the types defined in this International Standard "types". Other terms, such as "data structures" may be used.

5.3.2 Conformance statements

When an information processing entity claims indirect conformance with this International Standard, it should make a conformance statement.

This International Standard makes no rules about either the form of the statement or how it is published, but it should be made available to all interested parties associated with the scope of the information processing entity.

In addition to specifying that conformance statements shall contain formal statements pertaining to 5.3.1 a) to d), this International Standard makes additional rules about what they shall or should say or may choose to do.

5.3.2.1 Indirect conformance statements shall:

a) define which character set and encoding applies; the default is Unicode (see 6.7.5);

b) make clear what equality definitions apply and how (see 7.1.2);

c) make clear how attribute and collection cardinality are specified, if relevant (see 7.1.5);

d) *if* the structured documents types are used, document the scope of the document context and clearly define how references within this document context are resolved (see 7.12).

5.3.2.2 Indirect conformance statements should:

a) define defaulting rules for language (see 7.4.2.3.7);

b) if any exist, declare the mapping between W3C digital signature and alternate implementations (see 7.4.5.1).

5.3.2.3 Indirect conformance statements may also:

a) define additional datatype flavors or additional authorities for the definition of flavors (see 6.7.6);

b) make additional arrangements for the use of derived data and the DER NullFlavor (see 7.1.4);

c) define how the controlInformationRoot and controlInformationExtension properties on HXIT are used (see 7.3.2.3.4);

d) clarify how telecommunication and postal addresses are selected for particular purposes (see 7.6.2.3.2);

e) define the code systems to which different name and address part types are bound (see 7.7.3.6 and 7.7.5.6);

f) declare what languages are supported in the QTY.expression property (see 7.8.2.3.1);

g) describe which codes may be used in QSC.code (see 7.10.8.3);

h) *if* the structured documents types are used, define how version tracking works in the contexts where it is used (see 7.12.12.2.1).

6 Datatypes overview

6.1 What is a datatype?

In ISO/IEC 11404, a "datatype" is defined as a set of distinct values, characterized by properties of those values, and by operations on those values (ISO/IEC 11404:2007, 3.12).

A datatype consists of three main features:

— a value space;

— a set of properties;

— a set of characterizing operations.

Generally, the definitions of the scope of datatypes revolve around one or other of the following notions.

— Immutability (the properties of the datatype cannot change, instead a new instance is created: datatypes have no lifecycle).

— The relationship between equality and identity (if two datatypes are equal they are the same instance).

— Coherency of a single concept (each datatype should represent a single concept space).

Since the application of these concepts to the healthcare information domain and the implications of these for the scope of datatypes are inherently a matter of perspective, the selection criterion for the datatypes defined in this International Standard is based on the set that has emerged from the debates held within the various stakeholder standardization bodies that define healthcare information standards. Since healthcare information standards and specifications are expected to provide mappings to this International Standard, the process has been deliberately inclusive. These other standards may choose to represent these datatypes with other more complex structures, but should explain how to interconvert these structures with the datatypes defined herein.

6.2 Definitions of datatypes

This International Standard defines a set of named datatypes. Each datatype defined in this International Standard is allocated both a short name and a long name. The formal name of the datatype is the short name. Each datatype is defined in two different ways:

— in terms of the datatype specification language and types defined in ISO/IEC 11404;

— in UML, using primitive types taken from the UML kernel package.

The ISO/IEC 11404 definition is provided to ensure continuity between this International Standard and the ISO/IEC 11404 GPDs, while the UML definition is provided to foster software-driven implementation of these datatypes. The ISO/IEC 11404 definitions are semantic and abstract in nature, while the UML definitions are concrete structural definitions. This International Standard is focused on providing structural concrete definitions, such that the UML definitions take precedence over the ISO/IEC 11404-based definitions, which are provided in the interests of continuity with ISO/IEC 11404.

The datatypes defined in this International Standard are an implementation of the HL7 V3 Abstract Data Types (R2). What this means is that it is possible to implement the exchange of information based on the HL7 V3 Abstract Data Type definitions using the datatypes defined in this International Standard. Annex B demonstrates how these datatypes are implementations of the HL7 V3 Abstract Data Types (R2).

The datatypes defined in this International Standard are not restricted to the features described by the HL7 V3 Abstract Data Types, nor is the HL7 V3 Data Types Abstract specification required in order to make use of these datatypes. The semantic definitions in the HL7 V3 Abstract Data Types may be consulted for further useful information to help implementors understand the use of these datatypes.

6.3 Datatype names and re-use of common datatype names

Some of the names of these datatypes bear superficial similarity to similar datatypes defined in other specifications. For instance, this International Standard defines a type REAL and there is a type Real in ISO/IEC 11404, and a type called Real in the UML kernel (see Note 1 in B.2.7 concerning the use of floating value types).

This International Standard is not attempting to redefine or replace the definition of real in ISO/IEC 11404 or UML. Instead, a new type that wraps the underlying "primitive" type is defined, which builds on the functionality of the underlying type and fits it into the overall architectural framework.

There are many specifications, languages and implementation technologies that declare similar types to either the underlying real types or the REAL defined in this International Standard, with a profusion of names around the common theme of Real, Float, Decimal or Double.

This International Standard does not seek to find new previously unrecognised names for the types it declares that represent these base concepts; instead it just assigns commonly used names to them. Again, this International Standard is not attempting to redefine or replace any other definitions by using these common names.

For these name clashes between the datatypes defined in this International Standard and any other specification, implementers should use some form of namespacing to ensure that the names of the datatypes do not cause confusion, perhaps by prefixing the names with some string constant in implementation environments that do not support proper namespacing of types.

The types BL, ST, INT, REAL, SET, LIST and BAG defined in this International Standard use this "primitive type wrapper" pattern.

6.4 Mapping to this datatypes International Standard

Like ISO/IEC 11404, this International Standard anticipates the use of these datatypes within other specifications. These specifications must specify how the datatypes and features described in this International Standard are implemented within the specification. Datatypes may be adopted and used directly, or they may be mapped to other datatypes or structures in different places, or they may not be supported at all.

Each specification that uses these datatypes should publish a document or section describing the mapping, and providing assistance for implementors to interconvert data between specifications.

6.5 Conformance with ISO/IEC 11404

This International Standard asserts direct conformance with ISO/IEC 11404. Although this International Standard can be considered to provide support for all the general purpose datatypes, only the following types are actually used in this International Standard:

— boolean;
— enumerated;
— characterstring;
— integer;
— Real;
— class;
— set;
— bag;
— sequence;
— octet.

The healthcare datatypes are class constructs built using these base primitive types. The healthcare datatypes are partially defined using the datatype definition language defined in ISO/IEC 11404. Since this language does not provide for generalization/specialization relationships, the datatypes cannot be fully defined in the ISO/IEC 11404 language, since the generalization/specialization relationships are an important part of the definition of the datatypes.

6.6 Reference to UML 2

This International Standard defines the datatypes using the UML. The datatypes are all specializations of the UML Classifier, and the types are defined in this International Standard or are built on the following UML Kernel types as defined in the OCL 2 specification:

— enumeration;

— boolean;

— integer;

— string;

— collection;

— sequence;

— set;

— bag.

6.7 Modelling of datatypes

6.7.1 General

This International Standard presents the relationship between individual datatypes using the facilities of the Unified Modelling Language (UML) version 2. This modelling technique provides a means whereby:

— properties that are common across groups of datatypes can be expressed once;

— a mechanism is provided whereby one datatype within a specification may be substituted by another.

6.7.2 Attribute definitions

Unless otherwise specified, the default value for all attributes and associations is nil.

6.7.3 Generalization/specialization

This International Standard defines a series of class representations of healthcare datatypes. Within these classes, a number of generalization/specialization relationships are defined. This has the normal meaning associated with it as defined in the UML standard, and any instance of a class may be replaced by an instance of a specialization of that class. However, some of the specifications that rely upon this International Standard may make additional constraints concerning which specializations are permissible in a given context.

6.7.4 Enumeration definitions

This International Standard defines a number of attributes that have an enumerated set of possible values. Each value in the enumeration represents a concept in a terminology. Within the terminology, there may be generalization/specialization relationships. In this International Standard, the enumerations are defined in three ways:

— a list of codes as an ISO/IEC 11404 enumeration;

— a list of codes as a UML enumeration definitions;

— a table defining the enumeration in the narrative of the standard.

The table has four rows: Level, Code, Title and Definition.

Level	The level of the concept in the hierarchy of the terminology. All concepts marked with the same level and not separated by a concept with a level of less numerical magnitude are siblings, and all concepts that follow after another concept with a higher level value are children of that concept.
Code	The code that represents the concept. This is used in the enumerations and to identify the concept in any representation or exchange of data in this International Standard. The Code is indented to represent the hierarchy of the terminology.
Title	A short human-readable description of the concept.
Definition	A short definition of the intention of the concept.

The hierarchy in the enumeration is an important part of the specification. Although the enumerations are defined as linear lists within ISO/IEC 11404 and the UML definitions, any information processing entity that asserts direct or indirect conformance with this International Standard shall conform to the relationships when evaluating meaning (implication) within the enumeration. In addition, this International Standard occasionally refers to the relationships within the narrative when defining the outcome of some operations.

Except in the case of the AddressPartType enumeration, the hierarchies represent generalization/specialization (also known as subsumption). In these hierarchies, the children codes represent a more specialized meaning of the parent code. The AddressPartType enumeration is compositional in nature; here codes represented as child codes of another code represent parts of the concept represented by the parent code.

As an example, the following is a subset of the table that defines the NullFlavor Enumeration

NullFlavor Enumeration. OID: 2.16.840.1.113883.5.1008			
Level	Code	Description	Definition
1	NI	No information	The value is exceptional (missing, omitted, incomplete, improper). No information as to the reason for being an exceptional value is provided. This is the most general exceptional value. It is also the default exceptional value.
2	UNK	Unknown	A proper value is applicable, but not known.
3	ASKU	Asked but unknown	Information was sought but not found (e.g. patient was asked but did not know).
4	NAV	Temporarily unavailable	Information is not available at this time but it is expected that it would be available later.
3	NASK	Not asked	This information has not been sought (e.g. patient was not asked).
2	MSK	Masked	There is information on this item available but it has not been provided by the sender due to security, privacy or other reasons. There may be an alternate mechanism for gaining access to this information. Warning: Using this Nullflavor does provide information that may be a breach of confidentiality, even though no detailed data are provided. Its primary purpose is for those circumstances where it is necessary to inform the receiver that the information does exist without providing any detail.
2	NA	Not applicable	No proper value is applicable in this context (e.g. last menstrual period for a male).

In this table, all the concepts listed below NI are indented and marked with a higher level, therefore they are all specializations of NI. Codes ASKU, NAV and MASK are all specializations of the concept UNK, and codes UNK, MSK and NA are siblings, specializations of NI, but not of anything else. Consequently, the eumeration value ASKU implies that the enumeration value UNK is also applicable.

All the enumerations in this International Standard are maintained by HL7, unless otherwise specified. Revised tables are published on a regular basis. The values defined in this International Standard will not have their meaning changed, though they may be deprecated. When these revised tables are published by HL7 or ISO, new enumeration values may be pre-adopted by trading partner agreement prior to the issuance of a new edition of this International Standard.

The OID for the HL7 codeSystem is published for reference, and to assist when passing these enumerated codes to terminology subsystems which are likely to use the OID to unambiguously refer to the code.

6.7.5 Strings and character encoding

This International Standard makes reference to both the ISO/IEC 11404 characterstring, and the UML Kernel String. For the purposes of this International Standard, these types define the same functionality: an immutable sequence of known length containing zero or more logical characters. This type is hereinafter referred to in the narrrative as simply "String".

NOTE 1 Both ISO/IEC 11404 and the UML kernel define additional characterizing operations which might be useful for implementors, and which can be considered to apply, but are not directly of interest to this International Standard.

NOTE 2 This International Standard also declares a wrapper type for String called ST, which adds additional functionality related to how the notion of an immutable sequence of characters fits into the overall framework of healthcare datatypes with their associated notions to do with uncertainty, unreliability and conformance.

The String datatype contains a sequence of logical characters, which is different from carrying a sequence of bytes that encodes a sequence of logical characters.

NOTE 3 Implementers should consider the difference between the two concepts carefully when implementing these datatypes.

By default, the String type contains Unicode characters. Information processing entities claiming direct or indirect conformance should mandate that the unicode character set be used in all String types throughout this International Standard.

However, there a few character sets throughout the world which are not perfectly mapped into Unicode. For this reason, there are a few countries or regulatory domains which may mandate some other character set than Unicode. In these contexts, standards and specifications that claim direct or indirect conformance would support the use of character sets other than Unicode, and should be explicit about which character sets are supported and how they are represented.

NOTE 4 There is an obvious implementation cost to choosing to use something other than Unicode. Regulatary domains that choose to use something other than Unicode generally have considered this issue at length.

The character set for any given implementation environment, whether Unicode or something else, is referred to throughout this International Standard as "the String Character Set“.

Whether the String Character Set is Unicode or something else, any given encoding that serves the implementation of the primitive string type in a particular character set may include control bytes that alter the interpretation of the text. It is assumed that any operations performed take this into consideration. However, as the operations described in this International Standard are enacted upon the logical sequence of characters, this issue is not discussed any further.

6.7.6 Flavors

In addition to the basic datatypes, this International Standard also defines a number of datatype flavors not independent datatypes in their own right – they are not defined as UML classes, nor as XML schema types. Instead, Datatype flavors describe common constraint patterns on the datatypes. As such, datatype flavors cannot introduce new attributes, new codes, default values, or any new defintional material. Datatype flavors may only make rules about what or how existing features of a class may be used.

Since datatype flavors may not introduce new features or meaning, and since they do not exist as independent classes in their own right, information processing entities do not need to understand a flavor in order to process the information correctly. For this reason, any information processing entity declaring direct or indirect conformance with this International Standard is able to define datatype flavors or reference flavors defined by some other authority, as long as the standard naming rules are followed. These are:

— names shall consist of a sequence of valid characters, namely letters, digits, underscores and periods; non-whitespace unicode characters may be used at the descretion of the information processing entity;

— names shall begin with the name of the type from which they are derived, followed by a period, a namespace, another period and then some additional valid characters;

— namespaces are used to prevent flavors described by different sources from having clashing names;. the namespace should either be an ISO 3166-1 country code, the applicable HL7 Realm identifier, or a DNS name.

Examples:

TS.CA.BIRTH	Rules for dates of birth published by the relevant Canadian Authority
TS.NPFIT.NHS_NUMBER	NHS Number flavor (fixed root) published by NPFIT in UK
ED.AU.KESTRAL.DOCUMENT	Rules for acceptable document format for the Australian company "Kestral"

Datatype flavors defined in this International Standard do not need or have a namespace.

Applications should not reject an instance because the flavorId attribute references a flavor that is not known to the application. Applications may reject an instance that references a flavor to which the instance does not conform, but are not required to do so.

The flavors defined in this International Standard do not need to be implemented by or used in association with information processing entities declaring direct or indirect conformance with this International Standard.

6.7.7 Examples

Examples are provided for most datatypes. The examples serve to illustrate various points related to how the datatypes are used.

The examples are all given as XML examples following the form documented in Annex A. The examples are presented assuming that the XML document/element that contains them has a charset of UTF-8. Most examples include an xsi:type making their type explicitly clear, but this is generally not necessary where the type is fully specified by the context of use and/or schema.

The discussion in the examples may provide references to content published by standard development organizations other than ISO or HL7. These reference materials are not a normative part of this International Standard.

7 Datatypes

7.1 General properties

7.1.1 Immutability

These datatypes are conceptually immutable. The datatypes are defined with no lifecycle, and no operations that allow the existing datatype to change are defined. However, the datatypes are defined in this International Standard as classes with attributes, which allows for the value of the datatype to be modified after it is created.

Although this may be useful for implementation, it is not the intent of this International Standard to define types that are semantically mutable. Specifications that use these datatypes must consider the types themselves immutable. In particular, implementers shall be diligent to prevent the aliasing effects that may arise from allowing the datatype properties to change once they are in use.

7.1.2 Equality

There are two aspects to equality *viz* "are these two data values the same instance?" or "do these two data values represent the same semantic concept?".

In UML/OCL there are two properties to reflect these two aspects of equality. The first is "=", which returns whether these two values are the same instance, and the second is "equals", which returns whether these types represent the same concept. For OCL primitive types, such as integer, the two properties return the same value. For UML Classifiers, the results may be different.

In this International Standard, the equality criteria for each datatype are specified. This specifies the second form of equality: do these two data values represent the same concept? These definitions of equality are carefully constructed to meet the criterion that equal operations shall be reflexive, symmetric, and transitive, thus:

reflexive	x equals x must be true
symmetric	If x equals y is true, then y equals x must be true
transitive	If x equals y, and y equals z, then x must equal z

Because this form of equality is inherently a semantic notion, the equality definitions may depend on the semantics of the type, and it may not be simple to determine. Information processing entities claiming direct conformance with this International Standard shall conform to these equality definitions. Information processing entities claiming indirect conformance shall make clear what equality definitions apply.

Datatype flavors do not change the definition of equality.

7.1.3 History and audit trail

The base type HXIT defines properties for specifying a validTime during which the value is, was or will be valid, and for specifying the identity of an event that was associated with the changes (known as control information).

The validTime is not an audit trail of the information, used to track when any particular systems associated particular versions of data with a concept; it is used to make statements about the time period during which the data item was a correct description of the concept. For instance, in many countries a person may change their name by marriage or other legal means, and so a particular name is only associated with a person for a limited time. Similarly, a person's addresses and contact details may change as they move.

The control information is the identity that links to some event in an information system which is associated with “control” information concerning the change to this data value on a system. This information specifically relates to the association of data with its concept in systems. The control information reference can be used to build an audit trail of values across multiple exchanges between systems. See 7.3.2 for more details.

The various specifications that make use of this International Standard may provide alternative mechanisms for specifying this history and audit data, particularly the second part relating to system audit trails. In such cases, the specification will declare, usually in its conformance statement, how such information is handled and potentially mapped into the properties defined in this International Standard.

Datatypes are immutable – their value cannot change. The concept of valid time and control act does not apply within a datatype. For this reason, whenever a datatype is re-used as the type of an attribute of another datatype, the invariants will specify that the valid time and control act attributes shall be null.

7.1.4 Null and NullFlavor

The base type ANY introduces a concept called nullFlavor. Though the nullFlavor concept has some relationship with the UML/OCL null, it is not the same thing, and the relationship and differences between the two shall be understood to properly implement this International Standard.

Any instance of a class defined in this International Standard may be null. This is the null defined in UML and OCL. A null instance is an instance of the type OclVoid, and conforms to all types. It carries no other information other than the fact it is null. This International Standard uses the ocl operations oclIsDefined and oclIsUndefined to make constraints on the use of this form of null in attributes of the types defined in this International Standard.

Alternatively, an instance of the class may be created, and its nullFlavor can be set to one of the NullFlavors.

In this case, the value is referred to as nullFlavored, or exceptional. The intent of this is that the value represents an exception to the normal value domain of the type. This does not mean that an instance representing exceptional value is not bound by the rules defined in this International Standard; it must still meet the invariants defined herein. However many of the rules are different for exceptional values as they represent semantic exceptions to the normal data. All exceptional values shall have a NullFlavor, and the nullFlavor provides more information as to why the value is an exception to the rules. The opposite of an exceptional value is a proper value: this is a value with no nullFlavor.

Since an instance of the type must be created in order to carry a nullFlavor, it can also have values assigned to its other attributes. This is different from the UML/OCL null which does not exist and cannot have any nonNull attributes. If a nullFlavor is present, other attributes may be populated, but there is no requirement that any information processing entity make any use of these values, except that the following cases should be properly understood where applicable:

— NullFlavor OTH on CD;

— NullFlavors NINF and PINF on Interval low and high boundaries, respectively;

— NullFlavor UNK on an II with an extension and no root.

NullFlavor Enumeration. OID: 2.16.840.1.113883.5.1008			
1	NI	No information	The value is exceptional (missing, omitted, incomplete, improper). No information as to the reason for being an exceptional value is provided. This is the most general exceptional value. It is also the default exceptional value.
2	INV	Invalid	The value as represented in the instance is not a member of the set of permitted data values in the constrained value domain of a variable.
3	OTH	Other	The actual value is not a member of the set of permitted data values in the constrained value domain of a variable (e.g. concept not provided by required code system).
4	PINF	Positive infinity	Positive infinity of numbers.
4	NINF	Negative infinity	Negative infinity of numbers.
3	UNC	Unencoded	No attempt has been made to encode the information correctly but the raw source information is represented (usually in originalText).
3	DER	Derived	An actual value may exist, but it must be derived from the information provided (usually an expression is provided directly).
2	UNK	Unknown	A proper value is applicable, but not known.
3	ASKU	Asked but unknown	Information was sought but not found (e.g. patient was asked but didn't know).
4	NAV	Temporarily unavailable	Information is not available at this time, but it is expected that it will be available later.
3	NASK	Not asked	This information has not been sought (e.g. patient was not asked).
3	QS	Sufficient quantity	The specific quantity is not known, but is known to be non-zero and is not specified because it makes up the bulk of the material. 'Add 10 mg of ingredient X, 50 mg of ingredient Y, and sufficient quantity of water to 100 ml.' The null flavor would be used to express the quantity of water
3	TRC	Trace	The content is greater than zero, but too small to be quantified.
2	MSK	Masked	There is information on this item available, but it has not been provided by the sender due to security, privacy or other reasons. There may be an alternate mechanism for gaining access to this information. **WARNING — Use of this null flavor does provide information that may be a breach of confidentiality, even though no detailed data are provided. Its primary purpose is for those circumstances where it is necessary to inform the receiver that the information does exist without providing any detail.**
2	NA	Not applicable	No proper value is applicable in this context (e.g. last menstrual period for a male).

ISO/IEC 11404 defines the concept of a sentinel value, which is a value in the value space of the type that does not share in all the characterizing operations of the type. Though there are some conceptual similarities between nullFlavors and sentinel values, instances of type ANY with nullFlavors are not sentinel values. The

characterizing operations still apply, though the result of these characterizing operations will be some flavor of null. This is like the OCL null behaviour, behavioural similarity is why the property is named "nullFlavor".

Unless specifically documented otherwise, all operations defined in this International Standard behave in the same fashion.

- If the operation is performed on a UML/OCL null value, the result is a UML/OCL null.
- If the operation is performed on a value with a NullFlavor.
  - If the operation takes no parameters, it will return a NullFlavor. Usually the NullFlavor will be NA, but other NullFlavors may be appropriate, depending on the semantics of the NullFlavors. When performing operations upon nullFlavored values, the semantic meaning of the NullFlavor shall be considered.
  - If the operation takes parameters, and any of the parameters are a UML/OCL null, the result will be a UML/OCL null.
  - Where operations involve values that are null or have NullFlavors, the resulting value will be null or have a NullFlavor unless the semantics of the datatypes and NullFlavors dictate otherwise. There are a few rules about specific operations in this International Standard.
- If the operation is performed on a proper value.
  - If the operation takes parameters, and any of the parameters are a UML/OCL null, the result will be a UML/OCL null.
  - Where operations involve values that are null or have nullFlavors, the resulting value will be null or have a nullFlavor unless the semantics of the datatypes and nullFlavors dictate otherwise. There are a few rules about specific operations in this International Standard.
  - Otherwise, the operation will perform as described.

Some specific operations deviate from these rules. This is documented for each operation.

One special case arises with equality comparisons of various nullFlavored values. Two values that have NullFlavor NotApplicable are considered equal. While NINF cannot be equal to NINF and PINF cannot be equal to PINF, since the actual value is not known, NINF and PINF are clearly not equal. In other cases, it is generally not safe for the comparison to return anything but a NullFlavor – usually NI.

A value of any type with a NullFlavor of NI and where all the other attributes are null or meet this rule recursively is semantically equivalent with a UML/OCL null, and these forms may be interconverted if desired. Most simple attributes are declared using a UML or OCL type, such as String, and these may be converted to the complex equivalent, such as ST, with a NullFlavor of NI, if desired.

Because an attribute may be either null or a nullFlavor, many of the invariants take the form (x.oclIsDefined and x.isNotNull) implies {condition}. For some invariants, it is not necessary to make this rule since the result of null (the value of x.isNotNull if x is actually null) implies anything is null, and the invariant will fail gracefully, but for others the null must be protected against in order for the correct outcome in all cases.

The nullFlavor concept provides a general framework for handling incomplete data which is often encountered in healthcare information collection, use and analysis. The nullFlavor property may also play a special role in conformance frameworks in specifications that make use of these types.

Not all the NullFlavors can be used with all the different types. The NullFlavors PINF and NINF may only be used with specific types (INT, REAL, PQ and TS). The NullFlavor UNC may only be used with types that have an originalText (CD, QTY, QSET, and specializations). The NullFlavors QS and TRC may only be used with PQ.

The two NullFlavors OTH and INV with their other specializations draw a distinction between the actual value and the value as represented in the instance. Some of the datatypes may be used to provide a representation of the value which requires subsequent transformation to generate the real value. For instance, an expression may be provided which generates an actual value that is in the required value domain of the instance, or an uncoded CD value – with just an originalText. INV, DER and UNC offer the possibility that some transform – either based on additional information or knowlege – may generate a valid value, whereas OTH with its specializations is an assertion that it is believed that no better value exists. This invites questions of confidence – how confident is the source that no better information exists, how sure is the processor that it believes the source is correct? However these are not resolvable. For this reason, the difference should not be taken as absolute, but as a statement of intent from the source.

Although INV and its specializations represent exceptional values in their context of use, they are only exceptional within the parameters defined by this International Standard. Actual values shall always conform to the rules defined by this International Standard.

The NullFlavor DER shall only be used where the context of use makes clear how the actual value can be derived from the information provided. This International Standard provides QTY expression (see 7.8.2.3.1). Other information processing entities may make other arrangements for the use of DER in their conformance statements.

NOTE Most often, the correct NullFlavor to use is NI, and there need not be any burden on implementors with regard to choosing or persisting the correct NullFlavor. Other than some technical requirements that are clearly documented in this International Standard regarding the use of the NullFlavors NA, INV, DER, OTH, PINF and NINF, whenever it is not clear which NullFlavor is applicable, implementers should be comfortable using the NullFlavor NI. In particular, if a user does not respond to an input field in a data collection procedure, or the data are missing for some unknown reason, NI would be the most appropriate NullFlavor to choose.

EXAMPLES

Use Case	NullFlavor of Choice
User does not respond to input on a screen form	NI
Source is not configured to encode plain text input to required codeSystem	UNC
Source is unable to encode this particular plain text input to the required codeSystem because it cannot match the text	OTH
Patient is unconscious and cannot provide name	NAV
The system does not support this element	NI
No proper dosage is provided, but an expression is provided so that the destination system can calculate the proper dosage from the patient's weight	INV
The patient does not have an address – No Fixed Place of Abode	NA
Reporting the duration of an adverse reaction that is ongoing using an IVL<TS>	IVL.high = NA because the reaction is ongoing – the concept of high does not apply
Reporting the duration of an adverse reaction using an IVL<TS> when it is not known whether the reaction has terminated	IVL.high = UNK – we do not know
The source system is responding to a query for patient details, and has decided not to include the address because of applicable security and/or privacy policy.	MSK
NOTE Normal security/privacy policy is not to inform the information recipient that information has been suppressed, for good reason. However there are a few cases where workflow reasons dictate that it is necessary to inform the user that information has been denied. The MSK NullFlavor is provided for these minor cases.	

7.1.5 Conformance

Conformance, as discussed in Clause 5, is concerned whether information processing entities conform to this International Standard. There is an additional aspect, which is also often called conformance, which concerns which rules are applied to the datatypes by information processing entities such as other standards when the datatypes are used.

Any information processing entity that uses these datatypes may constrain their use by making some human language narrative statements concerning how they are used, or by using some formal language statement in a processible language such as OCL. In addition, this International Standard recognises two additional means by which the possible values of the datatypes can be constrained, called "mandatory" and "cardinality".

Any external attribute which is assigned a type from this International Standard may also have a nominal flag "mandatory" set to true. If the context of use sets this flag to true, the instance shall contain a valid data value that is not null, has no nullFlavor, conforms to all the constraints stated in this International Standard and any additional constraints on the value domain stated in the model. If this flag is not set to true, and the instance does not meet the constraints specified in the constraining model, the instance shall either be labelled using some form of nullFlavor (though other information may still be provided), or shall be omitted completely, in which case the default value (usually NullFlavor NI) applies.

The context of use may also apply a cardinality to an attribute. A cardinality consists of a minimum value, specified as a whole number, and a maximum value, specified as a whole number or "*". The cardinality is usually presented as [minimum value]..[maximum value], e.g. 0..1 or [1..*]. The meaning of the cardinality differs between collection based attributes and other attributes.

For attributes with a collection type (<dtref ref="dt-COLL"/> and its specializations), the cardinality specifies how many items may be in the collection. A cardinality maximum value of * means that there is no limit to the number of items in the collection.

NOTE 1 This does not imply that information processing entities are required to handle infinitely large collections of data, but the specification itself places no limit on the size of the collection). The minimum value specifies how many items must be in the collection.

NOTE 2 In the case of a mandatory collection, the collection shall contain at least one non-null (not null, and no nonFlavor) item.

For other attributes, the only cardinalities that may be applied are 0..0, 0..1, and 1..1. Cardinality of 0 means that the attribute is not to be represented in the instance, and has an implicit NullFlavor of NI. Cardinality of 1 means that the attribute has a value, though the value may be a NullFlavor unless the attribute is also mandatory.

A mandatory attribute shall have a minimum cardinality of 1 or more.

NOTE 3 This use of cardinality is a little different than the standard use of cardinality on attributes in UML. In UML, if an attribute is assigned a type of DSET(CS) and a cardinality of 2..3, this means that there must be 2 or 3 sets of CS. These uses are not incompatible; both forms of cardinality may be applied in any information processing entity that claims direct or indirect conformance. Which form is intended should be made clear in the documentation.

7.2 Top level model

For convenience and reference, Figure 1 provides an overview of the datatypes defined in this International Standard as a UML diagram (see Figure 1).

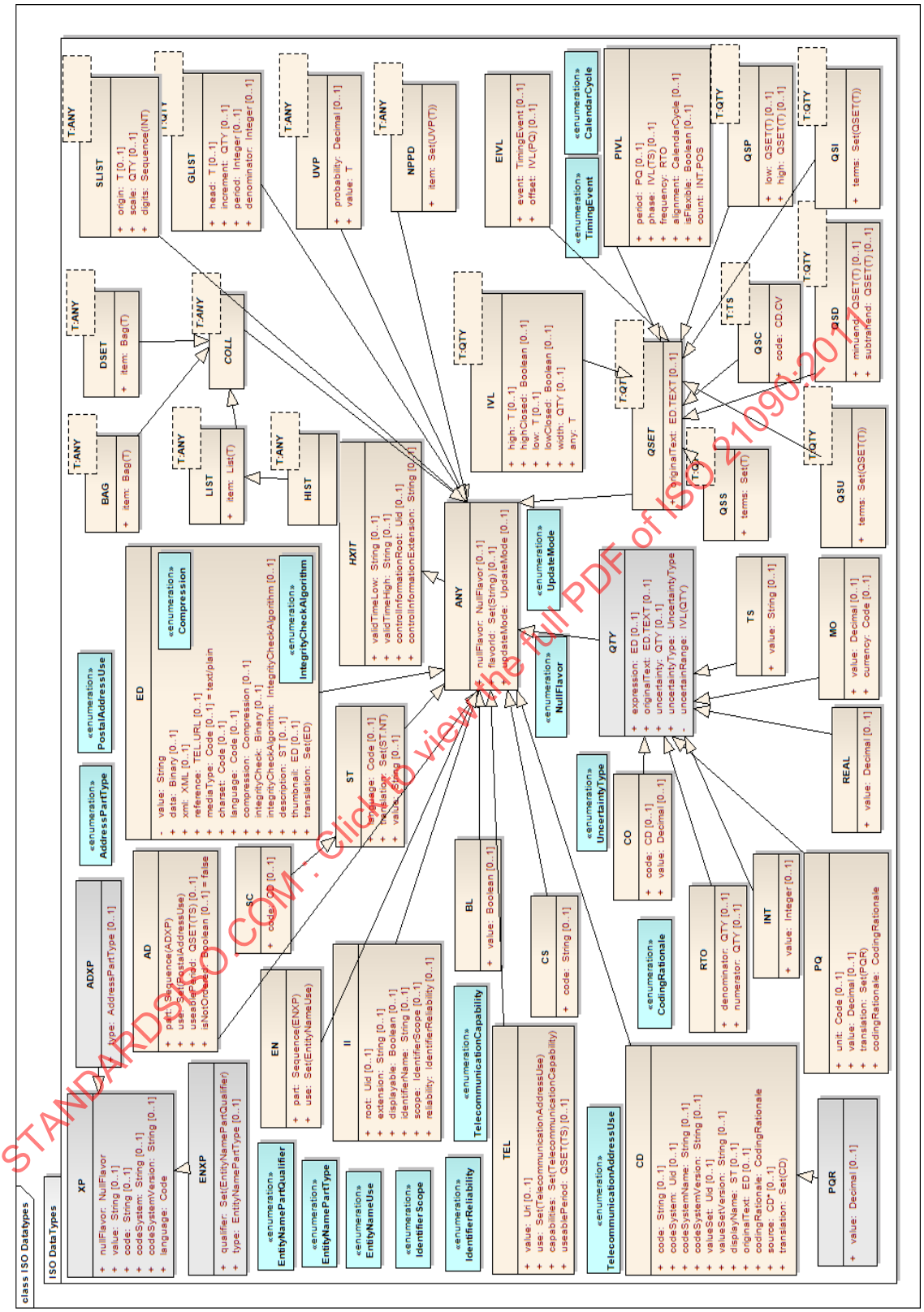

Figure 1 — Top level modelBasic datatypes

7.3 Basic datatypes

7.3.1 Overview

Basic datatypes are datatypes that provide infrastructural support for specific datatypes, which are defined in subsequent sections (see Figure 2).

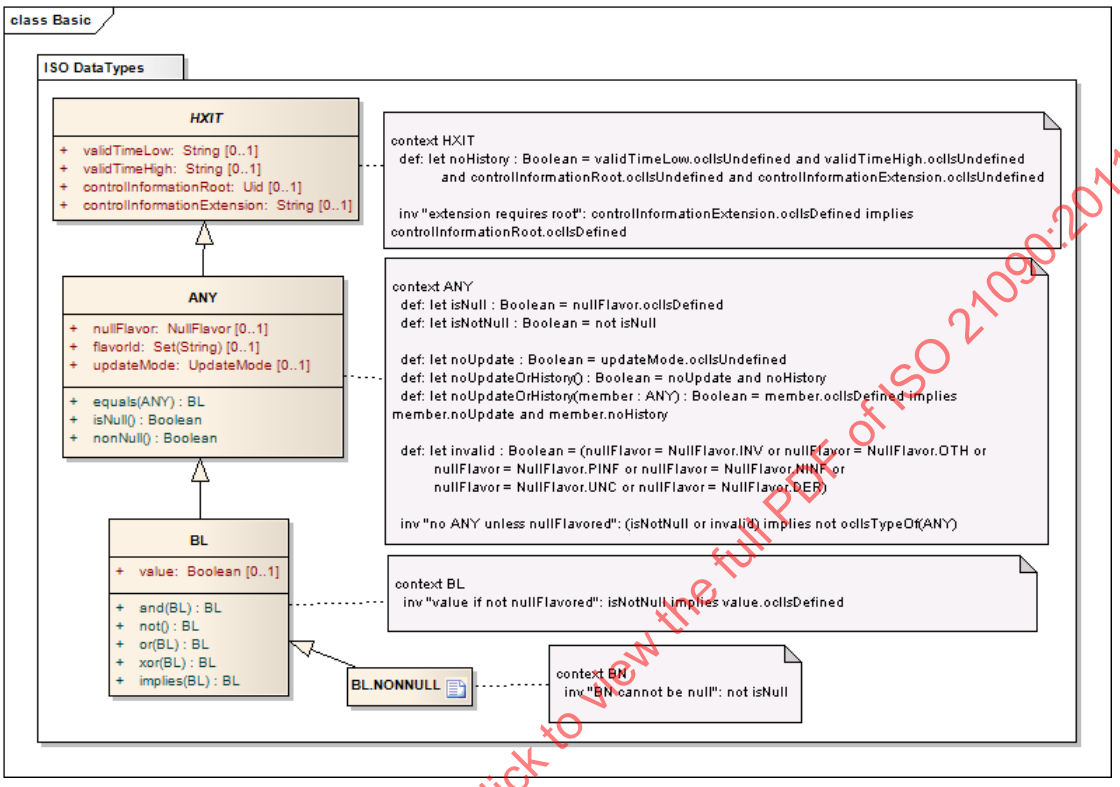

Figure 2 — Basic datatypes

7.3.2 HXIT

7.3.2.1 Description

Abstract and private – this datatype is not for use outside the datatypes in this International Standard.

Information about the history of this value: period of validity and a reference to an identified event that established this value as valid.

Because of the way that the types are defined, a number of attributes of the datatypes have values with a type derived from HXIT. In these cases the HXIT attributes are constrained to null. The only case where the HXIT attributes are allowed within a datatype is on items in a collection (DSET, LIST, BAG, HIST).

The use of these attributes is generally subject to further constraints in the specifications that make use of these types.

7.3.2.2 ISO/IEC 11404 Syntax

```
type HXIT = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring
)
```

7.3.2.3 Attributes

7.3.2.3.1 validTimeLow : String: The time that the given information became or will become valid.

This is not the time that any system first observed the value, but the time that the actual value became correct (i.e. when a patient changes their name).

7.3.2.3.2 validTimeHigh : String: The time that the given information ceased or will cease to be correct.

Both validTimeLow and validTimeHigh shall be valid timestamps using the format described in 7.8.13.3.1 (TS.value).

7.3.2.3.3 controlInformationRoot : Uid: The root of the identifier of the event associated with setting the datatype to its specified value.

7.3.2.3.4 controlInformationExtension : String: The extension of the identifier of the event associated with setting the datatype to its specified value.

Together, the root and extension identify a particular record of a real world event that may supply additional information about the value such as who made the change, when it was made, why it was made, what system originated the change. These attributes exist because sometimes this information is required, but the value is being represented in an external context that does not contain a proper relationship to the control information for the value itself. The record need not be directly or easily resolvable. Conformance statements may make additional statements about these two properties, or about how such a reference should be resolved.

7.3.2.4 Equality

There is no equality definition for HXIT, since it is an abstract and private type. The attributes of HXIT (validTimeLow, validTimeHigh, controlInformationRoot, controlInformationExtension) never participate in the determination of equality of specializations of HXIT.

7.3.2.5 Invariants

— If a controlInformationExtension is provided, a ControlInformationRoot shall also be provided

OCL for invariants:

```
def: let noHistory : Boolean =
     validTimeLow.oclIsUndefined and
     validTimeHigh.oclIsUndefined and
     controlInformationRoot.oclIsUndefined and
     controlInformationExtension.oclIsUndefined

inv "extension requires root":
     controlInformationExtension.oclIsDefined implies
     controlInformationRoot.oclIsDefined
```

7.3.2.6 Example

```
<example
  xsi:type="ST" value="This is some content"
  validTimeLow="200506011000" validTimeHigh="200507031500"
    controlInformationRoot="1.2.3.4.5.6">
</example>
```

In this example, the value of the example attribute "This is some content" was valid from June 1, 2005, 10 a.m. to July 3, 2005 at 3 p.m. The value was set to "This is some content" by an event which is uniquely identified by the OID of 1.2.3.4.5.6. Some information system somewhere (and how to determine that should be in an applicable conformance profile) will be able to resolve this OID to a reference that may be used to determine the user who entered these data into the system.

7.3.3 ANY

7.3.3.1 Description

Specializes HXIT

Defines the basic properties of every data value. This is conceptually an abstract type, meaning that no proper value can be just a data value without belonging to any concrete type. Every public concrete type is a specialization of this general abstract DataValue type.

However exceptional values (nullFlavored values) may be of type ANY, except for the exceptional values that imply the NullFlavor INV, since this requires a type to be meaningful. Not all NullFlavors may be used with the type ANY (see 7.1.4 for more details)

7.3.3.2 ISO/IEC 11404 Syntax

```
type ANY = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring)
)
```

The appropriate use of all three of the attributes on ANY (in addition to the four properties inherited from HXIT) is intimately bound to the specification with which the datatypes are used, and generally that specification needs to establish special ways to control their use. Information Processing Entities claiming direct or indirect conformance shall make it clear how the use of these three attributes is controlled.

7.3.3.3 Attributes

7.3.3.3.1 nullFlavor : NullFlavor: If the value is not a proper value, indicates the reason.

Though the nullFlavor concept has some relationship with the UML/OCL null, it is not the same thing, and the relationship and differences between the two must be understood to properly implement this International Standard. For further discussion, see 7.1.4 (Null and NullFlavor).

NOTE nullFlavor includes the concept of a UML null value, and also includes potentially fully populated instances that do not conform to the requirements placed on the instance (also known as "exceptional instances").

Both nonNull and nullFlavored values shall always be valid according to the rules expressed in this International Standard.

If populated, the value of this attribute shall be taken from the HL7 NullFlavor code system. The current values are:

NullFlavor Enumeration. OID: 2.16.840.1.113883.5.1008			
1	NI	No information	The value is exceptional (missing, omitted, incomplete, improper). No information as to the reason for being an exceptional value is provided. This is the most general exceptional value. It is also the default exceptional value.
2	INV	Invalid	The value as represented in the instance is not a member of the set of permitted data values in the constrained value domain of a variable.
3	OTH	Other	The actual value is not a member of the set of permitted data values in the constrained value domain of a variable. (e.g. concept not provided by required code system).
4	PINF	Positive infinity	Positive infinity of numbers.
4	NINF	Negative infinity	Negative infinity of numbers.
3	UNC	Unencoded	No attempt has been made to encode the information correctly but the raw source information is represented (usually in originalText).
3	DER	Derived	An actual value may exist, but it must be derived from the information provided (usually an expression is provided directly).
2	UNK	Unknown	A proper value is applicable, but not known.
3	ASKU	Asked but unknown	Information was sought but not found (e.g. patient was asked but didn't know).
4	NAV	Temporarily unavailable	Information is not available at this time but it is expected that it will be available later.
3	NASK	Not asked	This information has not been sought (e.g. patient was not asked).
3	QS	Sufficient quantity	The specific quantity is not known, but is known to be non-zero and is not specified because it makes up the bulk of the material."Add 10 mg of ingredient X, 50 mg of ingredient Y, and sufficient quantity of water to 100 ml." The null flavor would be used to express the quantity of water.
3	TRC	Trace	The content is greater than zero, but too small to be quantified.
2	MSK	Masked	There is information on this item, available but it has not been provided by the sender due to security, privacy or other reasons. There may be an alternate mechanism for gaining access to this information. Warning: Using this null flavor does provide information that may be a breach of confidentiality, even though no detailed data are provided. Its primary purpose is for those circumstances where it is necessary to inform the receiver that the information does exist without providing any detail.
2	NA	Not applicable	No proper value is applicable in this context (e.g. last menstrual period for a male).

ISO/IEC 11404 Syntax for nullFlavor attribute

```
 type NullFlavor = enumerated (NI, INV, OTH, NINF, PINF, UNC, DER, UNK,
ASKU, NAV, QS, NASK, TRC, MSK, NA)
```

Some of the Null flavors are not generally applicable to all datatypes. The NullFlavors NINF, PINF, QS, and TRC shall only be used in association with QTY types. The NullFlavor UNC shall only be used with any type that has an originalText, and when UNC is used the originalText property shall be populated. When the NullFlavor DER is used, an expression shall be provided.

7.3.3.3.2 updateMode : UpdateMode: This property allows a sending system to identify the role that the attribute plays in processing the instance that is being represented.

If populated, the value of this attribute shall be taken from the HL7 UpdateMode code system. The current values are:

UpdateMode Enumeration. OID: 2.16.840.1.113883.5.57			
1	A	Add	The item was (or is to be) added, having not been present immediately before. (If it is already present, this may be treated as an error condition.)
1	D	Remove	The item was (or is to be) removed (sometimes referred to as deleted). If the item is part of a collection, delete any matching items.
1	R	Replace	The item existed previously and has been (or is to be) revised. (If an item does not already exist, this may be treated as an error condition.)
1	AR	Add or replace	The item was (or is to be) either added or replaced. No assertion is made as to whether the item previously existed.
1	N	No change	There was (or is to be) no change to the item. This is primarily used when this element has not changed, but other attributes in the instance have changed.
1	U	Unknown	It is not specified whether or what kind of change has occurred to the item, or whether the item is present as a reference or identifying property.
1	K	Key	This item is part of the identifying information for the object that contains it.

ISO/IEC 11404 Syntax for updateMode attribute

```
type UpdateMode = enumerated (A, D, R, AR, N, U, K)
```

If no updateMode is provided, there is no information as to how this information updates any existing information. The above-mentioned descriptions use the word "matching". For the purposes of the datatypes, this means the equality operations defined in this International Standard (in other contexts where this code system is used, "matching" may have other meanings).

NOTE UpdateMode does not affect the semantics or behaviour of the datatype itself, but can affect the behaviour of systems processing objects containing instances of the datatype.

7.3.3.3.3 flavorId : Set(String): Signals the imposition of one or more sets of constraints on the datatype. The sole purpose of specifying that a constraint that has been used to further constrain the datatype is to support validation of the instance: a validation engine can look up the rules expressed for the specified flavors and confirm that the instance conforms to the rules for the flavor. No other processing should depend on the content of the flavor attribute**.**

No other semantic or computational use shall depend on the value of this property. If this value is populated, the datatype flavor(s) shall be a valid constraint on the type of the value.

There is further discussion about the use of datatype flavors in A.3.

7.3.3.4 Equality

By default, equals is determined as specified below. Selected specializations of ANY override equals to specify how semantic equality is evaluated for the type. Each type clearly documents how equality is determined.

The following table summarizes the relationship between null, nullFlavor and equals:

other **this**	null	nullFlavor	proper value
Null	null	null	Null
nullFlavor	null	nullFlavor[a]	nullFlavor[a]
proper value	null	nullFlavor[a]	proper value[b]

[a] First common generalization of both NullFlavors.
[b] Unless specifically defined for a specialization, use the general equality algorithm.

The general equality algorithm says that two values are equal if they have the same type, and if all the attributes not defined on HXIT or ANY are also equal. If any of the attribute's equality is null or a nullFlavor then the result is null or has the most common NullFlavor.

NOTE 1 See the comments on comparing nullValues in 7.1.4 (Null and NullFlavors).

The equality operation is *reflexive*, *symmetric*, *transitive*, and *consistent*, and implementations shall conform to these requirements. The equality rules defined in this International Standard conform to these requirements.

This operation conforms to the general rules for operations and nullFlavors defined in 7.1.4 (Null and NullFlavor), but the rules are described here in depth, for greater clarity. In particular, the rules are transitive within the equals operation: if any of the attributes or collection items have a nullFlavor (other than the NullFlavor NA), the result will become a value of that nullFlavor, unless specifically defined otherwise.

Equals does not override the = operation defined in OCL, nor should it override the normal equivalent for the OCL = operation in any implementation platform. It defines semantic equality.

NOTE 2 UpdateMode and flavorId are always ignored when testing for equality.

7.3.3.5 Invariants

An instance may only be of type ANY (not a specialization) if it has a NullFlavor, and not if the NullFlavor implies INV

OCL for invariants:

```
  def: let isNull : Boolean = nullFlavor.oclIsDefined
  def: let isNotNull : Boolean = not isNull
  def: let noUpdate : Boolean = updateMode.oclIsUndefined
  def: let noUpdateOrHistory() : Boolean = noUpdate and
     noHistory
  def: let noUpdateOrHistory(member : ANY) : Boolean =
     member.oclIsDefined implies member.noUpdate and member.noHistory
  def: let invalid : Boolean = (nullFlavor = NullFlavor.INV or
       nullFlavor = NullFlavor.OTH or nullFlavor = NullFlavor.PINF or
       nullFlavor = NullFlavor.NINF or nullFlavor = NullFlavor.UNC or
       nullFlavor = NullFlavor.DER)

  inv "no ANY unless nullFlavored": (isNotNull or invalid) implies
         not oclIsTypeOf(ANY)
```

7.3.3.6 Operations

7.3.3.6.1 equals(other : ANY) : BL: Defines whether the this data value is considered semantically equal to another data value – that they carry the same meaning. See 7.3.3.4 for a discussion on how equality is determined.

7.3.3.6.2 isNull() : Boolean: Defines whether this type has a nullFlavor or not.

This operation is an exception from the normal rules for operations and nullFlavors: it will return true if the value has a nullFlavor, and false if it does not.

7.3.3.6.3 notNull() : Boolean: Defines whether this type does not have a nullFlavor or it does.

This operation is an exception from the normal rules for operations and nullFlavors: it will return false if the value has a nullFlavor, and true if it does not.

7.3.3.7 Examples

7.3.3.7.1 Simple true value

```
<example xsi:type="ANY" nullFlavor="UNK"/>
```

The value is unknown, and it is not even known what type the value might be (not fixed by context and not known in the instance).

7.3.3.7.2 Null and NullFlavor

This is a simple CD representing a coded concept (see 7.5.2)

```
<example code="784.0" codeSystem="2.16.840.1.113883.6.42">
  <displayName value="Headache"/>
</example>
```

This provides a code, a codeSystem, and a displayName. All the other attributes are null. In terms of meaning, the following instance is identical:

```
<example code="784.0" codeSystem="2.16.840.1.113883.6.42">
  <displayName value="Headache"/>
  <originalText nullFlavor="NI"/>
</example>
```

This has the same meaning because if an attribute is null, all we can say is that we have no information concerning it, nor do we know why we have no information about it, which is the same statement as NullFlavor.NI: we have no information about why we are not a proper type.

In the same sense, this also has the same meaning:

```
<example code="784.0" codeSystem="2.16.840.1.113883.6.42">
  <displayName value="Headache"/>
  <translation nullFlavor="NI"/>
</example>
```

In this case, there is a clear difference: a translation exists. However there is no information about the translation, nor why there is no information. So the outcome in terms of meaning is the same.

This example shows that it is valid to provide additional information along with a nullFlavor:

```
< example code="784.0" codeSystem="2.16.840.1.113883.6.42">
  <displayName value="Headache"/>
  <translation nullFlavor="NI" codeSystem="2.16.840.1.113883.6.96"/>
</example>
```

This is a slightly different statement: that there is no information about the translation of this code into SNOMED-CT. There are cases where this information is of significance, though they are not common.

7.3.3.7.3 UpdateMode

The principle use for updateMode is in tightly coupled messaging systems. A tightly coupled message system is one where a limited number of applications have tight trading partner agreements, and well understood and managed information flows. In these circumstances, it may be advantageous to agree that instead of sending all available information in each message, only information about what has changed in each transaction is sent in each message. Doing this has the advantage of greatly saving on implementation costs for both sender and receiver, while raising the prospect of information loss or scrambling if the message flows get out of sequence. UpdateMode is required to properly implement information about what has changed in a transaction.

In more general use, such as clinical documents and EHRs, the use of such transaction-based processing is inappropriate, therefore the updateMode attribute should generally be fixed to null in these contexts.

The following examples provide some examples for how updateMode might be used in a tightly coupled messaging system. These examples should not taken as proscriptive guidance for how updateMode and the associated transaction-based processing should be implemented.

The first set of examples concern a simple case where an object model has a single attribute for the birth date of a person, birthDate: TS.

A user opens the patient management dialog box, and changes the patient's birthdate to June 21, 1975. Since the system already has a birth date, the existing patient's birth date is being replaced. This leads to an instance being sent to the target tightly coupled application that includes the following fragment:

```
<birthDate value="19750621" updateMode="R"/>
```

The application checks and finds that it already has a value, so replaces it with the new value. If no value already existed, this would indicate an error condition, though how this should be handled depends on the details of the local agreements in force to make this kind of processing safe.

Later, another user opens the patient management dialog box and removes the patient's birth date. This leads to an instance being sent to the target tightly coupled application that includes the following fragment:

```
<birthDate nullFlavor="NI" updateMode="D"/>
```

A nullFlavor is required to make the value valid, and means to delete the birth date and replace it with the NullFlavor NI. This is effectively the same as:

```
<birthDate nullFlavor="NI" updateMode="R"/>
```

which means to update the birthdate to the NullFlavor NI. As seen in this instance, updateMode is not very useful on single instances like this. Where it becomes useful is to differentiate between information sent because it is changing (UpdateMode.R) from information sent to establish identity. To illustrate this, it is assumed that the trading party agreement controlling this transactionally based processing has specified that when updating a patient record, the patient object always includes name, gender and date of birth as identifying data. In this case, the instance always contains a birth date. It can be marked that the birthDate attribute is present as an identifying rather than a changing attribute using the UpdateMode "Key":

```
<birthDate nullFlavor="NI" updateMode="K"/>
```

Where updateMode starts to become really useful is with maintaining lists. The following set of examples concern the patients contact list.

Typical applications have different entries on the user UI for the different kind of contact details. The particular kinds of contact details that need to be supported tend to be rather a moving target in the information age, so this International Standard implements the actual contact details as a typed list (increasingly many applications are following suit). In this hypothetical case, there is input fields for home telephone number, mobile telephone number, fax number and e-mail address, and the attribute is contacts: DSET(TEL). A typical patient record might generate a contact list like this in a non-transactional environment:

```
<contacts>
  <item value="tel:+11015551234" use="H" capabilities="voice"/>
  <item value="tel:+11995556787" use="MC" capabilities="voice sms"/>
  <item value="tel:+11015551235" capabilities="fax"/>
  <item value="mailto:example@example.com"/>
</contacts>
```

In a tightly coupled messaging system using transactional processing based on updateMode, this full list would rarely be sent. Instead, only bits of it are sent.

If a user entered the patient management dialog box and changed the user's home telephone number, the following instance would be sent:

```
<contacts>
  <item value="tel:+11015551234" use="H"
      capabilities="voice" updateMode="D"/>
  <item value="tel:+12315559876" use="H"
      capabilities="voice" updateMode="A"/>
</contacts>
```

The receiving application compares its information with the instance. If it cannot find the existing number +11015551234, that is an error to be handled according to local agreements. If it can, it deletes it, and adds the new number. It might be tempting to send this instance instead:

```
<contacts>
  <item value="tel:+12315559876" use="H"
      capabilities="voice" updateMode="R"/>
</contacts>
```

with the instruction to replace the existing number. However, which number should be replaced? There is no answer, except that matching is based on equality; thus, this says to replace the home number +12315559876 with the home number +12315559876.

For this reason, it is best to use Add and Delete with datatype values. Replace and Add/Replace are not very useful since matching is done on equality – all that can be said is "replace value A with value A" – of limited use. These are generally more useful one complex objects, but have been included in the list of legal values for datatypes to ensure that all known transactional processing use cases are met.

There is one use case that UpdateModel.R can be used for. Since matching is done by equality, and equality is only based on the number itself, the following two instances have the same meaning:

```
<contacts>
  <item value="tel:+11015551234" use="H"
      capabilities="voice" upateMode="D"/>
  <item value="tel:+ 11015551234" use="H WP"
      capabilities="voice fax" updateMode="A"/>
</contacts>
```

```
<contacts>
  <item value="tel:+ 11015551234" use="H W"
      capabilities="voice fax" updateMode="R"/>
</contacts>
```

Both these instances mean to replace the existing details for +12315559876 with a new set of details for +12315559876, updating the use and capabilities.

7.3.4 BL

7.3.4.1 Description

Specializes ANY

BL stands for the values of two-valued logic. A BL value can be either true or false, or may have a nullFlavor.

7.3.4.2 ISO/IEC 11404 Syntax

```
type BL = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  value : boolean
)
```

With any data value potentially having a nullFlavor, the two-valued logic is effectively extended to a three-valued logic as shown in the following truth tables:

NOT	
True	false
false	true
null	null

AND	true	false	null
true	true	false	null
false	false	false	false
null	null	false	null

OR	true	false	null
true	true	true	True
false	true	false	null
null	true	null	null

In this table, null stands for either true null or a nullFlavor. If the null (or either of the nulls) is a true null, the result will be a true null. Where a boolean operation is performed upon two datatypes with different NullFlavors, the nullFlavor of the result shall be any common ancestor of the two different nullFlavors. The result should be the first common ancestor.

7.3.4.3 Attributes

7.3.4.3.1 value: Boolean: The value of the BL.

This is an example of the primitive type wrapping pattern. See 6.3 for additional details.

7.3.4.4 Equality

Two boolean values are equal if they are nonNull and have the same value.

7.3.4.5 Invariants

The BL shall have a value if it does not have a nullFlavor

OCL for invariants:

```
inv "value if not nullFlavored":
            isNotNull implies value.oclIsDefined
```

7.3.4.6 Operations

7.3.4.6.1 and(other : BL) : BL: True if both values are true. False if either value is false. Null or nullFlavored otherwise.

7.3.4.6.2 or(other : BL) : BL: True if either this or other are true. False if both values are false. Null or nullFlavored otherwise

7.3.4.6.3 xor(other : BL) : BL: True if either this and other are different and not null. False if both this and other have the same value. Null or nullFlavored otherwise.

7.3.4.6.4 implies(other : BL) : BL: True if either this is false, or if this and other are true. False if this is true and other is false. Null or nullFlavored otherwise.

7.3.4.6.5 not() : BL: False if this true, true if this is false. Null or nullFlavored otherwise.

NOTE These operations do not always confom to the general rules for nullFlavors and operations due to the special nature of these logical operations. For example, (null/nullFlavored) or True is true, because it does not matter whether the missing value might actually be true or false, and therefore the result will be the same. Also, from an implementer's point of view, these operations are strictly symmetric, and this can require some special case handling to prevent implementation errors when performing operations on null objects.

7.3.4.7 Examples

7.3.4.7.1 Simple True Value

```
<example xsi:type="BL" value="true"/>
```

7.3.4.7.2 Unknown Value

```
<example xsi:type="BL" nullFlavor="UNK"/>
```

7.3.5 BL.NONNULL

7.3.5.1 Description:

A flavor that constrains BL

BL.NONNULL is a constrained instance of BL that cannot have a nullFlavor. By implication, a null can never be used in the place of a BL.NONNULL, though this is not a rule that can be enforced by this International Standard.

7.3.5.2 Invariants

BL.NONNULL cannot have a nullFlavor

OCL for invariants:

```
inv "cannot have a nullFlavor": not isNull
```

7.4 Text and binary datatypes

7.4.1 Overview

Text and binary datatypes are datatypes that provide support for text and multimedia data (see Figure 3).

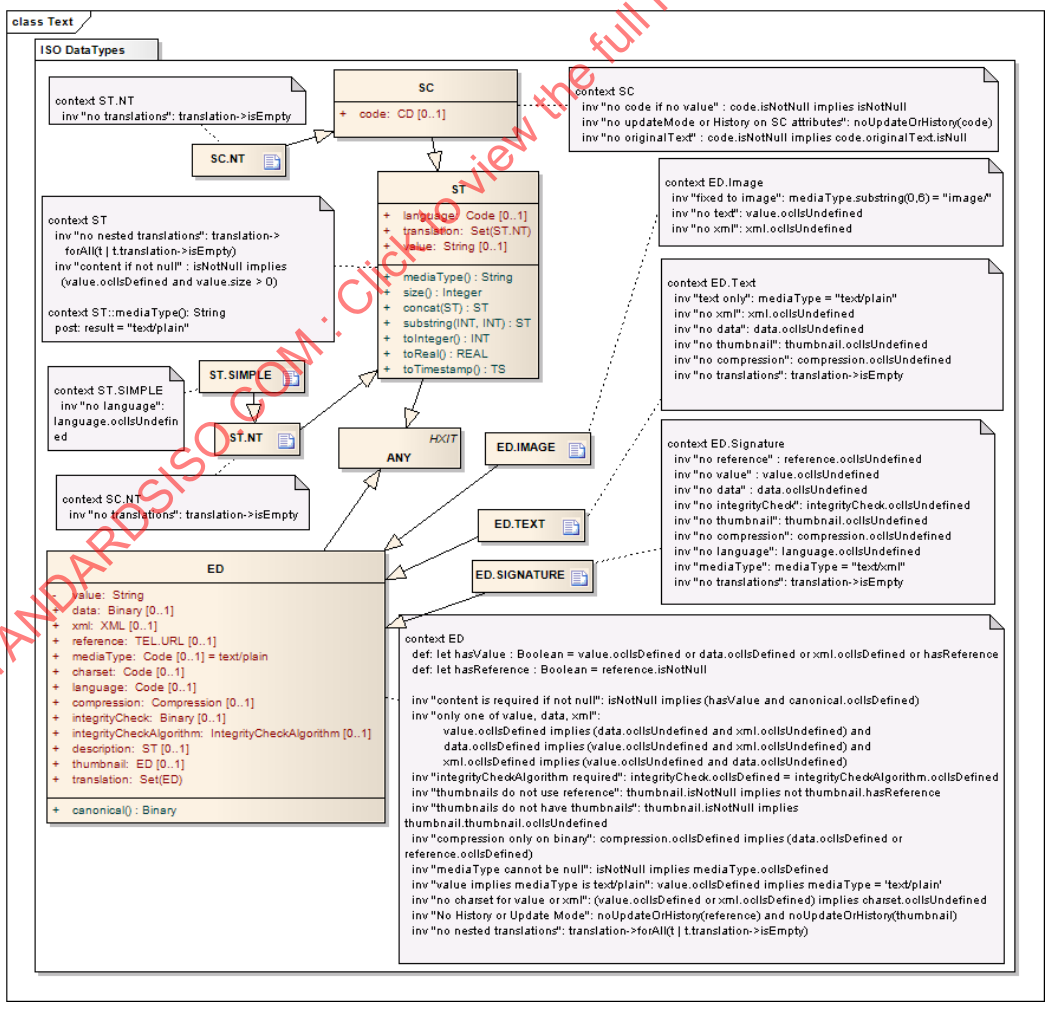

Figure 3 — Text and binary datatypes

7.4.2 ED (Encapsulated Data)

7.4.2.1 Description

Specializes ANY

Encapsulated data are data that are primarily intended for human interpretation or for further machine processing outside the scope of this International Standard. This includes unformatted or formatted written language, multimedia data, or structured information as defined by a different standard (e.g. XML-signatures).

NOTE Encapsulated data can be present in two forms, inline or by reference. The content is the same whether it is located inline or remote.Inline data are communicated or moved as part of the encapsulated data value, whereas by-reference data may reside at a different location: a URL/URI that provides reference to the information required to locate the data. Inline data may be provided in one of three different ways:

— as a plain sequence of characters (value):

— as a binary (a sequence of bytes) (data):

— as xml content (xml).

Content shall be provided if the ED has no nullFlavor. The content may be provided in-line (using only one of value, data or xml), or it may be provided as a reference.Content may be provided in-line and a reference also may be given; in these cases, it is expected that the content of the reference will be exactly the same as the in-line content. Information processing entities are not required to check this, but may regard it as an error condition if the content does not match.

7.4.2.2 ISO/IEC 11404 Syntax

```
type ED = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  value : characterstring,
  data : Sequence(Octet),
  xml : XML,
  reference : TEL.URL,
  mediaType : characterstring,
  charset : characterstring,
  language : characterstring,
  compression : Compression,
  integrityCheck : Sequence(Octet),
  integrityCheckAlgorithm : IntegrityCheckAlgorithm,
  description : ST,
  thumbnail : ED,
  translation : Set(ED)
)
```

7.4.2.3 Attributes

7.4.2.3.1 value : String: A simple sequence of characters that contains the content.

If value is used, the mediatype is fixed to text/plain and the charset shall be consistent with the String Character Set. See 6.7.5 for more details.

7.4.2.3.2 data : Binary: A simple sequence of byte values that contains the content.

7.4.2.3.3 xml : XML: The content represented in plain XML form.

A direct representation is provided for XML. This is because this International Standard includes an XML serialization of the data, and this xml attribute is handled specially in the serialization form. The xml data are not different in any semantic sense to the same data if represented in the value or data attributes.

NOTE These three representations of the ED data –as a sequence of characters, as a sequence of bytes, or as XML in a native XML format – are mutually incompatible and could also have been implemented as three specializations of an abstract ED supertype. However, doing so would complicate definition and implementation of the ED flavors and complicate the associated XML format (the additional of a mandatory xsi:type attribute) without significantly simplifying the overall implementation of ED.

7.4.2.3.4 reference : TEL.URL : A URL the target of which provides the binary content.

The semantic value of an encapsulated data value is the same, regardless of whether the content is present as inline content or just by reference. However, an encapsulated data value without inline content behaves differently, since any attempt to examine the content requires the data to be downloaded from the reference. An encapsulated data value may have both inline content and a reference.

If data are provded in the value, data or xml attributes, the reference shall point to the same data. It is an error if the data resolved through the reference does not match either the integrity check, data as provided, or data that had earlier been retrieved through the reference and then cached. The mediatype of the ED shall match the type returned by accessing the reference.

The reference may contain a usablePeriod to indicate that the data may only be available for a limited period of time. Whether the reference is limited by a usablePeriod or not, the content of the reference shall be fixed for all time. Any application using the reference shall always receive the same data, or an error. The reference cannot be re-used to send a different version of the same data, or different data.

7.4.2.3.5 mediaType : Code: Identifies the type of the encapsulated data and can be used to determine a method to interpret or render the content.

The IANA defined domain of media types is established by the IETF RFCs 2045 and 2046. mediaType has a default value of text/plain and cannot be null. If the media type is different to text/plain, the *mediaType* attribute shall be populated.

If the content is compressed using a specified compression algorithm, the mediaType shall refer the mediaType of the uncompressed data, whether the data are accessed by reference or not.

7.4.2.3.6 charset : Code: An Internet Assigned Numbers Authority (IANA) Charset Registered character set and character encoding for character-based encoding types**.**

Whenever the content of the ED is character type data in any form, the charset property needs to be known. If the content is provided directly in the value attribute, then the charset shall be a known character set consistent with the String Character Set (see 6.7.5 for more details). If the content is provided as a reference, and the access method does not provide the charset for the content (such as by a mime header), then the charset shall be conveyed as part of the ED.

7.4.2.3.7 language: Code: The human language of the content. Valid codes are taken from IETF RFC 3066. If this attribute is null, the language may be inferred from elsewhere, either from the context or from unicode language tags, for example.

Conformance profiles should define defaulting rules for language for a given usage environment of this International Standard.

NOTE While language attribute usually alters the interpretation of the text, the language attribute does not alter the meaning of the characters in the text.

7.4.2.3.8 compression : Compression: The compression algorithm, if any, used on the raw byte data.

If the attribute is null, the data are not compressed. Compression only applies to the binary form of the content.

If populated, the value of this attribute shall be taken from the HL7 CompressionAlgorithm code system. The current values are:

CompressionAlgorithm Enumeration. OID: 2.16.840.1.113883.5.1009			
1	DF	deflate	The deflate compressed data format as specified in IETF RFC 1951.
1	GZ	gzip	A compressed data format that is compatible with the widely used GZIP utility as specified in IETF RFC 1952 (uses the deflate algorithm).
1	ZL	zlib	A compressed data format that also uses the deflate algorithm. Specified as IETF RFC 1950.
1	Z	compress	Original UNIX compress algorithm and file format using the LZC algorithm (a variant of LZW). Patent encumbered and less efficient than deflate.
1	BZ	bzip	bzip-2 compression format. See [http://www.bzip.org/] for more information.
1	Z7	Z7	7z compression file format. See [`http://www.7-zip.org/7z.html`] for more information.

Some compression formats allow multiple archive files to be embedded within a single compressed volume. Applications shall ensure that the decompressed form of the data conforms to the stated media type.

ISO/IEC 11404 Syntax for compression attribute

```
type Compression = enumeration (DF, GZ, ZL, Z, BZ, Z7)
```

7.4.2.3.9 integrityCheck : Binary: A checksum calculated over the binary data.

The purpose of this property, when communicated with a reference is for anyone to validate later whether or not the reference still resolved to the same content that the reference resolved to when the encapsulated data value with reference was created. If the attribute is null, there is no integrityCheck.

It is an error if the data resolved through the reference does not match the integrity check.

The integrity check is calculated according to the integrityCheckAlgorithm. By default, the Secure Hash Algorithm-1 (SHA-1) shall be used. The integrity check is binary encoded according to the rules of the integrity check algorithm.

The integrity check is calculated over the raw binary data that is contained in the data component, or that is accessible through the reference. No transformations are made before the integrity check is calculated. If the data are compressed, the Integrity Check is calculated over the compressed data.

7.4.2.3.10 integrityCheckAlgorithm : IntegrityCheckAlgorithm: The algorithm used to compute the integrityCheck value.

If populated, the value of this attribute shall be taken from the HL7 IntegrityCheckAlgorithm code system. The current values are:

IntegrityCheckAlgorithm Enumeration. OID: 2.16.840.1.113883.5.1010			
1	SHA1	secure hash algorithm - 1	This algorithm is defined in FIPS PUB 180-1: Secure Hash Standard. As of April 17, 1995.
1	SHA256	secure hash algorithm - 256	This algorithm is defined in FIPS PUB 180-2: Secure Hash Standard.

ISO/IEC 11404 Syntax for integrityCheckAlgorithm attribute

```
type IntegrityCheckAlgorithm = enumeration (SHA1, SHA256)
```

7.4.2.3.11 description : ST: An alternative description of the media where the context is not suitable for rendering the media.

For example, short text description of an image or sound clip, etc.; this attribute is not intended to be a complete substitute for the original. For complete substitutes, use the "translation" property.

The intent of this property is to allow compliance with disability requirements, such as those expressed in the Americans with Disability Act (also known as "Section 508"), where there is a requirement to provide a short text description of included media in some form that can be read by a screen reader. This is similar to a very short thumbnail with mediaType = text/plain.

7.4.2.3.12 thumbnail : ED: An abbreviated rendition of the full content.

A thumbnail requires significantly fewer resources than the full content, while still maintaining some distinctive similarity with the full content. A thumbnail is typically used with by-reference encapsulated data. It allows a user to select the appropriate content more efficiently before actually downloading through the reference.

Originally, the term thumbnail refers to an image in a lower resolution (or smaller size) than another image. However, the thumbnail concept can be metaphorically used for media types other than images. For example a movie may be represented by a shorter clip; an audio-clip may be represented by another audio-clip that is shorter, has a lower sampling rate, or a glossy compression; or an abstract provided for a long document.

A thumbnail itself shall not contain a thumbnail.

7.4.2.3.13 translation : Set(ED): Alternate renditions of the same content translated into a different language or a different mediaType.

The translations shall convey the same information, but in a different language or mediaType. Translations shall not contain translations. The translations do not take part in the test for equality, consequently shall not introduce any new semantics to the value.

7.4.2.4 Equality

Two nonNull values of type ED are equal if – and only if – their mediaType and data are equal. For those ED values with compressed data or referenced data, only the dereferenced and uncompressed data count for the equality test (the canonical content; see 7.4.2.6.1). The compression, thumbnail, translation and reference property themselves are excluded from the equality test. In addition the language property is excluded from the test (refer to 7.4.2.7.2). If the mediaType is character-based and the charset property is not equal, the charset property must be resolved through mapping of the data between the different character sets.

An ED with a plain text content may also be equal to a ST with the same character content, following the rules described above.

7.4.2.5 Invariants

- Either reference, data, value or xml shall be provided if not nullFlavored, and at least one byte of data shall be referenced.
- Only one of data, value or xml may be specified.
- An integrityCheckAlgorithm shall be provided and may only be provided if an integrityCheck is provided.
- if a thumbnail is provided, it shall not use a reference.
- if a thumbnail is provided, it shall not have a thumbnail.
- Compression can only be specified if data are provided as a binary or as a reference.
- Mediatype cannot be null.
- If value is used, the mediaType is plain text.
- A character set shahll not be asserted for plain text or xml content (for plain text see 6.7.5 and implicitly derived for the XML content).
- Translations may not contain translations.

OCL for invariants:

```
def: let hasValue : Boolean = value.oclIsDefined or
       data.oclIsDefined or xml.oclIsDefined or hasReference
def: let hasReference : Boolean = reference.isNotNull

inv "content is required if not null": isNotNull implies
      (hasValue and canonical.oclIsDefined)
inv "only one of value, data, xml": value.oclIsDefined implies
        (data.oclIsUndefined and xml.oclIsUndefined) and
         data.oclIsDefined implies (value.oclIsUndefined and
               xml.oclIsUndefined) and
         xml.oclIsDefined implies (value.oclIsUndefined and
               data.oclIsUndefined)
inv "integrityCheckAlgorithm required": integrityCheck.oclIsDefined
          = integrityCheckAlgorithm.oclIsDefined
inv "thumbnails do not use reference": thumbnail.isNotNull implies
          not thumbnail.hasReference
inv "thumbnails do not have thumbnails": thumbnail.isNotNull
          implies thumbnail.thumbnail.oclIsUndefined
inv "compression only on binary": compression.oclIsDefined implies
           (data.oclIsDefined or reference.oclIsDefined)
inv "mediaType cannot be null": isNotNull implies
          mediaType.oclIsDefined
inv "value implies mediaType is text/plain": value.oclIsDefined
          implies mediaType = 'text/plain'
inv "no charset for value or xml": (value.oclIsDefined or
          xml.oclIsDefined) implies charset.oclIsUndefined
inv "No History or Update Mode": noUpdateOrHistory(reference) and
          noUpdateOrHistory(thumbnail)
   inv "no nested translations": translation->forAll(t |
         t.translation->isEmpty)
```

7.4.2.6 Operations

7.4.2.6.1 canonical : Binary: The sequence of bytes that is the actual data.

NOTE 1 This sequence of bytes is retrieved from the reference if one was provided, and decompressed if appropriate.

NOTE 2 This operation does not follow the normal rules for operations and nullFlavors because the return type cannot have a nullFlavor.

NOTE 3 The result is null if the ED has a nullFlavor.

7.4.2.7 Examples

7.4.2.7.1 Plain text

```
<example xsi:type="ED" value="this is plain text"
    mediaType="text/plain"/>
<example xsi:type="ED" value="this is plain text"/>
```

This ED is a simple representation of plain text. The mediaType is specified as text/plain. Since the default value for mediatype is text/plain, the mediaType does not need to be represented in the XML, and the second example is also valid. The character set is not specified and in this case, the character set shall not be present; the character set matches the encoding of the XML.

7.4.2.7.2 Language

```
<example xsi:type="ED" flavorId="ED.TEXT" value="this is plain text"
language="en" mediaType="text/plain"/>
<example xsi:type="ED" flavorId="ED.TEXT" value="dieses ist normaler Text"
language="de" mediaType="text/plain"/>
```

This explicitly notes that the language is English in the first case and German in the second case. If there is no language attribute, then the language is unknown, though it is usually safe to assume that the locally predominant language is appropriate. The type is assigned the flavor ED.TEXT – a text only flavor of ED.

```
<example xsi:type="ED" value="this is plain text" language="en">
  <translation xsi:type="ED" value="dieses ist normaler Text"
    language="de"/>
</example>
```

Here, the German version is contained as a translation of the English version.

```
<example xsi:type="ED" value="this is plain text" language="en-ca">
  <translation xsi:type="ED" value="ce ci est du texte non structuré"
    language="fr-ca"/>
</example>
```

Localized languages may also be used, as in this French Canadian example.

7.4.2.7.3 Binary content

```
<example xsi:type="ED">
  <data>dGhpcyBpcyBiaW5hcnkgY29udGVudA==</data>
</example>
```

This ED contains the plain text "this is binary content". Since the mediaType is text/plain, the mediaType attribute does not need to be populated as this is its default value.

NOTE The character set of this plain text is unknown; it does not necessarily match that of the XML, and it is not safe to assume that it does.

```
<example xsi:type="ED" charset="UTF-8">
  <data>dGhpcyBpcyBiaW5hcnkgY29udGVudA==</data>
</example>
```

For this reason it would generally be appropriate to define a charset when using the data element, as shown in this example.

7.4.2.7.4 Reference

```
<example xsi:type="ED" mediaType="image/jpg”>
  <reference value="http://www.tempuri.org/XXXXXXXXXXX">
</example>
```

The contents of this ED are found at the URL “http://www.tempuri.org/XXXXXXXXXXX”. When accessed, this reference returns a binary stream of bytes, the mime type of the HTTP response is "image/jpg".

NOTE The http protocol supports compression directly. The compression attribute of the ED does not refer to any compression applied as part of the HTTP response, but to the data once the HTTP response has been completed and interpreted.

```
<example xsi:type="ED" mediaType="image/jpg”
     compression="GZ">
  <reference value="http://www.tempuri.org/XXXXXXXXXXX">
</example>
```

This example specifies that the result of the HTTP response is a gzipped version of the image bytes. The HTTP response itself could also specify that the HTTP response stream was gzipped – this would represent a second (redundant) compression of the data (though even the first gzip compression would be redundant given that the base type – JPEG – is a highly efficient representation anyway).

7.4.2.7.5 XML content

```
<example xsi:type="ED" mediaType="text/xml">
  <data>PHBhcmVudD4NCiAgPGNoaWxkPlRoaXMgaXMgc29tZSB0ZXh0IGluI
    HRoZSBjaGlsZDwvY2hpbGQ+
  DQogIFRoaXMgaXMgc29tZSB0ZXh0IGluIHRoZSBwYXJlbnQNCjwvcG
   FyZW50Pg==</data>
</example>
```

This ED contains some XML content provided in binary form. Like the previous example, the character set of the XML content is unknown; it does not necessarily match that of the ED XML, and it is not safe to assume that it does.

```
<example xsi:type="ED" mediaType="text/xml" charset="ASCII">
  <data>PHBhcmVudD4NCiAgPGNoaWxkPlRoaXMgaXMgc29tZSB0ZXh0IGlu
    IHRoZSBjaGlsZDwvY2hpbGQ+
  DQogIFRoaXMgaXMgc29tZSB0ZXh0IGluIHRoZSBwYXJ
    lbnQNCjwvcGFyZW50Pg==</data>
</example>
```

The charset is explicitly defined in this case. It does not need to match the XML document encoding.

```
<example xsi:type="ED" mediaType="text/xml">
  <xml>
    <parent>
      <child>This is some text in the child</child>
      This is some text in the parent
    </parent>
  </xml>
</example>
```

In this example, the xml is provided in-line using the xml element. The mediaType of the xml content shall be provided (it cannot be text/plain). The charset cannot be defined since it shall match that of the xml encoding.

7.4.2.7.6 Other content types

```
<example mediaType="application/pdf" compression="GZ">
   <data>/9j/4AAQSkZJRgABAgEAgACAAAD/2wCEAAICAgIC
      AQICAgICAgICAwQDAwMDAwUEBAMEBgYHBgYG
    BgYHCAoIBwcJBwYGCAsJCQoKCwsLBwgMDQwKDAoLCwoBAgIC
      AwMDBQMDBQoHBgcKCgoKCgoKCgoK
    CgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgo
      KCv/AABEIAg4CuQMBIQACEQED
    EQH/xAGiAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgsBAAM
      BAQEBAQEBAQEAAAAAAAABAgME
    BQYHCAkKCxAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1F
      hByJxFDKBkaEII0KxwRVS0fAk
    ...
    CigAooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKAC
      gAooAKKACigAooAKKACigAoo
    AKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAK
      KACigAooAKKACigAooAKKACi
    gAooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKACigA
      ooAKKACigAooAKKAP//Z</data>
</example>
```

This example shows an Adobe Acrobat document that has been compressed using the GZip compression algorithm.

```
<example mediaType="image/png">
   <reference value="http://example.org/xrays/128s8d9ej229se32s.png">
      <useablePeriod xsi:type="IVL_TS">
        <low value="20000720084 5"/>
        <high value="20000820084 5"/>
      </useablePeriod>
   </reference>
  <integrityCheck>EQH/xAGiAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgsB
     AAMBAQEBAQEBAQEAAAAAAAABAgME</integrityCheck>
   <thumbnail mediaType="image/jpeg">
      <data>/9j/4AAQSkZJRgABAgEAgACAAAD/2wCEAAICAgIC
      AQICAgICAgICAwQDAwMDAwUEBAMEBgYHBgYG
    BgYHCAoIBwcJBwYGCAsJCQoKCwsLBwgMDQwKDAoLCwoBAgIC
      AwMDBQMDBQoHBgcKCgoKCgoKCgoK
    CgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgo
      KCv/AABEIAg4CuQMBIQACEQED
    EQH/xAGiAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgsBAAM
      BAQEBAQEBAQEAAAAAAAABAgME
    BQYHCAkKCxAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1F
      hByJxFDKBkaEII0KxwRVS0fAk
    ...
    CigAooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKAC
      gAooAKKACigAooAKKACigAoo
    AKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAK
      KACigAooAKKACigAooAKKACi
    gAooAKKACigAooAKKACigAooAKKACigAooAKKACigAooAKKACigA
      ooAKKACigAooAKKAP//Z </data>
   </thumbnail>
</example>
```

This example contains a reference to an image, stored at a particular URL and available for the next month. An integrity check is provided for the image, as well as in inline thumbnail.

7.4.3 ED.IMAGE

7.4.3.1 Description

A flavor that constrains ED

ED.IMAGE constrains ED so that the contents must be an image.

7.4.3.1.1 Invariants

- The mediaType shall start with “image/”.
- The content cannot be provided as a text or xml – it shall be binary and/or reference.

OCL for invariants:

```
inv "fixed to image": mediaType.substring(0,6) = "image/"
inv "no text": value.oclIsUndefined
inv "no xml": xml.oclIsUndefined
```

7.4.4 ED.TEXT

7.4.4.1 Description

A flavor that constrains ED

ED.TEXT constrains ED so that it may only contain plain text.

This is useful because there is sometimes a need to allow for references, but the content shall be a simple string. In addition, no translations are allowed.

7.4.4.2 Invariants

- The mediaType shall be text/plain.
- The content cannot be provided as a text or data – it shall be text and/or reference.
- Thumbnail, compression and translations are not allowed.

OCL for invariants:

```
inv "text only": mediaType = "text/plain"
inv "no xml": xml.oclIsUndefined
inv "no data": data.oclIsUndefined
inv "no thumbnail": thumbnail.oclIsUndefined
inv "no compression": compression.oclIsUndefined
inv "no translations": translation->isEmpty
```

7.4.5 ED.SIGNATURE

7.4.5.1 Description

A flavor that constrains ED

ED.SIGNATURE constrains ED such that the contents shall be an XML digital signature as defined by the W3C XML Signature Syntax and Processing Recommendation.

If this flavor is implemented in a context where indirect conformance applies, the implementation may differ from the W3C Recommendation, and the conformance statement should declare the mapping between the implementation and the W3C Recommendation.

7.4.5.2 Invariants

— No value, data, reference, integrity check, thumbnail, compression, language or translations are allowed.

— The media type shall be text/xml.

OCL for invariants:

```
inv "no reference" : reference.oclIsUndefined
inv "no value" : value.oclIsUndefined
inv "no data" : data.oclIsUndefined
inv "no integrityCheck": integrityCheck.oclIsUndefined
inv "no thumbnail": thumbnail.oclIsUndefined
inv "no compression": compression.oclIsUndefined
inv "no language": language.oclIsUndefined
inv "mediaType": mediaType = "text/xml"
inv "no translations": translation->isEmpty
```

7.4.6 ST (Character string)

7.4.6.1 Description

Specializes ANY

The character string datatype stands for text data, primarily intended for machine processing (e.g. sorting, querying, indexing, etc.) or direct display. Used for names, symbols, presentation and formal expressions.

A ST shall have at least one character or else have a nullFlavor.

7.4.6.2 ISO/IEC 11404 syntax

```
type ST = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  value : characterstring,
  language : characterstring
  translation : Set(ST.NT)
)
```

7.4.6.3 Attributes

7.4.6.3.1 value : String: The actual content of the string. See 6.7.5 for discussion on string character encodings.

This is an example of the primitive type wrapping pattern. See 6.3 for more details.

7.4.6.3.2 language: Code: The human language of the content. Valid codes are taken from the IETF RFC 3066. If this attribute is null, the language may be inferred from elsewhere, either from the context or from unicode language tags, for example.

Conformance profiles should define defaulting rules for language for a given usage environment of this International Standard.

While language tags usually alter the meaning of the text, the language does not alter the meaning of the characters in the text.

7.4.6.3.3 translation : Set(ST.NT): Alternate renditions of the same content translated into a different language. Translations may not contain translations.

The translations shall convey the same information, but in a different language. The translations do not take part in the test for equality, so shall not introduce any new semantics to the value.

7.4.6.4 Equality

Two nonNull values of type ST are equal if – and only if – the sequence of characters they represent is equal (i.e. if they are not nullFlavored and their value attributes are equal). The translation property is excluded from the equality test. In addition the language property is excluded from the test, due to the problems this would introduce for values of type ST where the language is not specified.

An ED with a plain text content may also be equal to a ST with the same character content, following the rules described for ED (Section 0).

7.4.6.5 Invariants

— if there is a value, there shall be at least one character;

— translations may not contain translations.

OCL for invariants:

```
inv "no nested translations": translation->
      forAll(t | t.translation->isEmpty)

inv "content if not nullFlavored" : isNotNull implies
    (value.oclIsDefined and value.size > 0)
```

7.4.6.6 Operations

7.4.6.6.1 mediaType : String: Returns a value of text/plain.

7.4.6.6.2 size : Integer: The number of characters in the string.

7.4.6.6.3 concat(other : ST) : ST: The concatenation of this and other.

7.4.6.6.4 substring(lower : INT, upper : INT) : ST: The substring of this starting at character number lower, up to and including character number upper.

Character numbers run from 1 to this.size().

NOTE When characters are extracted from a string, it might be necessary to copy other predecessor characters that set the appropriate context in some character encodings.

7.4.6.6.5 toInteger : INT: If the content of the string is a valid integer, the value as an INT. If the content is not a valid integer, then NullFlavor NI.

A string is a valid integer if it conforms to the integer-literal format defined in ISO/IEC 11404, or if it conforms to the lexical representation of the integer type defined in xml schema.

7.4.6.6.6 toReal : REAL: If the content of the string is a valid floating point number, the value as a REAL. If the content is not a valid floating point number, then NullFlavor NI.

A string is a valid floating point number if it conforms to the real-literal format defined in ISO/IEC 11404, or if it conforms to the lexical representation of the double type defined in xml schema.

7.4.6.6.7 toTimestamp : TS: If the content of the string is a valid timestamp, the value as a TS. If the content is not a valid timestamp, then NullFlavor NI. A string is a valid integer if it conforms to the format described under the TS type.

7.4.6.7 Examples

```
<example language="en" value="cellulitis of the left foot"/>
```

7.4.7 ST.NT

7.4.7.1 Description

A flavor that constrains ST

ST.NT constrains ST such that no translations are allowed.

7.4.7.2 Invariants

— No translations are allowed.

OCL for invariants:

```
inv "no translations": translation->isEmpty
```

7.4.8 ST.SIMPLE

7.4.8.1 Description

A flavor that constrains ST.NT

ST.SIMPLE constrains ST.NT such that it has no language.

7.4.8.2 Invariants

— No language is allowed.

OCL for invariants:

```
inv "no language": language.oclIsUndefined
```

7.4.9 SC (coded string)

7.4.9.1 Description

Specializes ST

A character string that optionally may have a code attached. The text shall always be present if a code is present.

NOTE The code is often a local code. SC is used in cases where coding is exceptional (e.g. user error messages are essentially text messages, and the text message is the important content. However sometimes messages come from a catalog of prepared messages, which SC allows to reference).

Any non-null SC value may have a code, however, a code shall not be given without the text.

The similarities and differences between SC and CD are discussed in 7.5.2.2, CD and SC.

7.4.9.2 ISO/IEC 11404 syntax

```
type SC = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  value : characterstring,
  language : characterstring,
  translation : Set(ST.NT),
  code : CD
)
```

7.4.9.3 Attributes

7.4.9.4 code : CD: The coded value associated with the string. If the value is null or nullFlavored, there is no coded value associated with this string.

7.4.9.5 Equality

The definition of equality for SC is the same as for ST. Code is excluded from the equality test, and values of type SC may be equal to values of type ST (and therefore to values of type ED following the rules documented in for ST in 7.4.6.4).

7.4.9.6 Invariants

— If there is a code, there shall also be some content on the SC (and therefore the SC shall not have a nullFlavor).

— The originalText value of the CD shall be null (the originalText is the SC.value).

OCL for invariants:

```
inv "no code if no value" : code.isNotNull implies
   isNotNull
inv "no updateMode or History on SC attributes":
 noUpdateOrHistory(code)
inv "no originalText" : code.isNotNull implies
   code.originalText.isNull
```

7.4.9.7 Examples

```
<example xsi:type="SC" value="Intestinal nematode infection">
  <code code="57540006" codeSystem="2.16.840.1.113883.6.96"
      codeSystemName="Snomed-CT">
    <displayName value="Intestinal nematode infection (disorder)"/>
  </code>
</example>
<example xsi:type="SC" value="Lung nematode infection"/>
```

Two examples of SC are in use. When SC is mandatory, text is required, and coding is optional. This is often suitable for front-line data collection, particularly in emergency medicine or relief efforts, where there is no opportunity to perform thorough evaluation of the choice of concept. If a concept is known, it is used, and the designator is used as the text. If no concept can immediately be located, the user enters some text which may be post-coded later.

7.4.10 SC.NT

7.4.10.1 Description

A flavor that constrains SC

SC.NT constrains SC such that no translations are allowed.

7.4.10.2 Invariants

— No translations are allowed.

OCL for invariants:

```
inv "no translations": translation->isEmpty
```

7.5 Coded datatypes (terminology)

7.5.1 Overview

These datatypes provide support for use of codes and terms from terminologies and classifications (see Figure 4).

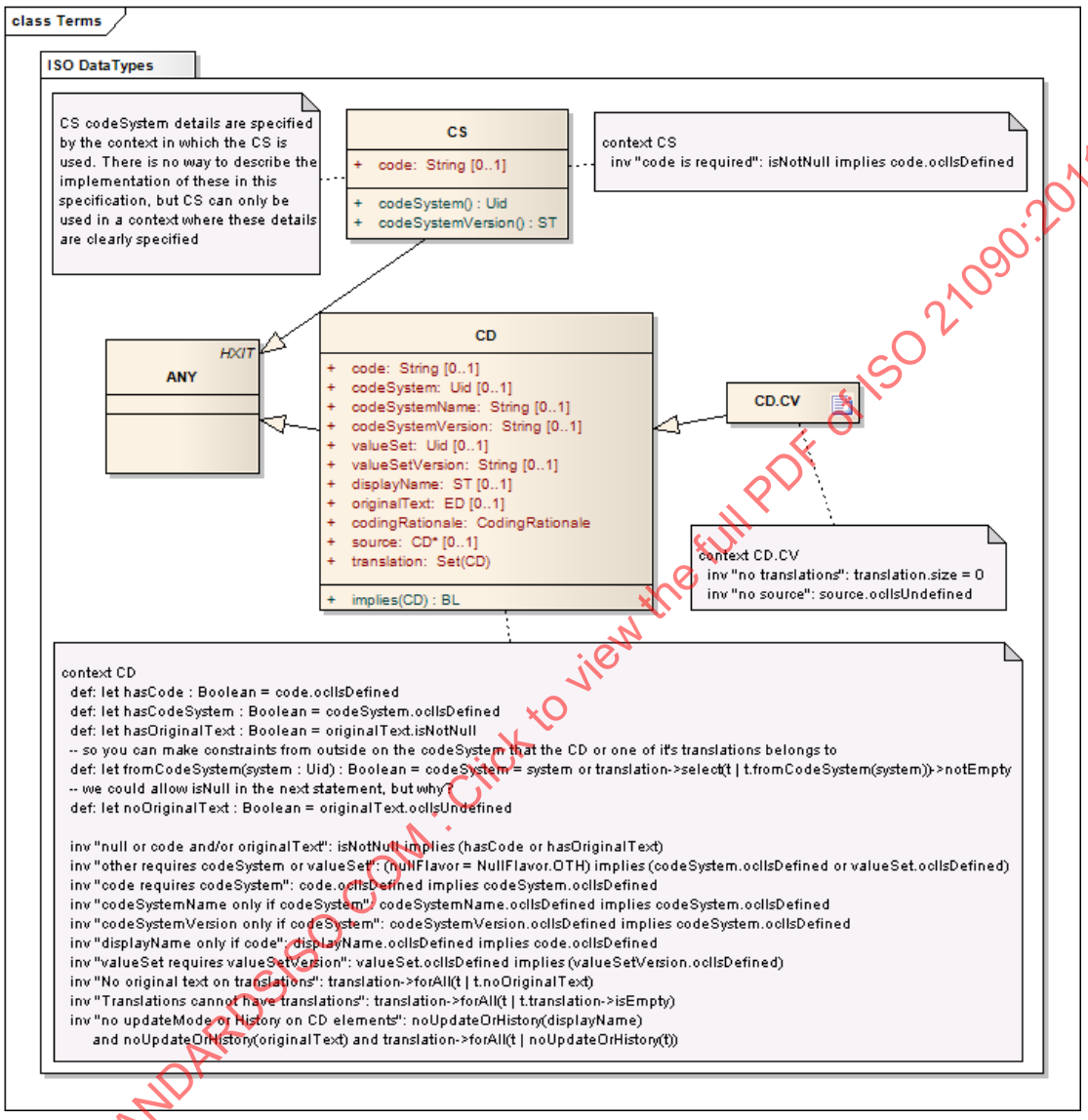

Figure 4 — Coded datatypes

7.5.2 CD (concept descriptor)

7.5.2.1 Description

Specializes ANY

A CD is a reference to a concept defined in an external code system, terminology or ontology. A CD may contain a simple code – that is, a reference to a concept defined directly by the referenced code system, or it may contain an expression in some syntax defined by the referenced code system that can be meaningfully evaluated, e.g. the concept of a "left foot" as a postcoordinated term built from the primary code "FOOT" and the qualifier "LEFT".

A CD may also contain an original text or phrase that served as the basis of the coding. This is preserved to allow for validation of the representation of the concept in various fashions.

A CD can contain one or more translations into multiple coding systems. The translations are all representations of the same concept in various code systems. There is only one concept, and only the first CD may contain an original text. It is possible to represent the translation chain – which CD was translated from which – if desired. Each CD may also carry a rationale to indicate why it is represented.

A CD with no nullFlavor attribute shall have a code attribute or nonNull originalText attribute. A CD that has a code, codeSystem or originalText attribute but does not meet external constraints of the applicable value set shall have a nullFlavor attribute with a value of "OTH".

Attributes with type CD are generally bound by externally specified constraints which constrain the coded concepts to which a CD may refer. These constraints may be qualified as "extensible" (CWE) or "not extensible“ (CNE). If the constraint is not extensible (CNE), then a CD that does not have a nullFlavor shall contain a code that conforms to the constraint. If the constraint is extensible (CWE) then a CD that does not have a nullFlavor shall contain either a code that exists in the domain with which the attribute is associated, a code from a locally defined code system, or just some originalText that describes the concept. If the code is taken from a locally defined code system, then the codeSystem property shall specify the local code system.

For both CNE and CWE constraint types, the translations may contain nonNull codes from any source, unless otherwise specified by the constraining model.

For code systems that define expression syntaxes, CNE constraints may be used, provided the code system definitions define the appropriate support to enable value sets to make useful statements about how to control the expression syntax, and that the value set machinery used also has the appropriate support.

7.5.2.2 CD and SC

The CD and SC types have very similar structures. CD has a code:codeSystem pair with translations, and an originalText which has type ED.Text – plain text that may be a reference. SC has a string and a code: CD to allow the string to be coded. In SC, the code does not have an originalText – it is fixed to the value attribute of the SC. Therefore, both types have a code:codeSystem pair with translations and originalText.

Although the types thus share the same capability of representing coded text, and have nearly the same core information structure, they differ in purpose. CD exists to provide a reference to a concept in defined in some code system, and possibly to reference some text in support. When a CD is mandatory in the context of use, a code must be provided, and an original text is optional, except in the CWE case discussed above, where either a code or an originalText must be provided. SC exists to provide text content that may additionally be encoded. When SC is mandatory, the text (which becomes the original text) shall be provided directly, and the code is always optional.

When it is obvious which aspect of the coded text is mandatory – code or text, then it will be obvious whether to use CD or SC. On the other hand, when neither aspect is mandatory, or both are mandatory, then it is not so obvious which to use.

NOTE When both are mandatory, either class can be constrained to require both aspects using some formal constraint language such as OCL).

Generally, when post-coding existing text, CD is an obvious choice, as it allows references into the existing text. In other cases, it may be ambiguous which of SC and CD is the correct choice. When choosing between them, users should consider the specialization hierarchy, which may dictate a particular choice (it may be useful that SC is a specialization of ST, or it may not), and also should consider the constraints that have been made on the type in the context of use; this may also dictate a particular choice.

In some circumstances, it is still ambiguous which type to use. In cases of doubt, implementors should choose CD by default.

7.5.2.3 ISO/IEC 11404 Syntax

```
type CD = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  code : characterstring,
  codeSystem : characterstring,
  codeSystemName : characterstring,
  codeSystemVersion : characterstring,
  valueSet : characterstring,
  valueSetVersion : characterstring,
  displayName : ST,
  originalText : ED,
  codingRationale : CodingRationale,
  translation : Set(CD),
  source : CD
)
```

7.5.2.4 Attributes

7.5.2.4.1 code : String: The plain code symbol defined by the code system, or an expression in a syntax defined by the code system which describes the concept.

If a code is provided, it shall be an exact match to a plain code symbol or expression defined by the code system. If the code system defines a code or expression that includes whitespace, the code shall include the whitespace. An expression can only be used where the codeSystem either defines an expression syntax, or there is a generally accepted syntax for the codeSystem. A code system may be defined that only defines an expression syntax with bindings to other code systems for the elements of the expression.

It is at the discretion of the interpreting system whether to check for an expression instead of a simple code and evaluate the expression instead of treating the expression as a code. In some cases, it may be unclear or ambiguous whether the code represents a single symbol or an expression. This usually arises where the code system defines an expression language and then defines pre-coordinated concepts with symbols which match their expression, e.g. UCUM. In other cases, it is safe to treat the expression as a symbol. There is no guarantee that this is always safe: the definitions of the codeSystem should always be consulted in order to determine how to handle potential expressions.

7.5.2.4.2 codeSystem : Uid: The code system that defines the code, or if no code was found, the codeSystem in which no code was found.

Code systems shall be referred to by a UID, which allows unambiguous reference to standard code systems and other local codesystems. Where either ISO or HL7 have assigned UID to code Systems, then these UIDs shall be used. Otherwise, implementations shall use an appropriate ISO Object Identifier (OID) or UUID to construct a globally unique local coding system identifier.

A CD that has a code attribute shall have a codeSystem specifying the system of concepts that defines the code.

A CD with a NullFlavor OTH indicates that a concept could not be coded in the coding system or value set specified. Thus, for these coding exceptions, the code system or value set that did not contain the appropriate concept shall be provided in codeSystem or valueSet.

7.5.2.4.3 codeSystemName : String: The common name of the coding system.

The code system name has no computational value. *codeSystemName* can never modify the meaning of codeSystem and cannot exist without codeSystem.

Information processing entities claiming direct or indirect conformance shall not functionally rely on *codeSystemName*. In addition, they may choose not to implement *codeSystemName* but shall not reject instances because *codeSystemName* is present.

NOTE The purpose of a code system name is to assist an unaided human interpreter of a code value to interpret codeSystem.

7.5.2.4.4 codeSystemVersion : String: If applicable, a version descriptor defined specifically for the given code system.

Different versions of one code system shall be compatible. By definition, a code symbol shall have the same meaning throughout all versions of a code system. Between versions, codes may be retired but not withdrawn or re-used. Where the definition of the meaning of a code symbol changes, it must still be compatible (equal) between different code system versions.

Whenever a code system changes in an incompatible way, it constitutes a new code system, not simply a different version, regardless of how the vocabulary publisher calls it. For example, the publisher of ICD-9 and ICD-10 calls these code systems, "revision 9" and "revision 10" respectively. However, ICD-10 is a complete redesign of the ICD code, not a backward compatible version. Therefore, for the purposes of this datatype International Standard, ICD-9 and ICD-10 are different code systems, not just different versions. By contrast, when LOINC updates from revision "1.0j" to "1.0k", this would be considered as just another version of LOINC, since LOINC revisions are backwards compatible.

7.5.2.4.5 valueSet : Uid: The value set that applied when this CD was coded.

Value sets shall be referred to by an identifier name which allows unambiguous reference to a value set. Where either ISO or HL7 have assigned an identifying name to a value set, then that name shall be used.

In many cases, a CD is created from a value set – either a code/code system pair is chosen from a valueSet, or one is not chosen and the CD has the exceptional value of NullFlavor.OTH. If no code is chosen, it is generally inappropriate to reference the code system from which the code was chosen as the value set may not match the code system (may include a subset of the codeSystem, or additional terms from other code systems); instead, the value set should be provided. In addition, there are some known use cases where the value set that a user or system was offered when choosing a code affects the interpretation of the code.

If a code is provided, the meaning of the code must come from the definition of the code in the code system. The meaning of the code shall not depend on the value set. Information Processing Entities claiming direct or indirect conformance shall not be required to interpret the code in the light of the valueSet, and they shall not reject an instance because of the presence or absence of any or a particular value set.

7.5.2.4.6 valueSetVersion : String: The version of the valueSet in which no code was found.

valueSetVersion shall be provided when a valueSet is provided, and otherwise shall be null. The value of the *valueSetVersion* shall properly identify a particular version of the value set following the rules defined by the value set or its publisher.

It is generally recommended that value set publishers specify that the version is identified by the date/time that the value set version is published, and that the publication process makes the date/time explicitly clear.

7.5.2.4.7 displayName : ST: A name, title, or representation for the code or expression as it exists in the code system.

If populated, the *displayName* shall be a valid human readable representation of the concept as defined by the code system at the time of data entry. The *displayName* shall conform to any rules defined by the codingSystem; if the codeSystem does not define a human representation for the code or expression, then none can be provided. *displayName* is included both as a courtesy to an unaided human interpreter of a code value and as a documentation of the name used to display the concept to the user. The display name has no functional meaning; it shall never exist without a code; and it shall never modify the meaning of the code. A display name may not be present if the code is an expression for which no display name has been assigned or can be derived. Information processing entities claiming direct or indirect conformance may choose not to implement *displayName* but shall not reject instances because *displayName* is present.

Display names shall not alter the meaning of the code value. Therefore, display names should not be presented to the user on a receiving application system without ascertaining that the display name adequately represents the concept referred to by the code value. Communication shall not simply rely on the display name. The display name's main purpose is to support implementation debugging.

7.5.2.4.8 originalText : ED: The text as seen and/or selected by the user who entered the data which represents the intended meaning of the user.

NOTE Local implementations might influence what is required to represent that original text.

Original text can be used in a structured user interface to capture what the user saw as a representation of the code on the data input screen, or in a situation where the user dictates or directly enters text, it is the text entered or uttered by the user.

It is valid to use the CD datatype to store only the text that the user entered or uttered. In this situation, original text will exist without a code. In a situation where the code is assigned sometime after the text was entered, originalText is the text or phrase used as the basis for assigning the code.

The details of the link in the originalText.reference between different artifacts of medical information (e.g. document and coded result) is outside the scope of this International Standard and may be further proscribed in other specifications that use this one.

The original text shall be an excerpt of the relevant information in the original sources, rather than a pointer or exact reproduction. Thus the original text shall be represented in plain text form. In specific circumstances, when the context of use is clearly described, the originalText may be a reference to some other text artefact for which the resolution scope is clearly described.

Values of type CD may have an original text despite not having a code. Any CD value with no code signifies a coding exception. In this case, originalText is a name or description of the concept that was not coded. Such CD values may also contain translations.

Translations directly encode the concept described in originalText. The originalText represents the originalText of the concept itself. Translations shall not have an originalText of their own.

7.5.2.4.9 translation : Set(CD): A set of other CDs that each represent a translation of this CD into equivalent codes within the same code system or into corresponding concepts from other code systems.

The translations are quasi-synonyms of one real-world concept. Every translation in the set is supposed to express the same meaning "in other words." However, exact synonymy rarely exists between two structurally different coding systems. For this reason, not all of the translations will be equally exact.

Translations shall not contain translations. The root CD has one set of translations which lists all the translations. The root translation is generally the one that best meets the conformance criteria for the CD. No implication about lineage of the translations can be drawn from the selection of the root code. Instead, the properties codingRationale and source is used to trace lineage.

In the absence of a constraining model that makes constraints on the value domain of the CD, any of the translations may be the root CD. If the constraining model makes constraints on the value domain of the CD and there is a translation that meets the constraints, that translation should be the root CD. If the constraining model makes constraints on the value domain of the CD and there is no translation that meets the constraints, then any of the translations may be the root, as long as they are assigned a nullFlavor. An alternative is to put none of the translations in the root, and give it a NullFlavor of choice, and put all the translations in the translation property of the root.

7.5.2.4.10 codingRationale : CodingRationale: the reason why a particular CD has been provided, either as the root concept or as one of the translations.

If populated, the value contained in this attribute shall be taken from this enumeration, composed from the HL7 CodingRationale code system. The current values are:

CodingRationale Enumeration. OID: 2.16.840.1.113883.5.1074			
1	O	Original	Originally produced code.
1	P	Post-coded	Post-coded from free text source.
1	R	Required	Required by the specification describing the use of the coded concept. The exact form of the requirement is not specified here; it may be required by the specification directly, or it may arise as an indirect result of other conformance tools. More than one different requirement may exist simultaneously, so more than one code in a CD complex may be required.
1	OR	Original and required	Originally produced code, required by the specification describing the use of the coded concept.
1	PR	Post-coded and required	Post-coded from free text source, required by the specification describing the use of the coded concept.

ISO/IEC 11404 Syntax for codingRationale attribute

```
type CodingRationale = enumeration (O, P, R, OR, PR)
```

A code is deemed to be post-coded if the user does not assign the code when they first enter the data. codingRationale is not expected to act as a quality review marker on the quality of the coding or the translation processes.

A code is required when it is present in the instance to meet some constraints imposed on the instance by the context of use. Information Processing Entities shall not be required to mark a particular translation as required even though it is required by the context of use, but may do so. Information processing entities shall not reject instances because of the presence or absence of the codingRationale flag.

7.5.2.4.11 source : CD: A reference to the CD that was the source of this translation, if this CD was created by translating it from another CD.

This property is a reference. The source to which the reference points shall be provided within the scope of this CD's root CD and translations; that is, another representation of the same concept in the same attribute.

A CD consists of a single root code and a set of translations, which do not have translations. Using the codingRationale property, a sender can indicate which is the original code. There are some circumstances in which it is useful to know which CD was translated from which CD. The source allows for the translation sequence from one translation to another to be represented. Each element of the translation set was translated from the original CD. Each translation may, however, also have further translations. Thus, when a code is translated multiple times the information about which code served as the input to which translation will be preserved.

7.5.2.5 Equality

The equality of two CD values is determined solely based upon code and codeSystem. The codeSystemVersion, originalText, codingRationale, source, value set information and translations are not included in the equality test. NullFlavored values are not equal even if they have the same Nullflavor or the same original text.

The equality is based on the literal value of the code and codeSystem. Information processing entities shall not consult the semantic meaning of the code+codeSystem pair to determine whether the same concept is intended.

NOTE 1 This means that for SNOMED, for example, two isomorphic forms of the same expression will not be equal. Implementations should choose the literal representation forms carefully.

NOTE 2 CD values can also be equal with values of the CS datatype.

NOTE 3 CD values can also be equal with values of the CO datatype. For details, see 7.8.6.4.

7.5.2.6 Invariants

- if the value is not null then code or originalText shall have a value;
- if code has a value then codeSystem shall have a value;
- valueSet requires a valueSetVersion;
- codeSystemName can only have a value if codeSystem has a value;
- codeSystemVersion can only have a value if codeSystem has a value;
- displayName can only have a value if code has a value;
- translations cannot have original text;
- translations cannot have translations.

OCL for invariants:

```
def: let hasCode : Boolean = code.oclIsDefined
def: let hasCodeSystem : Boolean = codeSystem.oclIsDefined
def: let hasOriginalText : Boolean = originalText.isNotNull
def: let fromCodeSystem(system : Uid) : Boolean =
 codeSystem = system or translation->select(t |
 t.fromCodeSystem(system))->notEmpty
```

NOTE fromCodeSystem is defined so that you can make constraints from outside on the codeSystem of the CD or one of its translations, i.e. inv: code.fromCodeSystem("2.16.840.1.113883.6.42")

```
def: let noOriginalText : Boolean = originalText.oclIsUndefined

inv "null or (one or both of code and originalText)":
   isNotNull implies (hasCode or hasOriginalText)
 inv "other requires codeSystem or valueSet":
      (nullFlavor = NullFlavor.OTH) implies
      (codeSystem.oclIsDefined or valueSet.oclIsDefined)
inv "code requires codeSystem": code.oclIsDefined
      implies codeSystem.oclIsDefined
inv "codeSystemName only if codeSystem":
      codeSystemName.oclIsDefined implies codeSystem.oclIsDefined
```

```
inv "codeSystemVersion only if codeSystem":
      codeSystemVersion.oclIsDefined implies codeSystem.oclIsDefined
inv "displayName only if code": displayName.oclIsDefined
implies code.oclIsDefined
inv "valueSet requires valueSetVersion":
    valueSet.oclIsDefined implies (valueSetVersion.oclIsDefined)
inv "No original text on translations":
    translation->forAll(t | t.noOriginalText)
inv "Translations cannot have translations":
    translation->forAll(t | t.translation->isEmpty)
inv "no updateMode or History on CD elements":
noUpdateOrHistory(displayName)and
     noUpdateOrHistory(originalText) and
      translation->forAll(t | noUpdateOrHistory(t))
```

7.5.2.7 Operations

7.5.2.7.1 implies(other : CD):BL: True if this code+codeSystem is a specialization of the other code+codeSystem or has the same meaning.

NOTE 1 In SNOMED, for example, two isomorphic forms of the same expression will imply each other.

NOTE 2 A terminology service can be used to make this determination.

7.5.2.8 Examples

7.5.2.8.1 ICD Examples

A simple example for code is the ICD-9 code for headache, which is "784.0".

```
<example code="784.0" codeSystem="2.16.840.1.113883.6.42"
       codeSystemName="ICD-9">
  <displayName value="Headache"/>
  <originalText value="general headache"/>
</example>
```

A possible ICD-10 equivalent is "G44.1" (the ICD-10 classifications are slightly different).

```
<example code="G44.1" codeSystem="2.16.840.1.113883.6.3"
       codeSystemName="ICD-10">
  <displayName value="Headache"/>
  <originalText value="general headache"/>
</example>
```

7.5.2.8.2 Coding failure examples

A common situation with CD is when the actual concept cannot be properly represented in a particular coding system. Usually this circumstance arises where the concept is expected to be represented in a particular coding system. For the purposes of these examples, we assume that all these examples are for an observation value of type CD that is bound to the full Snomed-CT valueset (Example OID for the value set = 2.16.840.1.113883.19.11.1 as published June 11, 2007, Real OID for the SNOMED-CT code system = 2.16.840.1.113883.6.96). Important: the OID root 2.16.840.1.113883.19 is an HL7 OID used for example-only OIDs and OIDs in this space are never valid in real instances. The OIDs used in these examples that in the OID space 2.16.840.1.113883.6, 2.16.840.1.113883.5 and 2.16.840.1.113883.11 are the correct OIDs for use in production instances.

The simplest case is where the CD is not represented in the instance at all, or simply represented as no information.

```
<value nullFlavor="NI"/>
```

However this is not a very useful representation; frequently the source system knows more information, and it is still useful to convey that information to the destination system, while still labelling the coding as incomplete.

```
<value nullFlavor="OTH" codeSystem="2.16.840.1.113883.6.96"/>
```

Or it may be encoded as

```
<value nullFlavor="OTH" valueSet="2.16.840.1.113883.19.11.1"
valueSetVersion="20070711"/>
```

In this example, we specify valueSetVersion as a timestamp to the nearest day. The actual value allowed for the valueSetVersion depends on the definition of the value set. In this case, it is assumed that the definition of the value set 2.16.840.1.113883.19.11.1 specifies that the version is quoted to the day of the official release by the owning authority.

Both examples say that the concept cannot be coded in SNOMED. Even more useful is to convey some specific information about the concept, even though it cannot be represented in SNOMED:

```
<value nullFlavor="OTH" codeSystem="2.16.840.1.113883.6.96">
 <originalText value="Burnt ear with iron. Burnt other ear calling for
ambulance"/>
</value>
```

It is also possible that the content was first encoded in some other code system than SNOMED, and the source system was unable to encode the value in SNOMED. In this case, there are two forms of representation. The first is when the binding to SNOMED is labelled as CWE: local extensions are allowed:

```
<value code="burn" codeSystem="2.16.840.1.113883.19.5.2">
 <originalText value="Burnt ear with iron. Burnt other ear calling for
ambulance"/>
</value>
```

In this case, because the binding is CWE, local extensions are allowed, and the source system can simply use its own codeSystem (identified by the OID "2.16.840.1.113883.19.5.2", which is an example OID) to extend the other code system. In fact, the source system can also use a code from another well known code system, such as ICD-9. If ICD-9 had a code "A10.1" which stood for this same concept, then this would be valid:

```
<value code="A10.1" codeSystem="2.16.840.1.113883.6.42">
  <originalText value="Burnt ear with iron. Burnt other ear
   calling for ambulance"/>
</value>
```

If, however, the binding to the SNOMED-CT valueset is labelled CNE, then the code must come from SNOMED. The same information as the above-mentioned case must be conveyed differently:

```
<value nullFlavor="OTH" codeSystem="2.16.840.1.113883.6.96">
  <originalText value="Burnt ear with iron. Burnt other ear
   calling for ambulance"/>
  <translation code="burn" codeSystem="2.16.840.1.113883.19.5.2">
</value>
```

Now the code is clearly marked as OTH: the code cannot be represented in SNOMED-CT, but a translation from another system is provided. Though it is pretty redundant in this case, the source system could indicate which translation comes from which using the source property:

```
<value nullFlavor="OTH" codeSystem="2.16.840.1.113883.6.96">
  <originalText value="Burnt ear with iron. Burnt other ear
    calling for ambulance"/>
  <translation id="s1" code="burn"
    codeSystem="2.16.840.1.113883.19.5.2">
  <source xref="s1"/>
</value>
```

All these examples have assumed that the attribute is bound to the fictitious value set 2.16.840.1.113883.19.11.1 which is all of SNOMED-CT. If the value set was extended to include the LOINC codes as well, it would no longer be appropriate to encode a failure to code like this:

```
<value nullFlavor="OTH" codeSystem="2.16.840.1.113883.6.96"/>
```

since it is not true that the concept could not be coded from SNOMED-CT; it could not be coded in either SNOMED-CT or LOINC. For this reason, it is appropriate to encode the failure to code in the valueSet form:

```
<value nullFlavor="OTH" valueSet="2.16.840.1.113883.19.11.1" valueSetVersion="20070711"/>
```

7.5.2.8.3 Expression examples

Expressions generally arise with complex medical terminologies such as SNOMED. For example, SNOMED CT defines a concept "cellulitis (disorder)" (128045006) an attribute "finding site" (363698007) and another concept "foot structure (body structure)" (56459004). SNOMED CT allows these codes to be combined in a code phrase:

```
128045006|cellulitis (disorder)|:{363698007|finding site|=56459004|foot
structure|}
```

The full CD form for this is:

```
<value code="128045006:{363698007=56459004}"
    codeSystem="2.16.840.1.113883.6.42" codeSystemName="Snomed-CT">
  <originalText value="Cellulitis of the foot"/>
</value>
```

The SNOMED compositional expression language allows for the inclusion of the term in the expression, as shown in the first example. These make the expression more readable for humans, and so are used throughout this subclause in the standalone expressions. However, the terms are optional and do not improve readability for computers; instead, their optional presence creates needless processing complexity, such as for testing equality. For this reason the expressions in CD instances should not include the terms, and no CD examples include the terms in the expressions in this International Standard. Value sets may make rules about the presence or absense of the terms in the expressions.

The SNOMED compositional expression language is currently undergoing comment, and may be found on the International Health Terminology Standards Development Organisation (IHTSDO) website. The next two examples are based on SNOMED CT Core Edition 2007-01-31.

This first example is the SNOMED code for "fracture of left tibia". It shows issues associated with grouping and nesting.

```
31978002|fracture of tibia|: 272741003|laterality|=7771000|left|
```

Strictly speaking (in normal form) a "fracture of left tibia" is not a "left fracture" of a "tibia bone" but is a "fracture" of the "left" "tibia bone" (that is, the qualification of "left" applies to the bone not to the fracture). Also, in this example the fracture and bone are grouped; this may look irrelevant, but is potentially significant for combined fractures where different morphology may apply to different bones. An alternative rendering for this same concept is:

```
64572001|disease|:{116676008|associated morphology|=72704001|fracture|,
  363698007|finding site|=(12611008|bone structure of
    tibia|:272741003|laterality|=7771000|left|)}
```

The second example shows a more complicated grouping and nesting structure. The SNOMED CT expression for "past history of fracture of left tibia" includes nesting even in its simplest form because the laterality does not apply to the past history but rather to the disorder.

```
417662000|past history of clinical finding|:246090004|associated finding|=
     (31978002|fracture of tibia|: 272741003|laterality|=7771000|left|)
```

The alternative rendering is even more nested:

```
243796009|situation with explicit context|:246090004|associated finding|=
   (64572001|disease|:{116676008|associated morphology|=72704001|fracture|,
   363698007|finding site|=(12611008|bone structure of tibia|:
   272741003|laterality|=7771000|left|)}),408729009|finding context|=
   410515003|known present|,408731000|temporal context|=410513005|past|,
   408732007|subject relationship context|=410604004|subject of record|
```

These are provided as examples of SNOMED expression syntaxes. A full discussion of the merits of the different forms, their relationship and how to work with them can be found in the above-mentioned SNOMED compositional expression language definition.

It is important to note that the expression syntax and semantic rules are specified by the code system. For instance, in SNOMED CT, there are a defined set of qualifying attributes, and only Findings and Disorders can be qualified with the "finding site" attribute. CD does not provide for normalization of compositional expressions, therefore it is possible to create ambiguous expressions. Users should understand that they must provide the additional constraints necessary to assure unambiguous data representation if they are planning to create compositional expressions using CD. Otherwise, they risk the inability to retrieve a complete set of all records corresponding to any given query.

ICD-10 allows dual coding. See, for example, 3.1.3 of the ICD-10 Instruction Manual. While ICD-10 clearly establishes the semantic basis for the dual coding, it does not define an actual literal expression form suitable for use with CD. In such cases, HL7 defines a suitable literal expression form and assigns an OID to that. The OID for this ICD-10 expression is 2.16.840.1.113883.6.260. The code system specifies that the two ICD-10 codes are separated by a space.

```
<value code="J21.8 B95.6" codeSystem="2.16.840.1.113883.6.260"
      codeSystemName="ICD-10 Dual Code Expression">
  <originalText value="Staph aureus bronchiolitis"/>
</value>
```

The ICD-10 code J21.8 is "acute bronchiolitis due to other specified organisms" and the code B95.6 is "staphylococcus aureus as the cause of diseases classified to other chapters".

Expressions also arise in UCUM. Because UCUM is stable, UCUM expressions are usually found in the unit attribute of PQ (see 7.8.9.3.2). Although this is a CS, it is still a case of expressions in a code (a CS can be converted to a CD by filling out the codeSystem explicitly). The following is a simple UCUM expression that is actually a direct reference to a simple UCUM concept:

```
<value xsi:type="PQ" value="1" unit="g">
```

A gram is a definitional concept in UCUM. A typical UCUM expression is more complex:

```
<value xsi:type="PQ" value="1" unit="mmol/l">
```

The concentration of the analyte is 1 mmol/l; mmol/l is a UCUM expression, as is the simpler case:

```
<value xsi:type="PQ" value="1" unit="mmol">
```

The amount of analyte is 1 mmol; mmol is also a UCUM expression, a combination of m for milli and mol for moles.

7.5.3 CD.CV (coded value)

7.5.3.1 Description

A flavor that constrains CD.

Coded data, specifying only a code, code system, and optionally display name and original text.

Used only as the type of properties of other datatypes.

CV is used when any reasonable use case will require only a single code value to be sent. Thus, it should not be used in circumstances where multiple alternative codes for a given value are desired or there may be a requirement to migrate to a new coding system.

7.5.3.2 Invariants

- no translations are allowed;
- no source is allowed.

OCL for invariants:

```
inv "no translations": translation.size = 0
inv "no source": source.oclIsUndefined
```

7.5.4 CS (coded simple value)

7.5.4.1 Description

Specializes ANY.

Coded data in their simplest form, where only the code is not predetermined.

The code system and code system version are implied and fixed by the context in which the CS value occurs.

Due to its highly restricted functionality, CS shall only be used for simple structural attributes with highly controlled and stable terminologies where:

- all codes come from a single code system;
- codes are not re-used if their concept is deprecated;
- the publication and extensibility properties of the code system are well described and understood.

7.5.4.2 ISO/IEC 11404 syntax

```
type CS = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  code : characterstring
)
```

7.5.4.3 Attributes

7.5.4.3.1 code : String: The plain code symbol defined by the code system. if the code value is empty or null, then there is no code in the code system that represents the concept.

Code shall only contain characters that are either a letter, a digit or one of '.', '-', '_' or ':'. Code systems that are used with CS shall not define code symbols or expression syntaxes that contain whitespace or any other characters not in this list.

7.5.4.4 Equality

The equality of two CS values is determined solely based upon the explicit code and the implicit codeSystem. The codeSystemVersion is not included in the equality test. NullFlavored values are not equal even if they have the same NullFlavor.

The equality is based on the literal value of the code and codeSystem. Information Processing Entiies shall not consult the semantic meaning of the code+codeSystem pair to determine whether the same concept is intended.

NOTE CS values can be equal to CD values if both specify the same code and codeSystem.

7.5.4.5 Invariants

— there shall be a code if not nullFlavored.

OCL for invariants:

```
inv "code is required": isNotNull implies
code.oclIsDefined
```

7.5.4.6 Operations

7.5.4.6.1 codeSystem() : Uid: Although a CS does not carry a codeSystem as an attribute, there shall always be an applicable codeSystem specified by the context in which CS is used. This operation returns the codeSystem that is specified by the context.

This operation shall always return a valid codeSystem, whether or not the CS is nullFlavored, as it is taken from the context of use.

7.5.4.6.2 codeSystemVersion() : String: Although a CS does not carry a codeSystemVersion as an attribute, there may be an applicable version specified by the context in which CS is used.

This operation returns the codeSystemVersion if one is specified by the context.

7.5.4.7 Examples

```
<code xsi:type="CS" code="NS"/>
```

A simple code NS in the designated code system.

7.6 Identification and location datatypes

7.6.1 Overview

These datatypes provide support for identifying objects, records and things, and specifically for URLs, URIs and telecommunication addresses (see Figure 5).

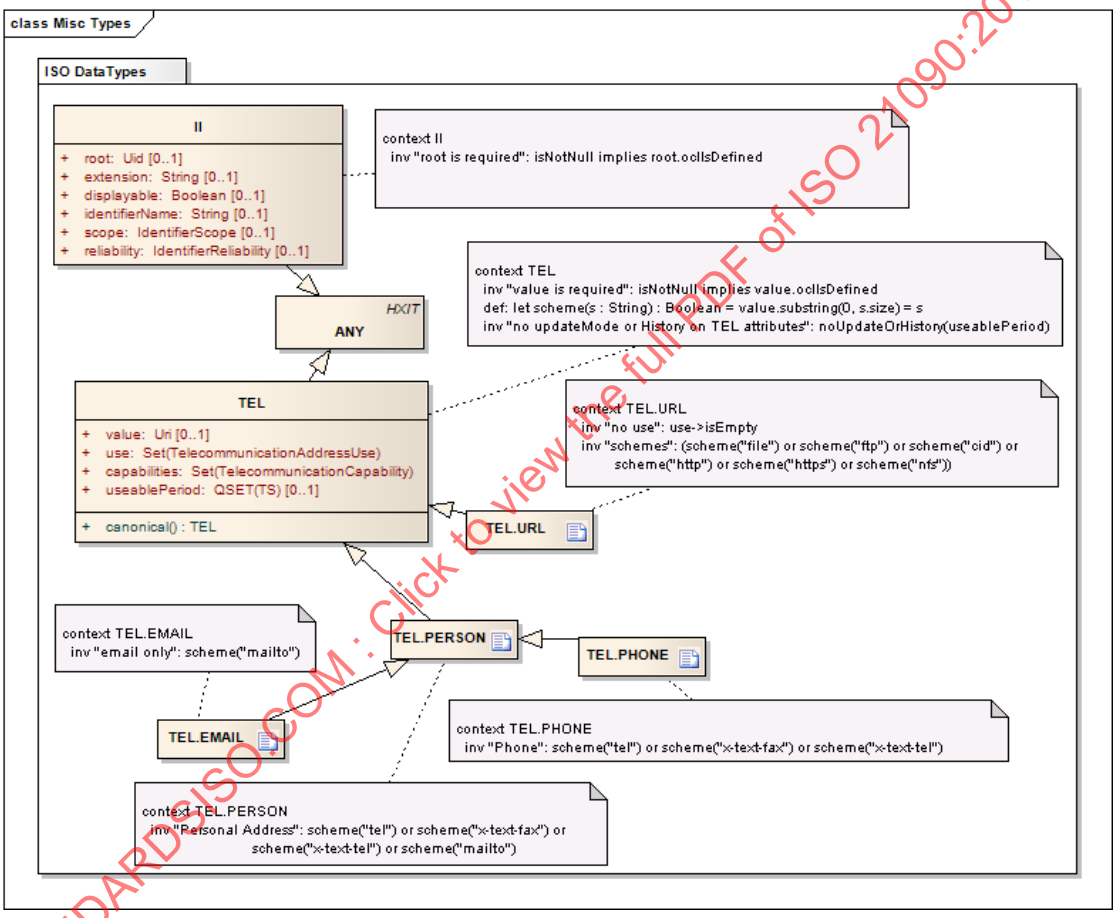

Figure 5 — Identification and location datatypes

7.6.2 TEL (Telecommunication Address)

7.6.2.1 Description

Specializes ANY

A locatable resource that is identified by a URI, such as a web page, a telephone number (voice, fax or some other resource mediated by telecommunication equipment), an e-mail address or any other locatable resource that can be specified by a URL.

The address is specified as a Universal Resource Locator (URL) qualified by time specification and use codes that help in deciding which address to use for a given time and purpose.

The value attribute is constrained to be a uniform resource locator specified according to IETF RFC 1738 and IETF RFC 3966 when used in this datatype.

NOTE The intent of this datatype is to be a locator, not an identifier; this datatype is used to refer to a locatable resource using an URL, and knowing the URL allows one to locate the object. However some use cases have arisen where a URI is used to refer to a locatable resource. Though this datatype allows for URIs to be used, the resource identified should always be locatable. A common use of locatable URI's is to refer to SOAP attachments.

7.6.2.2 ISO/IEC 11404 syntax

```
type TEL = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  value : characterstring,
  use : Set(TelecommunicationAddressUse),
  capabilities : Set(TelecommunicationCapability),
  useablePeriod : QSET(TS)
)
```

7.6.2.3 Attributes

7.6.2.3.1 value : Uri: A uniform resource identifier specified according to IETF RFC 2396.

The URI specifies the protocol and the contact point defined by that protocol for the resource.

EXAMPLE Notable uses of the telecommunication address datatype are for telephone and telefax numbers, e-mail addresses, Hypertext references, FTP references, etc.

If the TEL has a nullFlavor, it is not necessary for the value to contain a valid URL. For instance, if the flavor is UNK, the value may be just "tel:" to indicate that what is unknown is a telephone number.

7.6.2.3.2 use : Set(TelecommunicationAddressUse): One or more codes advising system or user which telecommunication address in a set of like addresses to select for a given telecommunication need.

The telecommunication use code is not a complete classification for equipment types or locations. Its main purpose is to suggest or discourage the use of a particular telecommunication address. There are no easily defined rules that govern the selection of a telecommunication address. Conformance statements may clarify what rules may apply or how additional rules are applied.

If populated, the values contained in this attribute shall be taken from the HL7 TelecommunicationAddressUse code system. The current values are:

TelecommunicationAddressUse Enumeration. OID: 2.16.840.1.113883.5.1011			
1	H	Home address	A communication address at a home, attempted contacts for business purposes might intrude privacy and chances are one contacts a family or other household members instead of the person one wishes to call. Typically used with urgent cases, or if no other contacts are available.
2	HP	Primary home	The primary home, to reach a person after business hours.
2	HV	Vacation home	A vacation home, to reach a person while on vacation.
1	WP	Work place	An office address. First choice for business-related contacts during business hours.
2	DIR	Direct	Indicates a workplace address or telecommunication address that reaches the individual or organization directly without intermediaries. For telephones, often referred to as a "private line".
2	PUB	Public	Indicates a workplace address or telecommunication address that is a "standard" address which may reach a reception service, mail-room, or other intermediary prior to the target entity.
1	BAD	Bad address	A flag indicating that the address is bad, in fact, useless.
1	TMP	Temporary address	A temporary address, might be good for visit or mailing. An address history can provide more detailed information.
1	AS	Answering service	An automated answering machine used for less urgent cases and if the main purpose of contact is to leave a message or gain access to an automated announcement.
1	EC	Emergency contact	A contact specifically designated to be used for emergencies. This is the first choice in emergencies, independent of any other use codes.
1	MC	Mobile contact	A telecommunication device that moves and stays with its owner. May have characteristics of all other use codes, suitable for urgent matters, not the first choice for routine business.
1	PG	Pager	A paging device suitable to solicit a callback or to leave a very short message

ISO/IEC 11404 syntax for telecommunicationAddressUse attribute

```
type TelecommunicationAddressUse = enumeration (H, HP, HV, WP, DIR,
PUB, BAD, TMP, AS, EC, MC, PG)
```

7.6.2.3.3 capabilities : Set(TelecommunicationCapability): One or more codes advising a system or user what telecommunication capabilities are known to be associated with the telecommunication address.

If populated, the values contained in this attribute shall be taken from the HL7 TelecommunicationCapability code system. The current values are:

TelecommunicationCapability Enumeration. OID: 2.16.840.1.113883.5.1118			
1	voice	Voice	This device can receive voice calls (i.e. talking to another person, a recording device or a voice activated computer)
1	fax	Fax	This device can receive faxes.
1	data	Data	This device can receive data calls (i.e. modem)
1	tty	Text	This device is a text telephone.
1	sms	SMS	This device can receive SMS messages

ISO/IEC 11404 syntax for TelecommunicationCapability attribute

```
 type TelecommunicationCapability = enumeration (voice, fax, data,
tty, sms)
```

7.6.2.3.4 useablePeriod: QSET(TS): The periods of time during which the telecommunication address can be used.

For a telephone number, this can indicate the time of day at which the party can be reached on that telephone. For a web address, it may specify a time range in which the web content is promised to be available under the given address.

7.6.2.4 Equality

Two nonNull TEL values are equal if their canonical forms have the same value attribute. The use and useablePeriod attributes are excluded from the equality test.

7.6.2.5 Invariants

— value shall be provided.

OCL for Invariants:

```
inv "value is required": isNotNull implies value.oclIsDefined
def: let scheme(s : String) : Boolean
      = value.substring(0, s.size) = s
inv "no updateMode or History on TEL attributes":
     noUpdateOrHistory(useablePeriod)
```

7.6.2.6 Operations

7.6.2.6.1 canonical : TEL: The TEL with any seperator or other non-significant characters stripped out of the address.

The tel: syntax allows for characters such as (and), which are syntactical separator characters, but do not change the actual telephone number. Canonical strips characters like these out of the address portion. The actual characters stripped out depend on the scheme.

The mailto: syntax allows for extra name and header information. This extra information is stripped out of the canonical form for the mailto: scheme, leaving just the plain email address(es).

7.6.2.7 Extensions to URL/URI syntax

This International Standard defines the following extensions to the URL scheme:

— x-text-tel: – indicates that the destination device is a text telephone; the syntax of the address portion of the URL is the same as TEL;

— x-text-fax: – indicates that the destination device is a fax machine; the syntax of the address portion of the URL is the same as TEL. This protocol replaces the deprecated W3C protocol fax.

This International Standard defines the following extensions to the **URN** scheme:

— hl7ii – a reference to an II value defined in this International Standard. The full syntax of the URN is urn:hl7ii:{root}[:{extension}] where {root} and {extension} (if present) are the values from the II that is being referenced. Full details of this protocol are defined in the HL7 Abstract Data Types Specification.

7.6.2.8 Examples

7.6.2.8.1 Web address

```
<example xsi:type="TEL" value="http://www.temp.org/example/234232"/>
```

A reference to the web page available from `http://www.temp.org/example/234232`.

7.6.2.8.2 Combined home and work telephone

```
<example xsi:type="TEL" value="tel:+15556755745"
   use="H WP" capabilities="voice fax"/>
```

A home (H) telephone number for a person who works at home (WP) that is capable of receiving both voice and fax calls.

7.6.2.8.3 Unknown home telephone number

```
<example xsi:type="TEL" nullFlavor="UNK" value="tel:" use="H"/>
```

An unknown home (H) telephone number.

7.6.2.8.4 Work telephone with extension

```
<example xsi:type="TEL" value="tel:+1(555)6755745;postd=545" use="WP"/>
```

A work telephone with an extension specified. Extensions are not the only use for the post-dial sequence. See IETF RFC 3966 for further details. The canonical form of this example is:

```
<tel value="tel:+15556755745;postd=545" use="WP"/>
```

7.6.3 TEL.URL

7.6.3.1 Description

A flavor that constrains TEL

TEL.URL constrains TEL so that it shall point to a locatable resource that returns binary content.

7.6.3.2 Invariants

— No use codes;

— The URL scheme shall be file, nfs, ftp, cid (for SOAP attachments), http, or https.

OCL for Invariants:

```
inv "no use": use->isEmpty
inv "schemes": (scheme("file") or scheme("ftp") or
scheme("cid") or scheme("http") or scheme("https")
 or scheme("nfs"))
```

7.6.4 TEL.PERSON

7.6.4.1 Description

A flavor that constrains TEL.

TEL.PERSON constrains TEL so that it shall refer to a method of communication with a person.

7.6.4.2 Invariants

— The URL scheme shall be tel, x-text-fax, x-text-tel or mailto.

OCL for Invariants:

```
inv "Personal Address": scheme("tel") or
      scheme("x-text-fax")or scheme("x-text-tel") or
      scheme("mailto")
```

7.6.5 TEL.PHONE

7.6.5.1 Description

A flavor that constrains TEL.PERSON

TEL.PHONE constrains TEL.PERSON so it shall refer to some telephone-based communication system with a person.

7.6.5.2 Invariants

— the URL scheme shall be tel, x-text-fax, or x-text-tel.

OCL for Invariants:

```
inv "Phone": scheme("tel") or scheme("x-text-fax") or
   scheme("x-text-tel")
```

7.6.6 TEL.EMAIL

7.6.6.1 Description

A flavor that constrains TEL.PERSON

TEL.EMAIL constrains the TEL.PERSON type to be an email address.

7.6.6.2 Invariants

— the URL scheme shall be mailto.

OCL for Invariants:

```
inv "email only": scheme("mailto")
```

7.6.7 II (Instance Identifier)

7.6.7.1 Description

Specializes ANY.

An identifier that uniquely identifies a thing or object.

EXAMPLE An object identifier for HL7 RIM objects, medical record number, order id, service catalogue item id, vehicle identification number (VIN), etc.

NOTE Instance identifiers are usually defined based on ISO object identifiers.

An identifier allows someone to select one record, object or thing from a set of candidates. Usually an identifier alone without any context is not usable. Identifiers are distinguished from concept descriptors as concept descriptors never identify an individual thing, although there may sometimes be an individual record or object that represents the concept.

Information processing entities claiming direct or indirect conformance shall never assume that receiving applications can infer the identity of issuing authority or the type of the identifier from the identifier or components thereof.

7.6.7.2 ISO/IEC 11404 syntax

```
type II = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  root : characterstring,
  extension : characterstring,
  identifierName : characterstring,
  displayable : boolean,
  scope: IdentifierScope,
  reliability : IdentifierReliability
)
```

7.6.7.3 Attributes

7.6.7.3.1 root : Uid: A unique identifier that guarantees the global uniqueness of the instance identifier.

If root is populated, and there is no nullFlavor or extension, then the root is a globally unique identifier in its own right. In the presence of a non-null extension, the root is the unique identifier for the "namespace" of the identifier in the extension. This does not necessarily correlate with the organization that manages the issuing of the identifiers. A given organization may manage multiple identifier namespaces, and control over a given namespace may transfer from organization to organization over time while the root remains the same.

This field can be either a DCE UUID, an Object Identifier (OID) or a special identifier taken from lists that may be published by ISO or HL7.

Comparison of root values is always case sensitive. UUIDs shall be represented in upper case, so UUID case should always be preserved.

The root shall not be used to carry semantic meaning; all it does is ensure global computational uniqueness.

7.6.7.3.2 extension: String: A character string as a unique identifier within the scope of the identifier root.

The root and extension scheme means that the concatenation of root and extension shall be a globally unique identifier for the item that this II value identifies.

Some identifier schemes define certain style options to their code values. For example the Social Security Number (SSN) of the USA is normally written with dashes that group the digits into a pattern "123-12-1234". However, the dashes are not meaningful and an SSN can also be represented as "123121234" without the dashes. In the case where identifier schemes provide for multiple representations, HL7 or ISO may make a ruling about which is the preferred form and document that ruling where that respective external identifier scheme is recognised.

If no extension attribute is provided in a non-null II, then the root is the complete unique identifier. If the root is not a complete unique identifier, and the extension is not known, then the II shall have a nullFlavor even if the root is populated.

7.6.7.3.3 identifierName : String: This is a human-readable name for the namespace represented in the root.

NOTE It is a descriptive name for the actual namespace, e.g. "California, US Driving Licence Number, 1970-".

IdentifierName does not refer to the organization which issued the identifier (e.g. California Department of Motor Vehicles). It is intended for use as a human readable label when an identifier shall be displayed to a human user where an OID would not be meaningful.

The identifier name has no computational value. IdentifierName can never modify the meaning of the root attribute. The purpose of the identifer name is to assist an unaided human interpreter of an II value to interpret the authority. Applications shall not attempt to perform any decision-making, matching, filtering or other processing based on the presence or value of this property. It is for display and development assistance only. All decision logic shall be based solely on the root and extension properties. Information processing entities claiming direct or indirect conformance may choose not to implement identifierName but shall not reject instances because identifierName is present.

In general, it should only be used when an extension is present, allowing for a display such as "California, US Driving Licence Number, 1970-: 123456789". There are absolutely no guidelines for the contents of this text other than it should be completely descriptive of the namespace, e.g. "Driving Licence" or even "California Driving Licence" would not be ideal. However, formatting, capitalization, whitepace, language, etc. are completely up to the sender.

7.6.7.3.4 displayable : Boolean: If the identifier is intended for human display and data entry (displayable = true) as opposed to pure machine interoperation (displayable = false).

Information processing entities claiming direct or indirect conformance may choose not to implement displayable but shall not reject instances because displayable is present.

7.6.7.3.5 scope: IdentifierScope : The scope in which the identifier applies to the object with which it is associated.

If populated, the value of this attribute shall be taken from the HL7 IdentifierScope code system. The current values are:

IdentifierScope Enumeration. OID: [not yet assigned]			
1	BUSN	Business identifier	An identifier whose scope is defined by business practices associated with the object. In contrast to the other scope identifiers, the scope of the use of the id is not necessarily restricted to a single object, but may be re-used for other objects closely associated with the object due to business practice.
1	OBJ	Object identifier	The identifier associated with a particular object. It remains consistent as the object undergoes state transitions.
1	VER	Version identifier	An identifier that references a particular object as it existed at a given point in time. The identifier shall change with each state transition on the object, i.e. The version identifier of an object prior to a "suspend" state transition is distinct from the identifier of the object after the state transition. Each version identifier can be tied to exactly one ControlAct event which brought that version into being (though the control act may never be instantiated). Applications that do not support versioning of objects shall ignore and not persist these ids to avoid confusion resulting from leaving the same identifier on an object that undergoes changes.
1	VW	View specific identifier	An identifier for a particular snapshot of a version of the object. This identifies a view of the business object at a particular point in time, and as such, identifies a set of data items that can be digitally signed and/or attested. This is in contrast to the version identifier which identifies the object at a specific time, but not the amount of information being asserted about the object. This identifier would be changed when a transformation of the information is performed (e.g. to add code translations, to provide a simplified textual rendering, or to provide additional information about the object as it existed at the specific point in time)

ISO/IEC 11404 syntax for identifierScope attribute

```
type IdentifierScope = enumeration (BUSN, OBJ, VER, VW)
```

7.6.7.3.6 reliability: IdentifierReliability : The reliability with which this identifier is known. This attribute may be used to assist with identifier matching algorithms.

If populated, the value of this attribute shall be taken from the HL7 IdentifierReliability code system. The current values are:

IdentifierReliability Enumeration. OID: [not yet assigned]			
1	ISS	Issued by system	The identifier was issued by the system responsible for constructing the instance.
1	VRF	Verified by system	The identifier was not issued by the system responsible for constructing the instance, but the system that captured the id has verified the identifier with the issuing authority, or with another system that has verified the identifier.
1	UNV	Unverified by system	The identifier was provided to the system that constructed the instance, but has not been verified, e.g. a driving licence entered manually into a system by a user.

ISO/IEC 11404 syntax for identifierReliability attribute

```
type IdentifierReliability = enumeration (ISS, VRF, UNV)
```

7.6.7.4 Equality

Two instance identifiers are equal if and only if they are not nullFlavored, their root is equal and their extensions are both null or equal. The displayable, identifierName, scope and reliability properties are ignored, though the scope and reliability properties may be used to determine whether the equality is significant in a given context.

7.6.7.5 Invariants

— a root shall be present if the II is not nullFlavored.

OCL for Invariants:

```
inv "root is required": isNotNull implies root.oclIsDefined
```

7.6.7.6 ISO/TS 22220 Comments

ISO/TS 22220 defines four fields for subject of care identifiers: designation, geographic, issuer, and type. Only the first, the designation, matches the scope of the II type. The designation is defined as:

> a number or code assigned to a person by an organization, establishment, agency or domain in order to uniquely identify that person as a subject of health care within that health care organization, establishment, agency or domain.

II fulfills this role by providing a unique identifier. When an II is used to identify a subject of care, the context of use should provide support for the scope, issuer and type properties.

NOTE The II provides a field called "IdentifierName" but this field does not provide formal support for identifying the issuer of the identifier.

7.6.7.7 Examples

7.6.7.7.1 Driving licence

```
<example xsi:type="II" root="2.16.840.1.113883.12.333" extension="45634353344"
reliability="UNV" scope="BUSN"/>
```

The OID 2.16.840.1.113883.12.333 has been issued by HL7 as a generic driving licence authority. The extension contains the actual driving licence number. The reliability is UNV – the value has been entered into the system, but the system cannot verify the number. The scope of the driving licence is BUSN, because the driving licence number may be used to identify several different objects associated with the same patient. This example uses the generic OID for a driving licence authority, but it is recommended not to use this in practice if possible because there are many driving licence authorities, and their issuing numbers will clash.

7.6.7.7.2 US SSN

```
<example xsi:type="II" root="2.16.840.1.113883.4.1" extension="123456789"
reliability="UNV" scope="BUSN"/>
```

Though the identifier is often formatted as 123-45-6789, the "-" should be removed so the right format is 123456789.

7.6.7.7.3 NHS Number

```
<example xsi:type="II" root="2.16.840.1.113883.2.1.4.1" extension="9999999484"
reliability="VRF" scope="BUSN"/>
```

This is an example of an NHS number from England/Wales. The NHS root OID is 2.16.840.1.113883.2.1.4.1. Because the NHS number is reused in multiple instances of a clinical document for the same patient, and for many other records, it generally does not have scope = OBJ – except on the NHS master repository itself. Usually the scope is BUSN. Most NHS systems maintain patient information interfaces with the master NHS patient registry, or with other systems that do, therefore the reliability in these cases is "VRF".

7.6.7.7.4 Australian medicare number

```
<example xsi:type="II" root="1.2.36.174030967" extension="1234567892"
reliability="VRF" scope="OBJ"/>
```

This is an example of an Australian medicare number. The root is the business root oid for the issuing authority HIC. (This should be a further qualified oid, but they have not, as yet, published their own scheme for their oid space.) The reliability is "VRF" because the sending system has checked that this is the correct identity for the patient with HIC itself using one of their electronic interfaces, and the scope is OBJ because this number is only used for identifying the account associated with this patient's family by the HIC.

7.6.7.7.5 Record identifier

```
<example xsi:type="II" root="D6A7AB37-4220-4D80-9052-8A4959A203E3"
reliability="ISS" scope="VER"/>
```

An UUID issued by the sending system associated with the particular version of the record being represented in the instance.

7.6.7.7.6 Lab number

```
<example xsi:type="II" root="2.16.840.1.113883.19.5.34" extension="2345344"
reliability="ISS" scope="OBJ"/>
```

A laboratory identifier that identifies the report. This has scope=”OBJ” and will be retained throughout the lab workflow as the sample is received, processed, reported and signed. This lab number is reported by the laboratory system itself which issued the identifier.

7.6.7.7.7 ISO 13606 record component id

```
<example xsi:type="II" root="2.16.840.1.113883.19.5.462" extension="976295765"
reliability="VRF" scope="VER"/>
```

ISO 13606-1 rc_id (record component id), which is persisted across EHR repositories, but not re-used across different versions of the record and not re-used between workflow state changes. Hence the scope is "VER". The reliability is VRF; in the context of ISO 13606 the use of ISS is not recommended as it can only be used by the primary issuing system, and this makes it unfeasible to attest to the record.

The primary use of ISS is in patient management etc., where it can be used to help build linking/unlinking/merging workflows.

7.6.7.7.8 Message snapshot

```
<example xsi:type="DSET_II">
  <item root="2.16.840.1.113883.19.5.971" extension="763491"
     reliability="VRF" scope="OBJ"/>
  <item root="2.16.840.1.113883.19.5.972" extension="351324"
     reliability="VRF" scope="VER"/>
  <item root="0282CA34-2E4E-4B9D-82A5-BD2BF8497940"
     reliability="ISS" scope="VW"/>
</example>
```

This example shows the full use of a DSET(II) in a message. The message carries a snapshot of an object in a class in the message, and the class has an attribute id : DSET(II). The information in in the class that carries the DSET(II) attribute is based on the object 763491 in the OID space 2.16.840.1.113883.19.5.971. The version on which the snapshot is based is identified by the identifier 351324 in the OID space 2.16.840.1.113883.19.5.972. Finally, the snapshot is given its own UUID identifier 0282CA34-2E4E-4B9D-82A5-BD2BF8497940, which may be used if a system wished to record that particular snapshot it its audit trail.

7.7 Name and address datatypes

7.7.1 Overview

These datatypes provide support for names and addresses(see Figure 6).

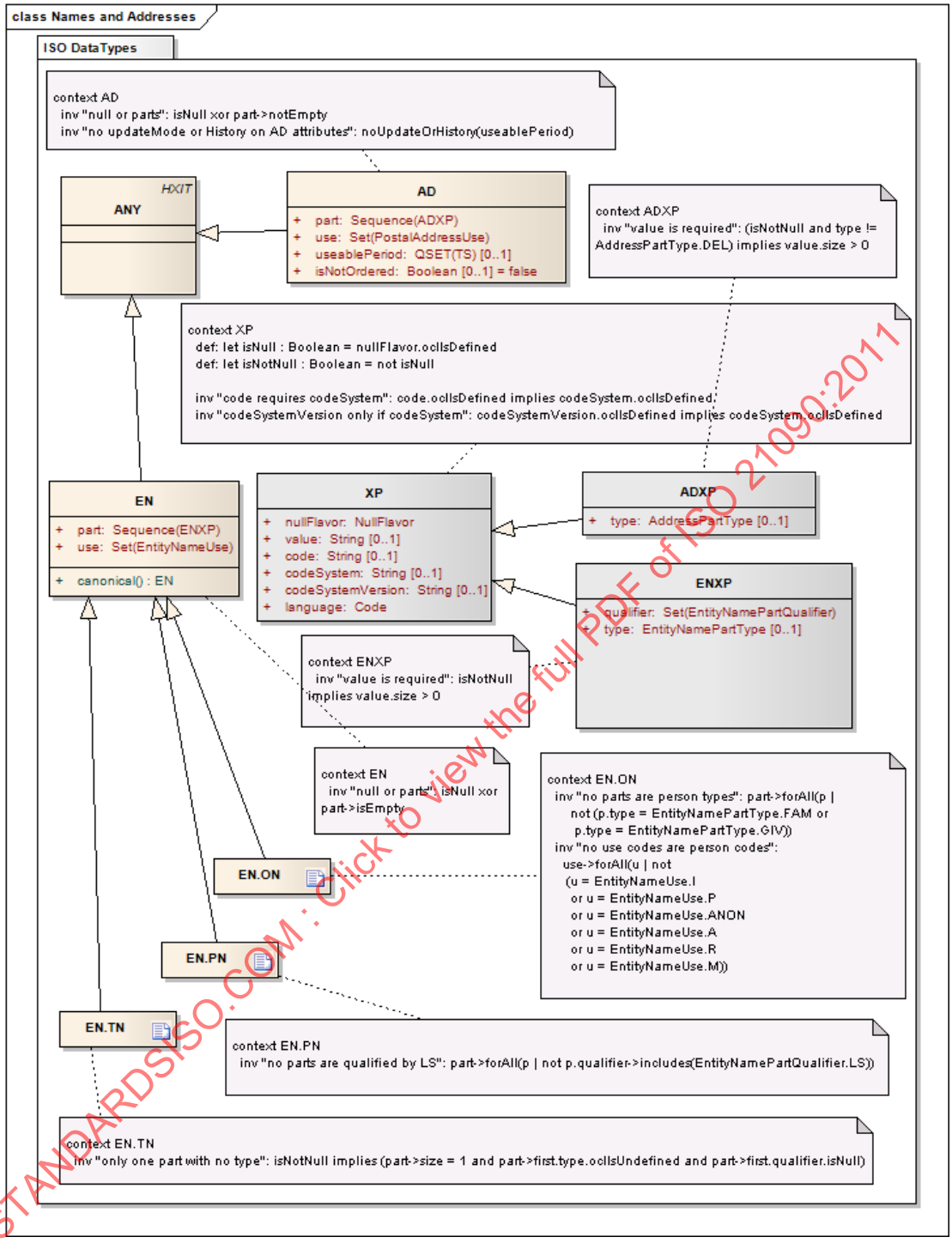

Figure 6 — Name and address datatypes

7.7.2 XP (name or address part)

Abstract private type

7.7.2.1 Description

A part of a name or address. Each part is a character string that may be coded, and that also may have a nullFlavor. The string content shall always be provided whether a code is provided or not.

7.7.2.2 ISO/IEC 11404 syntax

```
type XP = class (
  nullFlavor : NullFlavor,
  value : characterstring,
  code : characterstring,
  codeSystem : characterstring,
  codeSystemVersion : characterstring,
  language : characterstring,
)
```

7.7.2.3 Attributes

7.7.2.3.1 nullFlavor : NullFlavor: If the part is not a proper value, indicates the reason.

For further information concerning nullFlavor, see 7.3.3.3.1.

7.7.2.3.2 value : String: The actual string value of the part. If no nullFlavor is provided, some content shall be present in this attribute.

7.7.2.3.3 code : String: A code assigned to the part by some coding system, if appropriate.

7.7.2.3.4 codeSystem : String: The code system from which the code is taken.

The choice of coding system depends on the part type defined in the concrete specializations. The codeSystem shall be populated if a code is populated.

7.7.2.3.5 codeSystemVersion : String: The version of the coding system, if required.

The codeSystem shall be populated if a codeSystemVersion is populated.

7.7.2.3.6 language: Code: The human language of the content. Valid codes are taken from the IETF RFC 3066. If this attribute is null, the language may be inferred from elsewhere, either from the context or from unicode language tags, for example.

While parts may be assigned a language, the meaning of the part is not dependent on the language, and applications shall not be required to indicate the linguistic origin of any name or address part.

7.7.2.4 Equality

There is no definition of equality for values of type XP.

7.7.2.5 Invariants

— if code has a value then codeSystem shall have a value;

— codeSystemVersion can only have a value if codeSystem has a value.

OCL for Invariants:

```
def: let isNull : Boolean = nullFlavor.oclIsDefined
def: let isNotNull : Boolean = not isNull

inv "code requires codeSystem": code.oclIsDefined implies
codeSystem.oclIsDefined
inv "codeSystemVersion only if codeSystem":
codeSystemVersion.oclIsDefined implies
codeSystem.oclIsDefined
```

7.7.3 ADXP (address part)

7.7.3.1 Description

Specializes XP.

A part that may have a type-tag signifying its role in the address. Typical parts that exist in about every address are street, house number or post box, postal code, city, country, but other roles may be defined regionally, nationally or on an enterprise level (e.g. in military addresses).

Addresses are usually broken up into lines, which may be indicated by special line-breaking delimiter elements (e.g. DEL).

7.7.3.2 ISO/IEC 11404 Syntax

```
type ADXP = class (
  nullFlavor : NullFlavor,
  value : characterstring,
  code : characterstring,
  codeSystem : characterstring,
  codeSystemVersion : characterstring,
  language : characterstring,
  type : AddressPartType,
)
```

7.7.3.3 Attributes

7.7.3.3.1 type : AddressPartType: Whether an address part names the street, city, country, postal code, post box, etc.

If the type is NULL the address part is unclassified and would simply appear on an address label as is.

If populated, the value of this attribute shall be taken from the HL7 AddressPartType code system. The current values are:

AddressPartType Enumeration. OID: 2.16.840.1.113883.5.16			
1	AL	Address line	An address line is for either an additional locator, a delivery address or a street address. An address generally has only a delivery address line or a street address line, but not both.
2	ADL	Additional locator	This can be a unit designator, such as apartment number, suite number or floor. There may be several unit designators in an address (e.g. "thirrd floor, Appt. 342"). This can also be a designator pointing away from the location, rather than specifying a smaller location within some larger one (e.g. Dutch "t.o." means "opposite to" for house boats located across the street facing houses).
3	UNID	Unit identifier	The number or name of a specific unit contained within a building or complex, as assigned by that building or complex.
3	UNIT	Unit designator	Indicates the type of specific unit contained within a building or complex, e.g. apartment, floor.
2	DAL	Delivery address line	A delivery address line is frequently used instead of breaking out delivery mode, delivery installation, etc. An address generally has only a delivery address line or a street address line, but not both.

3	DINST	Delivery installation type	Indicates the type of delivery installation (the facility to which the mail will be delivered prior to final shipping via the delivery mode) e.g. post office, letter carrier depot, community mail centre, station.
3	DINSTA	Delivery installation area	The location of the delivery installation, usually a town or city, and only required if the area is different from the municipality. Area to which mail delivery service is provided from any postal facility or service such as an individual letter carrier, rural route or postal route.
3	DINSTQ	Delivery installation qualifier	A number, letter or name identifying a delivery installation, e.g. for Station A, the delivery installation qualifier would be "A".
3	DMOD	Delivery mode	Indicates the type of service offered, method of delivery, e.g. post office box, rural route, general delivery.
3	DMODID	Delivery mode identifier	Represents the routing information, such as a letter carrier route number. It is the identifying number of the designator (the box number or rural route number).
2	SAL	Street address line	A street address line is frequently used instead of breaking out building number, street name, street type, etc. An address generally has only a delivery address line or a street address line, but not both.
3	BNR	Building number	The number of a building, house or lot alongside the street. Also known as "primary street number". This does not number the street but rather the building.
4	BNN	Building number numeric	The numeric portion of a building number.
4	BNS	Building number suffix	Any alphabetic character, fraction or other text that may appear after the numeric portion of a building number.
3	STR	Street name	The name of the street, including the type.
4	STB	Street name base	The base name of a roadway or artery recognised by a municipality (excluding street type and direction).
4	STTYP	Street type	The designation given to the street. (e.g. Street, Avenue, Crescent).
3	DIR	Direction	Direction (e.g. N, S, W, E).
2	INT	Intersection	Denotes that the actual address is located at or close to the intersection of two or more streets.
1	CAR	Care of	The name of the party who takes receipt at the specified address, and takes on responsibility for ensuring delivery to the target recipient. NOTE This is included only to support the convention of writing c/- address lines. This item is not appropriate for use when information is entrusted to one party on behalf of another in some significant way.
1	CEN	Census tract	A geographic subunit delineated for demographic purposes.
1	CNT	Country	Country.
1	CPA	County or parish	A subunit of a state or province. (49 of the United States of America use the term "county"; Louisiana uses the term "parish").

1	CTY	Municipality	The name of the city, town, village or other community or delivery centre.
1	DEL	Delimiter	Delimiters are printed without framing white space. If no value component is provided, the delimiter appears as a line break.
1	POB	Post box	A numbered box located in a post station.
1	PRE	Precinct	A subsection of a municipality.
1	STA	State or province	A subunit of a country with limited sovereignty in a federally organized country.
1	ZIP	Postal code	A postal code designating a region defined by the postal service.
1	DPID	Delivery point identifier	A value that uniquely identifies the postal address.
NOTE The hierarchical nature of this code system shows composition rather than subsumption, e.g. "Street Name" is part of "Street Address Line".			

ISO/IEC 11404 Syntax for type Attribute

```
type AddressPartType = enumeration (AL, ADL, UNID, UNIT, DAL, DINST,
DINSTA, DINSTQ, DMOD, DMODID, SAL, BNR, BNN, BNS, STR, STB, STTYP,
DIR, INT, CAR, CEN, CNT, CPA, CTY, DEL, POB, PRE, STA, ZIP)
```

7.7.3.4 Equality

Two ADXP values are equal if their type and value attributes are equal. The code attributes and language are ignored.

NOTE Clarification: two type attributes of null are considered equal.

7.7.3.5 Invariants

— If the part is nonNull, the value cannot be empty unless the part type is DEL.

OCL for Invariants:

```
inv "value is required": isNotNull implies value.size > 0
```

7.7.3.6 Binding

For the code+codeSystem properties inherited from XP, the part type CNT (country) is bound to the codes defined in ISO 3166, either the 2- or 3-letter alphabetic codes or the numeric codes. Conformance statements may specify bindings for other part types or restrict the choice of codes for country.

7.7.4 AD (address)

7.7.4.1 Description

Specializes ANY.

Mailing and home or office addresses. AD is primarily used to communicate data that allows printing mail labels, or that allows a person to physically visit that address. The postal address datatype is not supposed to be a container for additional information that might be useful for finding geographic locations (e.g. GPS coordinates) or for performing epidemiological studies. Such additional information should be captured by other, more appropriate data structures.

Addresses are essentially sequences of address parts, but add a "use" code and a valid time range for information about if and when the address can be used for a given purpose.

7.7.4.2 ISO/IEC 11404 Syntax

```
type AD = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  part : Sequence(ADXP),
  use : Set(PostalAddressUse),
  useablePeriod : QSET(TS),
  isNotOrdered : boolean
)
```

7.7.4.3 Attributes

7.7.4.3.1 part : Sequence(ADXP): A sequence of address parts, such as street or post office box, city, postal code, country.

7.7.4.3.2 use : Set(PostalAddressUse): A set of codes advising a system or user which address in a set of like addresses to select for a given purpose.

An address without specific use code might be a default address useful for any purpose, but an address with a specific use code would be preferred for that respective purpose.

If populated, the values contained in this attribute shall be taken from the HL7 PostalAddressUse code system. The current values are:

PostalAddressUse Enumeration. OID: 2.16.840.1.113883.5.1012			
1	*AddressUse*		
2	H	Home address	A communication address at a home; attempted contacts for business purposes might intrude privacy and chances are one will contact family or other household members instead of the person one wishes to call. Typically used with urgent cases, or if no other contacts are available.
3	HP	Primary home	The primary home, to reach a person after business hours.
3	HV	Vacation home	A vacation home, to reach a person while on vacation.
2	WP	Work place	An office address; first choice for business-related contacts during business hours.
3	DIR	Direct	Indicates a workplace address or telecommunication address that reaches the individual or organization directly without intermediaries. For telephones, often referred to as a "private line".
3	PUB	Public	Indicates a workplace address or telecommunication address that is a "standard" address which may reach a reception service, mailroom, or other intermediary prior to the target entity.
2	BAD	Bad address	A flag indicating that the address is bad, in fact, useless.
2	PHYS	Physical visit address	Used primarily to visit an address.

2	PST	Postal address	Used to send mail.
2	TMP	Temporary address	A temporary address, may be good for visit or mailing. An address history can provide more detailed information.
1	*AddressRepresentationUse.* Identifies the different representations of the address. The representation may affect how the address is used (e.g. use of ideographic for formal communications).		
2	ABC	Alphabetic	Alphabetic transcription of name (Japanese: romaji).
2	IDE	Ideographic	Ideographic representation of name (e.g. Japanese kanji, Chinese characters).
2	SYL	Syllabic	Syllabic transcription of name (e.g. Japanese kana, Korean hangul).
1	SRCH	Search type uses	A name intended for use in searching or matching.
2	SNDX	Soundex	An address spelled according to the SoundEx algorithm.
2	PHON	Phonetic	The address as understood by the data enterer, i.e. a close approximation of a phonetic spelling of the address, not based on a phonetic algorithm.

ISO/IEC 11404 Syntax for the postalAddressUse attribute

```
type PostalAddressUse = enumeration (H, HP, HV, WP, DIR, PUB, BAD,
TMP, ABC, IDE, SYL, PHYS, PST, SRCH, SNDX, PHON)
```

7.7.4.3.3 useablePeriod : QSET(TS): A General Timing Specification (GTS) specifying the periods of time during which the address can be used. This is used to specify different addresses for different times of the week or year.

7.7.4.3.4 isNotOrdered : Boolean: A boolean value specifying whether the order of the address parts is known or not. While the address parts are always a sequence, the order in which they are presented may or may not be known to be true or important. Where this matters, the isNotOrdered property can be used to convey this information. The default value for isNotOrdered is false.

7.7.4.4 Equality

Two address values are considered equal if they contain the same address parts, independent of ordering. Use code, useablePeriod and isNotOrdered are excluded from the equality test.

NOTE 1 Even if isNotOrdered is false – it is known that the order of the address parts is representationally significant – the order of the parts is irrelevant for checking equality of addresses.

NOTE 2 Two values that refer to the same address but that are encoded using different address parts (perhaps to different levels of detail) would not be considered equal.

7.7.4.5 Invariants

— either the AD is nullFlavored or it has at least one part.

OCL for Invariants:

```
inv "null or parts": isNull xor part->notEmpty
inv "no updateMode or History on AD attributes":
    noUpdateOrHistory(useablePeriod)
```

7.7.4.6 ISO/TS 22220 comments

The various address parts defined by ISO/TS 22220, map to address part types, and the address type maps to the use attribute. The start and end date accuracy indicators are partially supported by the precision of the dates provided.

7.7.4.7 Examples

7.7.4.7.1 Address with layout

```
<example xsi:type="AD" use="WP">
   <part value="1050 W Wishard Blvd" />
   <part type="DEL"/>
   <part value="RG 5th floor"/>
   <part type="DEL"/>
   <part value="Indianapolis, IN 46240"/>
</example>
```

This work address consists of three unknown parts with two line delimiters. None of the parts is labelled with regard to their semantic significance.

7.7.4.7.2 Address with types

```
<example xsi:type="AD" use="WP">
   <part type="AL" value="1050 W Wishard Blvd"/>
   <part type="AL" value="RG 5th floor"/>
   <part type="CTY" value="Indianapolis"/>
   <part type="STA" value="IN"/>
   <part type="ZIP" value="46240"/>
</example>
```

This is the same address using standard typing rather than a presentation focus. This is probably the most common form of presentation for addresses – a series of address lines followed by city, state, zip and possibly country.

NOTE Although this presentation of the address suggests that lines are required after the two address lines, this is not implied by this example (see 7.7.4.8).

7.7.4.7.3 Line types

```
<example xsi:type="AD" use="WP">
   <part type="SAL" value="1050 W Wishard Blvd"/>
   <part type="ADL" value="RG 5th floor"/>
   <part type="CTY" value="Indianapolis"/>
   <part type="STA" value="IN"/>
   <part type="ZIP" value="46240"/>
</example>
```

This is the same address from a system that differentiates between different line types.

7.7.4.7.4 Fully typed addresses

```
<example xsi:type="AD" use="WP">
   <part type="BNR" value="1050"/>
   <part type="DIR" value="W"/>
   <part type="STB" value="Wishard"/>
   <part type="STTYP" value="Blvd"/>
   <part type="ADL" value="RG 5th floor"/>
   <part type="CTY" value="Indianapolis"/>
   <part type="STA" value="IN"/>
   <part type="ZIP" value="46240"/>
</example>
```

The same address fully broken down; the above-mentioned form is not used in the USA. However, it is useful in Germany, where many systems keep house number as a distinct field.

```
<example xsi:type="AD" use="HP">
   <part type="STR" value="Windsteiner Weg"/>
   <part type="BNR" value="54a"/>
   <part type="CNT" code="DEU" codeSystem=" 1.0.3166.1.2"
      value="D"/>
   <part type="ZIP" value="14165"/>
   <part type="CTY" value="Berlin"/>
</example>
```

This is a home address in a standard German format. The country has been coded in ISO 3166 to assist with interoperability.

7.7.4.7.5 Unknown addresses

```
<example xsi:type="AD" use="WP" nullFlavor="UNK"/>
```

The work address is unknown.

7.7.4.8 Presenting addresses

The primary purpose of an address is to be presented on a delivery label affixed to an envelope. A fully specified address (one that includes specified line breaks) can be presented directly by simply presenting the text of the various parts with whitespace separating them, and following the explicit line breaks. If the elements are moved into the xhtml namespace, the AD content can be treated as html directly.

For this reason, the address should always be generated with appropriate line breaks included in the address. This enables applications that do not understand the semantics of the address to reproduce it correctly.

However, because there is no single presentation model for addresses, applications may ignore the explicitly specified line breaks in addresses; they are not bound to follow the presentation as specified in any particular address.

7.7.5 ENXP (Entity Name Part)

7.7.5.1 Description

Specializes XP.

A part that may have a type code signifying the role of the part in the whole entity name, and qualifier codes for more detail about the name part type. (Typical name parts for person names are given names, and family names, titles, etc.)

7.7.5.2 ISO/IEC 11404 syntax

```
type ENXP = class (
  nullFlavor : NullFlavor,
  value : characterstring,
  code : characterstring,
  codeSystem : characterstring,
  codeSystemVersion : characterstring,
  language : characterstring,
  type : EntityNamePartType,
  qualifier : Set(EntityNamePartQualifier)
)
```

7.7.5.3 Attributes

7.7.5.3.1 type : EntityNamePartType: Indicates whether the name part is a given name, family name, prefix, suffix, etc.

Not every name part shall have a type code, if the type code is unknown, not applicable, or simply undefined this is expressed by a null value (type.isNull). For example a name may be "Rogan Sulma" and it might not be clear which one is a given name or which is a last name, or whether Rogan is a title.

If populated, the value of this attribute shall be taken from the HL7 EntityNamePartType2 code system. The current values are:

EntityNamePartType Enumeration. OID: 2.16.840.1.113883.5.1121			
1	FAM	Family	Family name, this is the name that links to the genealogy. In some cultures (e.g. Eritrea), the family name of a son is the first name of his father.
1	GIV	Given	Given name. NOTE Not to be called "first name" since given names do not always come first.
1	TITLE	Title	Part of the name that is acquired as a title due to academic, legal, employment or nobility status, etc. NOTE Title name parts include name parts that come after the name, such as qualifications.
1	DEL	Delimiter	A delimiter has no meaning other than being literally printed in this name representation. A delimiter has no implicit leading and trailing white space.

ISO/IEC 11404 Syntax for the entityNamePartType attribute

```
type EntityNamePartType = enumeration (FAM, GIV, TITLE, DEL)
```

When a name is hyphenated, such as Mary-Ann, it may be ambigious whether to use a delimiter separating two name parts, or a single name part with a hyphen in it. As a rule of thumb, if each name part should contribute an initial when the name is presented as initials, then a delimiter should be used to separate two parts.

7.7.5.3.2 qualifier : Set(EntityNamePartQualifier): The qualifier is a set of codes each of which specifies a certain subcategory of the name part in addition to the main name part type.

EXAMPLE: A given name can be flagged as a nickname (CL), a family name might be a name acquired by marriage (SP) or a name from birth (BR).

If populated, the values contained in this attribute shall be taken from the HL7 EntityNamePartQualifier2 code system. The current values are:

EntityNamePartQualifier Enumeration. OID: 2.16.840.1.113883.5.1122			
1	LS	Legal status	For organizations a suffix indicating the legal status, e.g. "Inc.", "Co.", "AG", "GmbH", "B.V." "S.A.", "Ltd.".
1	*TitleStyles:* Extra information about the style of a title		
2	AC	Academic	Indicates that a prefix like "Dr." or a suffix like "M.D." or "Ph.D." is an academic title.

2	NB	Nobility	In Europe and Asia, there are still people with nobility titles (aristocrats). German "von" is generally a nobility title, not a mere voorvoegsel. Others are "Earl of" or "His Majesty King of..." etc. Rarely used nowadays, but some systems do keep track of this.
2	PR	Professional	Primarily in the British Imperial culture people tend to have an abbreviation of their professional organization as part of their credential suffices.
2	HON	Honorific	An honorific such as “The Right Honourable” or “Weledelgeleerde Heer”.
1	BR	Birth	A name that a person was given at birth or established as a consequence of adoption. NOTE This is not used for temporary names assigned at birth such as “Baby of Smith” – which is just a name with a use code of “TEMP”.
1	AD	Acquired	A name part that a person acquired. The name part may be acquired by adoption or the person may have chosen to use the name part for some other reason. NOTE This differs from an other/psuedonym/alias in that an acquired name part is acquired on a formal basis rather than an informal one (e.g. registered as part of the official name).
2	SP	Spouse	The name assumed from the partner in a marital relationship. Usually the spouse’s family name. No inference about gender may be made from the existence of spouse names.
1	MID	Middle Name	Indicates that the name part is a middle name. In general, the English "middle name" concept is all of the given names after the first. This qualifier may be used to explicitly indicate which given names are considered to be middle names. The middle name qualifier may also be used with family names. This is a Scandinavian use case, matching the concept of "mellomnavn"/"mellannamn". There are specific rules that indicate what names may be taken as a mellannamn in different Scandinavian countries.
1	CL	Callme	Callme is used to indicate which of the various name parts is used when interacting with the person.
1	IN	Initial	Indicates that a name part is just an initial. Initials do not imply a trailing period since this would not work with non-Latin scripts. Initials may consist of more than one letter, e.g. "Ph." could stand for "Philippe" or "Th." for "Thomas".
1	PFX	Prefix	A prefix has a strong association to the name part immediately following. A prefix has no implicit trailing white space (although it has implicit leading white space).
1	SFX	Suffix	A suffix has a strong association to the immediately preceding name part. A suffix has no implicit leading white space (although it has implicit trailing white space).

ISO/IEC 11404 Syntax for the entityNamePartQualifier attribute

```
type EntityNamePartQualifier = enumeration (LS, AC, NB, PR, HON, BR,
AD, SP, MID, CL, IN, PFX, SFX)
```

The Scandinavian “Mellomnavn/Mellannamn” translates to “middle name” but does not match the English "middle name" concept. The general English "middle name" concept is simply all of the given names after the first. The qualifiers PFX and SFX are mutually incompatible. It is not legal to use both on the same part type. It is not necessary to label the name part following a prefix as a suffix or vice versa.

NOTE Initials are allowed to be more than one letter specifically to cater for linguistic norms in the applicable language. Abbreviations, such as Dr. for Doctor, are not initials.

7.7.5.4 Equality

Two ENXP values are equal if their type and value attributes are equal. The code attributes, language and qualifier are ignored.

NOTE Clarification: two type attributes of null are considered equal.

7.7.5.5 Invariants

— if the part is nonNull, the value cannot be empty.

OCL for Invariants:

```
inv "value is required": isNotNull implies value.size > 0
```

7.7.5.6 Binding

Conformance statements may specify bindings for the various part types.

7.7.5.7 Implementation notes

There is a relationship between the part type and the qualifiers which can be used. This table summarizes the qualifiers that can be used with the different part types:

	FAM (family)	GIV (given)	TITLE (title)	DEL (delimiter)	null
LS (legal status)			✓✓		✓
AC (academic)			✓✓		✓
NB (nobility)			✓✓		✓
PR (professional)			✓✓		✓
HON (honorifc)			✓✓		✓
BR (birth)	✓✓	✓✓			✓
AD (adopted)	✓✓	✓✓			✓
SP (spouse)	✓✓	✓			✓
MID (middle name)	✓✓	✓✓			✓
CL (call me)	✓	✓✓			✓
IN (initial)	✓	✓✓			✓
PFX (prefix)	✓✓	✓✓	✓✓		✓
SFX (suffix)	✓✓	✓✓	✓✓		✓
✓ = This combination is allowed, though it is not expected to be in common usage. ✓✓ = This combination is allowed, and it is expected that this combination will be encountered in practice.					

7.7.6 EN (entity name)

7.7.6.1 Description

Specializes ANY.

A name for a person, organization, place or thing.

EXAMPLE "Jim Bob Walton, Jr.", "Health Level Seven, Inc.", "Lake Tahoe", etc. An entity name can be as simple as a character string or can consist of several entity name parts, such as, "Jim", "Bob", "Walton", and "Jr.", "Health Level Seven" and "Inc.".

Entity names are essentially sequences of entity name parts, but add a "use" code and a valid time range for information about when the name was used and how to choose between multiple aliases that may be valid at the same time.

7.7.6.2 ISO/IEC 11404 syntax

```
type EN = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  part : Sequence(ENXP),
  use : Set(EntityNameUse)
)
```

7.7.6.3 Attributes

7.7.6.3.1 part : Sequence(ENXP): A sequence of name parts, such as given name or family name, prefix, suffix.

7.7.6.3.2 use : Set(EntityNameUse): A set of codes advising a system or user which name in a set of names to select for a given purpose.

A name without specific use code might be a default name useful for any purpose, but a name with a specific use code would be preferred for that respective purpose. Names should not be collected without at least one use code, but names may exist without use code, particularly for legacy data.

If populated, the values contained in this attribute shall be taken from the HL7 EntityNameUse2 code system. The current values are:

EntityNameUse Enumeration. OID: 2.16.840.1.113883.5.1120			
1	*RepresentationUse.* Identifies the different representations of a name. The representation may affect how the name is used. (E.g. use of Ideographic for formal communications)		
2	ABC	Alphabetic	Alphabetic transcription of name (Japanese: romaji).
2	IDE	Ideographic	Ideographic representation of name (e.g. Japanese kanji, Chinese characters).
2	SYL	Syllabic	Syllabic transcription of name (e.g. Japanese kana, Korean hangul).
1	C	Customary	Known as/conventional/the one you normally use.

1	OR	Official registry name	The formal name as registered in an official (government) registry, but which name might not be commonly used. May correspond to the concept of legal name.
1	T	Temporary	A temporary name. A name valid time can provide more detailed information. This may also be used for temporary names assigned at birth or in emergency situations.
1	*Assumed*: A name that a person has assumed or has been assumed to them		
2	I	Indigenous/Tribal	Chief Red Cloud, for example.
2	P	Other/ pseudonym/alias	A non-official name by which the person is sometimes known. (This may also be used to record informal names such as a nickname.)
2	ANON	Anonymous	Anonymous assigned name (used to protect a person's identity for privacy reasons).
2	A	Business Name	A name used in a professional or business context. EXAMPLES: Continuing to use a maiden name in a professional context or using a stage performing name (some of these names are also pseudonyms).
2	R	Religious	A name assumed as part of a religious vocation, e.g. Sister Mary Francis, Brother John.
1	OLD	No longer in use	This name is no longer in use. NOTE Names can also carry valid time ranges. This code is used to cover the situations where it is known that the name is no longer valid, but no particular time range for its use is known.
2	DN	Do not use	This name should no longer be used when interacting with the person (i.e. in addition to no longer being used, the name should not even be mentioned when interacting with the person). NOTE Applications are not required to compare names labelled "Do not use" and other names in order to eliminate name parts that are common between the other name and a name labelled “Do not use”.
1	M	Maiden name	A name used prior to marriage. Marriage naming customs vary greatly around the world. This name use is for use by applications that collect and store “maiden” names. Though the concept of maiden name is often gender specific, the use of this term is not gender specific. The use of this term does not imply any particular history for a person’s name, nor should the maiden name be determined algorithmically.
1	SRCH	Search type uses	A name intended for use in searching or matching.
2	PHON	Phonetic	The name as understood by the data enterer, i.e. a close approximation of a phonetic spelling of the name, not based on a phonetic algorithm.

ISO/IEC 11404 syntax for the entityNameUse attribute

```
 type EntityNameUse = enumeration (C, OR, T, I, P, A, R, OLD, DN, M,
SRCH, PHON, ABC, SYL, IDE)
```

The use and qualifier codes are both used as sets, where more than one of each type may be used. This allows syntactically well-formed but semantically absurd constructions. The following rules apply:

— a single entity name may not have more than one NameRepresentationUse code;

— T, ABC, SYL and IDE should be accompanied by some other name use code;

— an organization entity name part qualifier code of "LS" may not be combined with any other qualifiers except PFX or SFX;

— the qualifiers BR and AD (or SP) are mutually incompatible.

7.7.6.4 Equality

Two name values are considered equal if their canonical forms contain the same name parts in the same order. Use code and valid time are excluded from the equality test.

7.7.6.5 Invariants

— either the EN is nullFlavored or it has at least one part.

OCL for Invariants:

```
inv "null or parts": isNull xor part-> notEmpty
```

7.7.6.6 Operations

7.7.6.6.1 canonical():EN: The Entity Name with a standard ordering imposed on the parts.

The canonical form is primarily defined for the purposes of defining equality, and may differ from the socially accepted order for the name parts in various cultures around the world.

The canonical form contains all the part types except for the delimiters, in the following order:

a) prefixes with qualifier title;

b) given names, with any prefixes and/or suffixes associated with the given names;

c) family names, with any prefixes and/or suffixes associated with the family names;

d) suffixes with qualifier title.

Each list of part types shall be in the order of the original name.

7.7.6.7 ISO/TS 22220 comments

The various name groups defined in ISO/TS 22220 map directly to the ENXP types. The conditional use and name usage components map to the use attribute. The context of use of the EN may need to allow for multiple EN values (as some kind of collection) to support all the functionality described in ISO/TS 22220.

7.7.6.8 Examples

7.7.6.8.1 Simple example

```
<example xsi:type="EN" >
   <part type="GIV" value="Adam"/>
   <part type="GIV" value="A."/>
   <part type="FAM" value="Everyman"/>
</example>
```

A very simple encoding of "Adam A. Everyman".

7.7.6.8.2 Complex germanic example

```
<example xsi:type="EN.PN">
   <part type="GIV" qualifier="AC" value="Dr. phil."/>
   <part type="GIV" value="Regina"/>
   <part type="GIV" value="Johanna"/>
   <part type="GIV" value="Maria"/>
   <part type="TITLE" qualifier="PFX NB" value="Gräfin"/>
   <part type="FAM" qualifier="BR" value="Hochheim"/>
   <part type="DEL" value="-"/>
   <part type="FAM" qualifier="SP" value="Weilenfels"/>
   <part type="TITLE" qualifier="SFX PR" value="NCFSA" />
</example>
```

Dr.phil. Regina Johanna Maria Gräfin Hochheim-Weilenfels, NCFSA. This example shows extensive use of multiple given names, prefixes, suffixes, for academic degrees, nobility titles and professional designations.

7.7.6.8.3 Organization name

```
<example xsi:type="EN.TN">
   <part value="Health Level Seven, Inc"/>
</example>
```

An organization name, "Health Level Seven, Inc." in simple string form: (Trivial Name – EN.TN).

```
<example xsi:type="EN.ON">
   <part value="Health Level Seven, "/>
   <part type="TITLE" qualifier="SFX LS" value="Inc."/>
</example>
```

As a fully parsed name.

7.7.6.8.4 Japanese example

```
<example xsi:type="EN" use="IDE">
   <part type="FAM" value="木村"/>
   <part type="GIV" value="通男"/>
</example>
<example xsi:type="EN" use="SYL">
   <part type="FAM" value="きむら"/>
   <part type="GIV" value="みちお"/>
</example>
<example xsi:type="EN" use="ABC">
   <part type="FAM" value="KIMURA"/>
   <part type="GIV" value="MICHIO"/>
</example>
```

A Japanese name in the three forms: ideographic (Kanji), syllabic (Hiragana) and alphabetic (Romaji).

7.7.6.8.5 Russian example

```
<example xsi:type="EN">
   <part type="FAM" value="ЕМЕЛИН"/>
   <part type="GIV" value="ИВАН"/>
   <part type="GIV" value="ВЛАДИМИРОВИЧ"/>
</example>
<example xsi:type="EN">
   <part type="FAM" value="EMELIN"/>
   <part type="GIV" value="IVAN"/>
</example>
```

A Russian name in Cyrillic with a Latin alphabet transliteration. In Russian usage, these names are known as the domestic and foreign names respectively. Systems should determine the appropriate form for a particular use based on the character set of the name parts.

7.7.6.8.6 Scandinavian examples

```
<example xsi:type="EN" use="OR">
   <part type="GIV" value="Jan"/>
   <part type="GIV" value="Erik"/>
   <part type="FAM" qualifier="MID" value=" Östlund"/>
   <part type="FAM" value="Erikson"/>
</example>
<example xsi:type="EN">
   <part type="GIV" value="Jan"/>
   <part type="FAM" value="Erikson"/>
</example>
```

Erikson is the family name. Jan Erik are the given names, and Östlund, the family name of the mother, which is taken as a Mellannamn.

```
<example xsi:type="EN" use="T">
  <!-- Use could be OR+OLD, depends how record keeping is done -->
   <part type="GIV" value="Margrete Jente"/>
   <part type="FAM" value="Hansen"/>
</example>
```

Jan Erikson has a daughter, Karin, with his wife Margrete Hansen. The first communications of the newborn name is “Margrete Jente” (Margrete's Girl) and the mother's family name, not the given name (Karin). The father's family name is not used at all. This is a known temporary name assigned directly after the birth of the child.

```
<example xsi:type="EN" use="OR C">
   <part type="GIV" value="Karin"/>
   <part type="FAM" qualifier="MID" value="Hansen"/>
   <part type="FAM" value="Erikson"/>
</example>
```

The baby's name is subsequently changed to the fathers' family name, and to use the mother's name as mellomnamn.

```
<example xsi:type="EN" use="OR">
   <part type="GIV" value="Karin"/>
   <part type="FAM" qualifier="MID" value="Erikson"/>
   <part type="FAM" qualifier="SP" value="Berg"/>
</example>
<example xsi:type="EN" use="C">
   <part type="GIV" value="Karin"/>
   <part type="FAM" value="Berg"/>
</example>
```

Karin gets married to Per Berg and decides to adopt Berg as her family name, and also decides to use Erikson as the mellom navn.

NOTE Karin could have chosen to use another mellom navn, e.g. the family name of her mother, her father or other family names as specified by naming laws of the country in question.

7.7.6.8.7 Nickname/informal name examples

```
<example xsi:type="EN">
   <part type="GIV" value="Peter"/>
   <part type="GIV" qualifier="CL" value="James"/>
   <part type="FAM" value="Chalmers"/>
</example>
```

The full name is Peter James Chalmers. The person prefers to be called by James (not "Jim" – no, do not call him that).

```
<example xsi:type="EN" use="OR">
   <part type="GIV" value="David"/>
   <part type="GIV" value="Woodford"/>
   <part type="FAM" value="Smith"/>
</example>
<example xsi:type="EN" use="C">
   <part type="GIV" value="Woody"/>
   <part type="FAM" value="Smith"/>
</example>
```

The person's proper name is David Woodford Smith, but he prefers to be called "Woody".

```
<example xsi:type="EN" use="OR">
   <part type="GIV" value="Uy"/>
   <part type="GIV" value="Dung"/>
   <part type="FAM" value="Nguyen"/>
</example>
<example xsi:type="EN" use="C">
   <part type="GIV" value="Dennis"/>
   <part type="FAM" value="Nguyen"/>
</example>
```

The person was born as "Uy Dung Nguyen", but when he migrated to a western nation, he chose to use Dennis as his normal "westernized" name. This is common practice among immigrants.

```
<example xsi:type="EN" use="OR C">
   <part type="GIV" value="Grahame"/>
   <part type="GIV" value="David"/>
   <part type="FAM" value="Grieve"/>
</example>
<example xsi:type="EN" use="P">
   <part type="GIV" value="Junior"/>
</example>
```

The person was born as "Grahame Grieve" and uses this name in normal use. However, he has sometimes been called "Junior" as well.

7.7.6.8.8 Title example

```
<example xsi:type="EN" use="OR C">
   <part type="TITLE" value="Dr"/>
   <part type="GIV" value="John"/>
   <part type="GIV" value="Paul"/>
   <part type="FAM" value="Jones"/>
   <part type="TITLE" qualifier="SFX" value="III"/>
   <part type="DEL" value=", "/>
   <part type="TITLE" qualifier="AC" value="PhD"/>
</example>
```

Dr John Paul Jones III, PhD. This name is given the use code "OR" for Official Registry Name, but contains titles. For the purposes of this International Standard, titles and delimiters are not part of the official registry name; they can be present and there is no assertion that they are actually registered.

NOTE "Dr" is an abbreviation, not an initial. Initials can contain more than one letter for linguistic reasons, but they are not the same as an abbreviation. Titles are often abbreviated.

7.7.6.8.9 Complex examples

```
<example xsi:type="EN" use="OR C">
   <part type="GIV" value="Mary Jane"/>
   <part type="FAM" value="Contrata"/>
</example>
```

Mary Jane are two specifically space separated and ordered portions of the first name, rather than "Jane" being a middle name.

NOTE Generating initials algorithmically from this name, they would usually be MC not MJC.

```
<example xsi:type="EN" use="OR C">
   <part type="GIV" value="Karen"/>
   <part type="FAM" value="Van"/>
   <part type="FAM" value="Hentenryck"/>
</example>
```

Karen Van Hentenryck is of Dutch origin, and the "Van"' is a voorvoegsel.

```
<example xsi:type="EN" use="OR C">
   <part type="GIV" value="Selby"/>
   <part type="FAM" qualifier="SP" value="Butt"/>
   <part type="FAM" value="Beeler"/>
</example>
<example xsi:type="EN" use="OR OLD">
   <part type="GIV" value="Mary"/>
   <part type="FAM" qualifier="CL" value="Selby"/>
   <part type="FAM" value="Butt"/>
</example>
```

Born Mary "Selby" Butt, but changed her name to Selby Butt Beeler upon marriage, and this is the name on her passport.

```
<example xsi:type="EN" use="OR A OLD">
   <part type="GIV" value="Jacqueline "/>
   <part type="GIV" value="Janette"/>
   <part type="GIV" value="Patricia"/>
   <part type="FAM" value="Campbell"/>
</example>
<example xsi:type="EN" use="P OLD">
   <part type="GIV" value="Ruth"/>
   <part type="FAM" value="Brinkman"/>
</example>
<example xsi:type="EN" use="P OLD">
   <part type="GIV" value="Ruth"/>
   <part type="FAM" qualifier="SP" value="Grieve "/>
</example>
<example xsi:type="EN" use="OR">
   <part type="GIV" value="Jacqueline"/>
   <part type="GIV" value="Janette"/>
   <part type="GIV" value="Patricia"/>
   <part type="FAM" value="Grieve"/>
</example>
<example xsi:type="EN" use="C">
   <part type="GIV" value="Jacque"/>
   <part type="FAM" value="Grieve"/>
</example>
<example xsi:type="EN" use="C OLD">
   <part type="GIV" value="Jacque Ruth"/>
   <part type="FAM" value="Grieve"/>
</example>
<example xsi:type="EN" use="M">
   <part type="FAM" value="Brinkman "/>
</example>
```

This is a particularly complex example, but anonymized from a real person. She was born as “Jacqueline Janette Patricia Campbell”, but grew up under the foster name “Ruth Brinkman”. Upon marriage, she was known as “Ruth Grieve” but her legal name was “Jacqueline Janette Patricia Grieve”. Later, changed her name to “Jacque-Ruth” and then just “Jacque”. Out of all this, reports her maiden name as “Brinkman”.

```
<example xsi:type="EN" use="OR OLD">
   <part type="GIV" qualifier="BR" value="Del-Roy"/>
   <part type="FAM" qualifier="BR" value="Burgess"/>
</example>
<example xsi:type="EN" use="P">
   <part type="GIV" value="Yor-Led"/>
   <part type="FAM" value="Ssegrub"/>
</example>
<example xsi:type="EN" use="OR ABC OLD">
   <part type="GIV" qualifier="AD PFX" value="Abdul"/>
   <part type="DEL" value="-"/>
   <part type="GIV" qualifier="AD SFX" value="Malik"/>
   <part type="FAM" qualifier="AD" value="Shakir"/>
</example>
<example xsi:type="EN" use="OR ABC C">
   <part type="GIV" qualifier="AD" value="AbdulMalik"/>
   <part type="FAM" qualifier="AD" value="Shakir"/>
   <part type="TITLE" value="Sr"/>
</example>
<example xsi:type="EN" use="P DN">
   <part type="GIV" qualifier="AD" value="Abdul"/>
</example>
<example xsi:type="EN" use="P">
   <part value="AMS"/>
</example>
```

Another complicated example taken from a real person, who says: "I was born Del-Roy Burgess, and my nickname was Yor-Led Ssegrub. I changed my name to Abdul-Malik Shakir when adopting Islam as my religion. The spelling is a phonetic spelling of an Arabic name using the Latin alphabet. If Abdul-Malik is a bit of a mouthful, do not call me Abdul, please call me AMS instead. I recently began spelling my first name in camel case and dropped the "-" delimiter (i.e. AbdulMalik not Abdul-Malik). I also recently started using the suffix Sr. to differentiate my identity from my son's because he has the same name. The suffix is not commonly used except on a few official registries such as passport, driving licence and other areas where identity disambiguation is important.".

7.7.7 EN.TN (trivial name)

7.7.7.1 Description

A flavor that constrains EN.

A restriction of EN that is effectively a simple string used for a simple name for things and places. Trivial names are typically used for places and things, such as Lake Erie or Washington-Reagan National Airport.

7.7.7.2 Invariants

— if the EN.TN is not null, there can only be one part, and it can have no type or qualifier.

OCL for Invariants:

```
inv "only one part with no type": isNotNull implies
      (part->size = 1 and part->first.type.oclIsUndefined and
         part->first.qualifier->isEmpty)
```

7.7.8 EN.PN (person name)

7.7.8.1 Description

A flavor that constrains EN.

A restriction of EN used when the named entity is a person. A sequence of name parts, such as given name or family name, prefix, suffix.

A name part is a restriction on entity name part that only allows those entity name parts qualifiers applicable to person names.

NOTE Since the structure of entity name is mostly determined by the requirements of person name, the restriction is very minor.

7.7.8.2 Invariants

— none of the parts of a persons name can be qualified by the status LS.

OCL for Invariants:

```
inv "no parts are qualified by LS": part->forAll(p | not
    p.qualifier->includes(EntityNamePartQualifier.LS))
```

7.7.9 EN.ON (Organization Name)

7.7.9.1 Description

A flavor that constrains EN.

7.7.9.2 Invariants

— none of the parts of a organization name can be FAM or GIV;

— the following qualifiers shall not be used in the name of an organization: I, P, ANON, A, R, DN and M.

OCL for Invariants:

```
inv "no parts are person types": part->forAll(p |
    not (p.type = EntityNamePartType.FAM or
      p.type = EntityNamePartType.GIV))
inv "no use codes are person codes":
  use->forAll(u | not
    (u = EntityNameUse.I
    or u = EntityNameUse.P
    or u = EntityNameUse.ANON
    or u = EntityNameUse.A
    or u = EntityNameUse.R
    or u = EntityNameUse.M)).M))
```

7.8 Quantity datatypes

7.8.1 Overview

These datatypes provide support for quantitative values (see Figure 7).

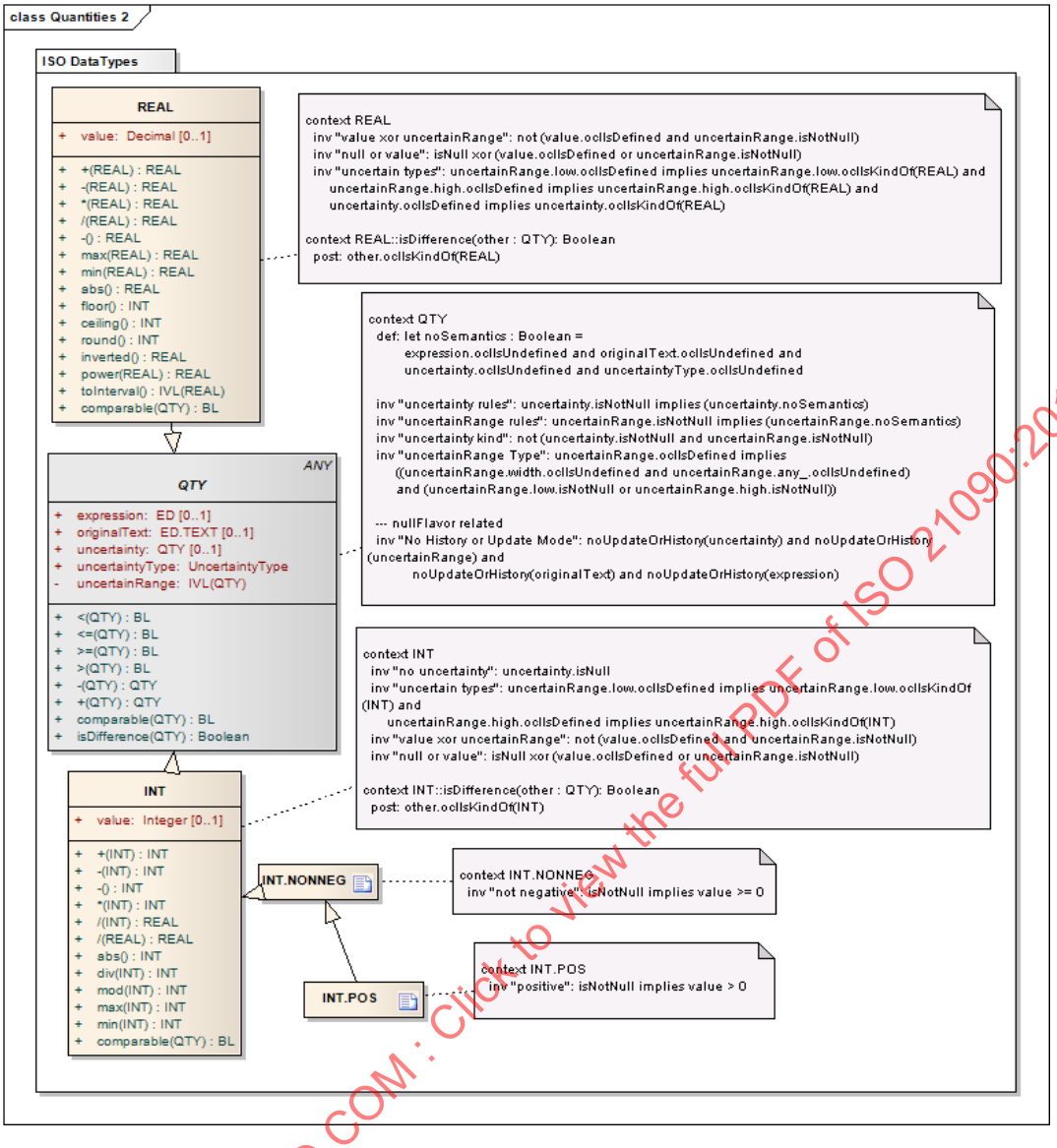

Figure 7 (*continued*)

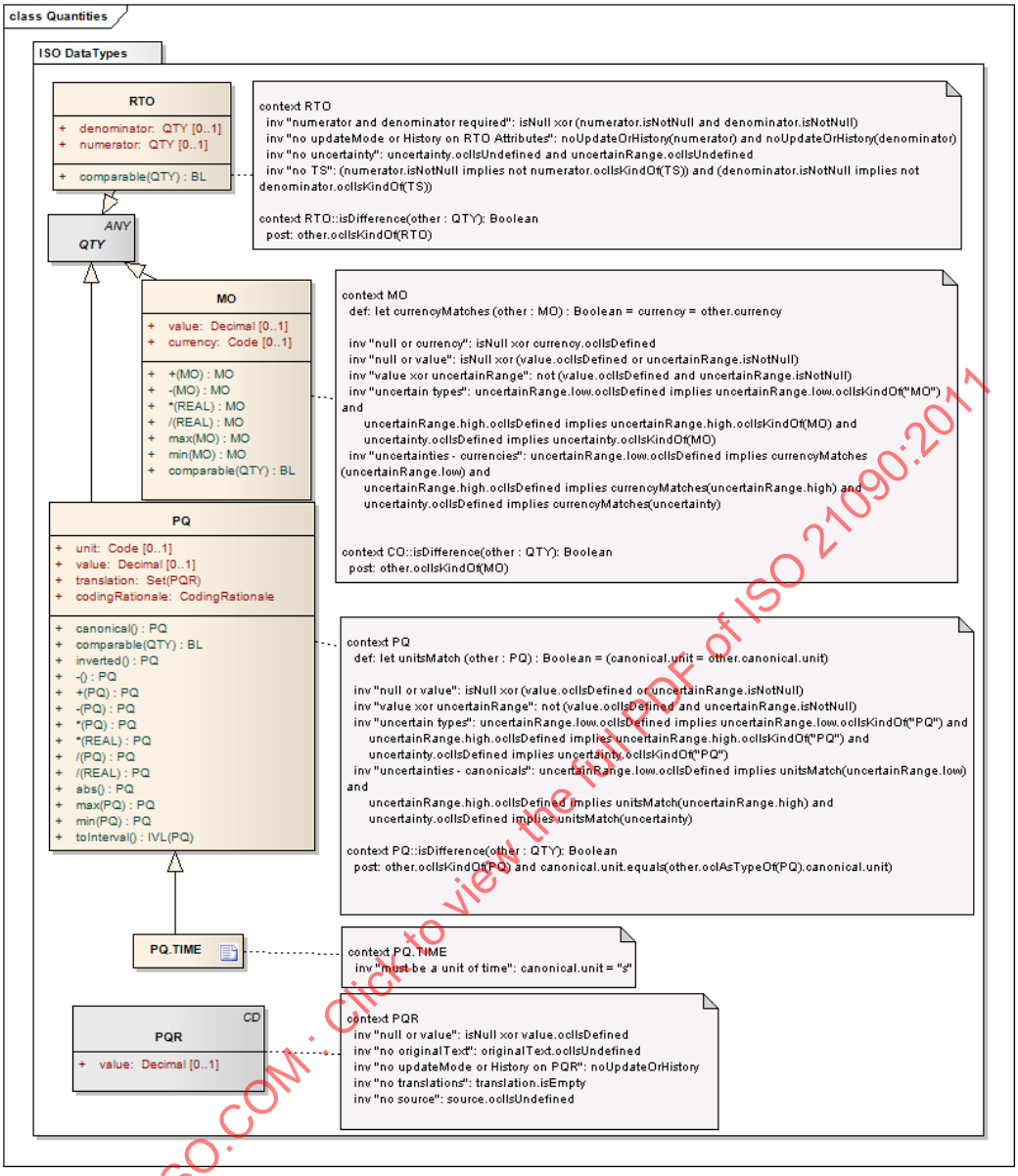

Figure 7 (*continued*)

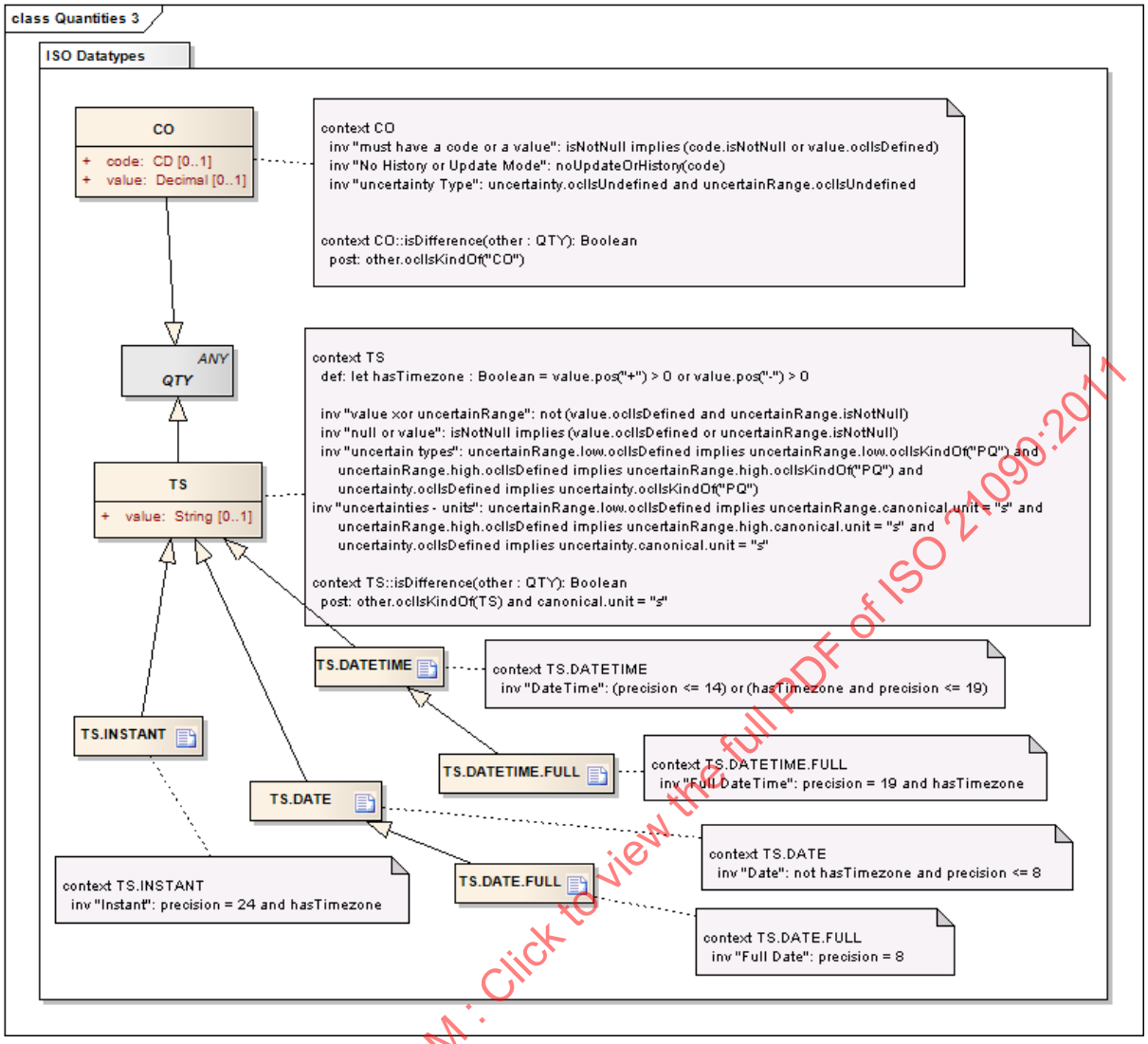

Figure 7 — Quantity datatypes

7.8.2 QTY (quantity)

7.8.2.1 Description

Specializes ANY.

The quantity datatype is an abstract generalization for all datatypes whose domain values have an order relation (less-or-equal) and where difference is defined in all of the datatype's totally ordered value subsets.

The quantity type abstraction is needed in defining certain other types, such as the interval and probability distributions.

7.8.2.2 ISO/IEC 11404 syntax

```
type QTY = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  expression : ED,
  originalText : ED.TEXT,
  uncertainty : QTY,
  uncertaintyType : UncertaintyType
  uncertainRange : IVL(QTY)
)
```

QTY defines three facilities that all quantities may carry: an expression that may be used to derive the actual value, an originalText that carries the original form in which the quantity was represented and the uncertainty associated with the value. There are two different ways to represent the uncertainty: one is a statistical form – usually suited for measured values, and a range form, which is usually associated with instructions (i.e. take 4 to 6 tablets).

The presence of these attributes may considerably complicate proper understanding of the value. For this reason, their use should be strictly controlled in all contexts of use. Conformance statements shall make clear exactly how and when these attributes are used if quantities are used by the associated Information processing entities.

7.8.2.3 Attributes

7.8.2.3.1 expression : ED: An expression that can be used to derive the actual value of the quantitive given information taken from the context of use.

For example expression can be used for expressing dosage instructions that depend on patient's body weight.

If no proper value is provided for the QTY, then the value shall have a nullFlavor, whether or not an expression is provided. If no proper value is provided, and an expression is provided, the appropriate NullFlavor is usually DER. No nullFlavor is required if both a proper value and an expression is provided; in such cases, it is up to the processing to determine when the expression should be evaluated.

The language of the expression is inferred from the mediatype. If multiple translations are provided in the expression, the evaluator is free to choose whichever language is preferred; all translations shall specify the same outcome.

The language defines the forms that the expression property can take, how the information available in the context of the expression is made available within the features of the language, and how the language declares the new form of the value. Languages may only be used if this information has been appropriately defined for the context in which the QTY is used.

Information processing entities are not required to implement any languages in order to claim direct or indirect conformance with this International Standard, but should declare what languages are supported in their conformance statements.

Language	Mediatype
OCL	text/plain+ocl
Factor	application/hl7-factor+xml
MathML	application/mathml+xml
NOTE Factor is an HL7 specific language documented in the Abstract Data Types Specification.	

7.8.2.3.2 originalText : ED.TEXT: The text representation from which the QTY was encoded, if such a representation is the source of the QTY.

Original text can be used in a structured user interface to capture what the user saw as a representation of the quantity on the data input screen, or in a situation where the user dictates or directly enters text, it is the text entered or uttered by the user.

It is valid to use a QTY derived datatype to store only the text that the user entered or uttered. In this situation, original text will exist without a valid value. In a situation where the value is determined sometime after the text was entered, originalText is the text or phrase used as the basis for determining the value. The originalText is not a substitute for a valid value. If the actual value of the QTY is not valid, then the QTY shall be nullFlavored, irrespective of whether originalText has a value or not.

The original text shall be an excerpt of the relevant information in the original sources, rather than a pointer or exact reproduction. Thus the original text shall be represented in plain text form. In specific circumstances, when clearly described in the context of use, the originalText may be a reference to some other text artefact for which the resolution scope is clearly described.

NOTE The details of the link in the originalText.reference between different artifacts of medical information (e.g. document and coded result) is outside the scope of this International Standard and can be further proscribed in specifications that use this International Standard.

7.8.2.3.3 uncertainty : QTY: The uncertainty of the quantity using a distribution function and its parameters. It is the primary measure of variance/uncertainty of the value (the square root of the sum of the squares of the differences between all data points and the mean). The actual type of uncertainty depends on the type of the QTY and is fixed for each type.

There are two different kinds of uncertainty representation. This kind of uncertainty, along with uncertaintyType, represents statistical uncertainty. uncertainRange specifies a different kind of uncertainty with no implied statistical distribution.

This form of uncertainty shall only be applied to value domains that have a continuous distribution (REAL, PQ, MO and TS). Uncertainty may be applied separately to the numerator and denominator of an RTO.

Uncertainty shall not have an expression. Uncertainty shall not have uncertainty of its own. Uncertainty shall not have originalText – any uncertainty associated with the QTY should be conveyed as part of the originalText of the QTY itself.

Uncertainty does not have its own originalText because it is expected that the uncertainty of the quantity should be expressed in the originalText of the quantity itself.

7.8.2.3.4 uncertaintyType : UncertaintyType: A code specifying the type of probability distribution in uncertainty.

There are two different kinds of uncertainty representation. This kind of uncertainty, along with uncertainty, represents statistical uncertainty. uncertainRange specifies a different kind of uncertainty with no implied statistical distribution.

The null value (unknown) for the type code indicates that the probability distribution type is unknown. In that case, uncertainty has the meaning of an informal guess if it is populated.

If populated, the value of this attribute shall be taken from the HL7 DistributionType code system. The current values are:

UncertaintyType Enumeration. OID: 2.16.840.1.113883.5.1020			
1	U	Uniform	The uniform distribution assigns a constant probability over the entire interval of possible outcomes, while all outcomes outside this interval are assumed to have zero probability. The width of this interval is 2 σ √3. Thus, the uniform distribution assigns the probability densities $f(x) = (2\ \sigma\ \sqrt{3})^{-1}$ to values $\mu - \sigma\ \sqrt{3} \geqslant x \leqslant \mu + \sigma\ \sqrt{3}$ and $f(x) = 0$ otherwise.
1	N	Normal (gaussian)	This is the well-known bell-shaped normal distribution. Because of the central limit theorem, the normal distribution is the distribution of choice for an unbounded random variable that is an outcome of a combination of many stochastic processes. Even for values bounded on a single side (i.e. greater than 0) the normal distribution may be accurate enough if the mean is "far away" from the bound of the scale measured in terms of standard deviations.
1	LN	Log-normal	The logarithmic normal distribution is used to transform skewed random variable X into a normally distributed random variable $U = \log X$. The log-normal distribution can be specified with the properties mean, μ, and standard deviation, σ. However, mean, μ, and standard deviation, σ, are the parameters of the raw value distribution, not the transformed parameters of the log-normal distribution that are conventionally referred to by the same letters. Those log-normal parameters, μ_{log} and σ_{log}, relate to the mean, μ, and standard deviation, σ, of the data value through $_{log}^2 = \log\ (\sigma^2/\mu^2 + 1)$ and $\mu log = \log\ \mu - \sigma log^2/2$.
1	G	? (Gamma)	The gamma-distribution used for data that is skewed and bounded to the right, i.e. where the maximum of the distribution curve is located near the origin. The γ-distribution has two parameters α and β. The relationship to mean μ and variance σ^2 is $\mu = \alpha\ \beta$ and $\sigma^2 = \alpha\ \beta^2$.
1	E	Exponential	Used for data that describes extinction. The exponential distribution is a special form of γ-distribution where $\alpha = 1$, hence, the relationship to mean, μ, and variance, σ^2, are $\mu = \beta$ and $\sigma^2 = \beta^2$.
1	X2	?	Used to describe the sum of squares of random variables that occurs when a variance is estimated (rather than presumed) from the sample. The only parameter of the χ^2-distribution is υ, the so called *number of degrees of freedom* (which is the number of independent parts in the sum). The χ^2-distribution is a special type of γ-distribution with parameter $\alpha = \upsilon/2$ and $\beta = 2$. Hence, $\mu = \upsilon$ and $\sigma^2 = 2\ \upsilon$.
1	T	t (Student)	Used to describe the quotient of a normal random variable and the square root of an χ^2 random variable. The t-distribution has one parameter, υ, the number of degrees of freedom. The relationship to mean, μ, and variance, σ^2, are: $\mu = 0$ and $\sigma^2 = \upsilon/(\upsilon - 2)$.
1	F	f	Used to describe the quotient of two χ^2 random variables. The F-distribution has two parameters υ_1 and υ_2, which are the numbers of degrees of freedom of the numerator and denominator variable respectively. The relationship to mean, μ, and variance, σ^2, are: $\mu = \upsilon_2/(\upsilon_2 - 2)$ and $\sigma^2 = (2\ \upsilon_2^{\ 2}\ (\upsilon_2 + \upsilon_1 - 2))/[\upsilon_1\ (\upsilon_2 - 2)^2\ (\upsilon_2 - 4)]$.
1	B	? (Beta)	The beta-distribution is used for data that are bounded on both sides and may or may not be skewed (e.g. occurs when probabilities are estimated.) Two parameters, α and β, are available to adjust the curve. The mean, μ, and variance, σ^2, relate as follows: $\mu = \alpha/(\alpha + \beta)$ and $(\sigma^2 = \alpha\ \beta/[(\alpha + \beta)^2\ (\alpha + \beta + 1)]$.

Many distribution types are defined in terms of special parameters (e.g. the parameters α and β for the γ-distribution, number of degrees of freedom for the t-distribution). For all distribution types, however, the mean and standard deviation are defined.

If no value (null) is provided for distributionType, then the mean is estimated without any closer consideration of its probability distribution. In this case, the meaning of the standard deviation is not crisply defined. However, interpretation should be along the lines of the normal distribution, e.g. the interval covered by the mean ± 1 standard deviation should be at the level of about two thirds confidence.

The three distribution-types unknown (null), uniform and normal shall be supported by every system that claims to support uncertainty. All other distribution types are optional. When a system interpreting a uncertainty representation encounters a distribution type that it does not recognise, it maps this type to the unknown (null) distribution-type.

ISO/IEC 11404 syntax for the distributionType attribute

```
type UncertaintyType = enumeration (U, N, LN, G, E, X2, T, F, B)
```

7.8.2.3.5 uncertainRange : IVL(QTY): Indicates that the value comes from a range of possible values.

uncertainRange is used where the actual value is unknown, but it is known that the value comes from a known range of possible values. uncertainRange differs from uncertainty in that uncertainty is used to report a particular value along with an associated distribution of uncertainty for the value, or to report the summary distribution of a set of data, whereas uncertainRange indicates that there is a single value that, although unknown, comes from a particular range of values. No inference regarding distribution of values can be taken. uncertainRange is often associated with an instruction to perform a particular operation at some point within a given time interval.

If an uncertainRange is provided, a low or high shall be provided. The IVL any and width attributes cannot be used. If an uncertainRange is provided, no value can be provided.

7.8.2.4 Equality

Equality is not defined for the QTY datatype as it is an abstract type.The QTY attributes (expression, originalText, uncertainty and uncertaintyType) never participate in the determination of equality of specializations of QTY.

7.8.2.5 Invariants

- uncertainty has no expression, uncertainty or originalText;
- uncertainRange has no expression, uncertainty or originalText;
- cannot have both uncertainty and uncertainRange;
- cannot have width or any on uncertainRange.

OCL for Invariants:

```
def: let noSemantics : Boolean = expression.oclIsUndefined and
   originalText.oclIsUndefined and uncertainty.oclIsUndefined and
   uncertaintyType.oclIsUndefined

inv "uncertainty rules": uncertainty.isNotNull implies
   (uncertainty.noSemantics)
inv "uncertainRange rules": uncertainRange.isNotNull implies
   (uncertainRange.noSemantics)
inv "uncertainty kind": not (uncertainty.isNotNull and
```

```
     uncertainRange.isNotNull)
inv "uncertainRange Type": uncertainRange.oclIsDefined implies
    ((uncertainRange.width.oclIsUndefined and
      uncertainRange.any.oclIsUndefined)
    and (uncertainRange.low.isNotNull or
       uncertainRange.high.isNotNull))

inv "No History or Update Mode": noUpdateOrHistory(uncertainty) and
         noUpdateOrHistory(uncertainRange) and
         noUpdateOrHistory(originalText) and
         noUpdateOrHistory(expression))
```

7.8.2.6 Operations

7.8.2.6.1 lessthan[<](other : QTY):BL: True if the value of this is less than the value of other. For uncertain values, this may not be known (result = NullFlavor.UNK).

7.8.2.6.2 lessOrEqual[<=](other : QTY):BL: True if the value of this is less than or equal to the value of other. For uncertain values, this may not be known (result = NullFlavor.UNK).

7.8.2.6.3 greaterOrEqual[>=](other : QTY):BL: True if the value of this is greater than or equal to the value of other. For uncertain values, this may not be known (result = NullFlavor.UNK).

7.8.2.6.4 greaterThan[>](other : QTY):BL: True if the value of this is greater than the value of other. For uncertain values, this may not be known (result = NullFlavor.UNK).

7.8.2.6.5 plus[+](other : QTY):QTY: The result of addition of this and other. Other must be of the right type of value (same type as this, except for TS, where the value must be a PQ with units of time, or for PQ, where the value of other must have compatible units), else the result is NullFlavor NI. Uncertainties should be carried through the operation. If the values have mixed uncertainties, the result may be unknown (result = NullFlavor.UNK).

7.8.2.6.6 minus[-](other : QTY):QTY: The result of subtraction of other from this. Other must be of the type of value (same type as this, except for TS, where the value must be a TS or a PQ with units of time, or for PQ, where the value of other must have compatible units), else the result is NullFlavor NI. Uncertainties should be carried through the operation. If the values have mixed uncertainties, the result may be unknown (result = NullFlavor.UNK).

7.8.2.6.7 comparable(other : QTY) : Boolean: whether this and other can be compared using equality.

NOTE Generally, this is true if both this and other are the same type, unless noted otherwise for specializations for QTY.

7.8.2.6.8 isDifference(other : QTY):BL True if other is an instance that expresses the difference between two instances of this type.

NOTE Usually this is true if other is the same as the type, except for TS, where the difference is expressed as a PQ with a unit that is a kind of time, and for PQ, where the units must be compatible.

7.8.3 INT (integer)

7.8.3.1 Description

Specializes QTY.

Integer numbers (–1,0,1,2, 100, 3398129, etc.) are precise numbers that are results of counting and enumerating. Integer numbers are discrete, the set of integers is infinite but countable. No arbitrary limit is imposed on the range of integer numbers.

7.8.3.2 ISO/IEC 11404 syntax

```
type INT = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  expression : ED,
  originalText : ED.TEXT,
  uncertainty : QTY,
  uncertaintyType : UncertaintyType,
  uncertainRange : IVL(QTY)
  value : integer
)
```

7.8.3.3 Attributes

value : Integer: The value of the INT. This International Standard imposes no limitations on the size of integer, but most implementations map this to a 32 or 64 bit integer.

This is an example of the primitive type wrapping pattern. See 6.3 for more details.

7.8.3.4 Equality

Two nonNull INT are equal if they are not nullFlavored and have the same value, or their uncertainRanges not null or nullFlavored and equal.

7.8.3.5 Invariants

- a value or an uncertain range shall be provided if not nullFlavored;
- cannot provide both a value and an uncertain range;
- uncertainRange shall be an IVL(INT);
- uncertainty shall not be populated.

OCL for Invariants:

```
inv "no uncertainty": uncertainty.isNull
inv "uncertain types": uncertainRange.low.oclIsDefined
    implies uncertainRange.low.oclIsKindOf(INT) and
    uncertainRange.high.oclIsDefined implies
    uncertainRange.high.oclIsKindOf(INT)
inv "value xor uncertainRange": not (value.oclIsDefined and
    uncertainRange.isNotNull)
inv "null or value": isNull xor (value.oclIsDefined or
    uncertainRange.isNotNull)
inv "null or value": isNull xor value.oclIsDefined
```

7.8.3.6 Operations

7.8.3.6.1 negated[–] : INT: The negative value of this.

7.8.3.6.2 plus[+] (other : INT) : INT: the value of the addition of this and other.

7.8.3.6.3 minus[–] (other : INT) : INT: The value of the subtraction of other from this.

7.8.3.6.4 times[*] (other : INT) : INT: The value of the multiplication of this and other.

7.8.3.6.5 dividedBy[/] (other : INT) : REAL: The value of this divided by other. If other is 0, then the result is NullFlavor NI.

7.8.3.6.6 dividedBy[/] (other : REAL) : REAL: The value of this divided by other. If other is 0, then the result is NullFlavor NI.

7.8.3.6.7 abs() : INT: The absolute value of this.

7.8.3.6.8 div(other : INT) : INT: The number of times that other fits completely within this.

7.8.3.6.9 mod(other : INT) : INT: The result is this modulo other.

7.8.3.6.10 max(other : INT) : INT: The maximum of this and other.

7.8.3.6.11 min(other : INT) : INT: The minimum of this and other.

7.8.3.6.12 comparable(other : QTY):BL: Integer values may always be compared.

7.8.3.7 Examples

7.8.3.7.1 Plain value

```
<example xsi:type="INT" value="23"/>
```

The integer 23.

7.8.3.7.2 Unknown value

```
<example xsi:type="INT" nullFlavor="NASK"/>
```

The patient was not asked for this value. For instance, the patient has never been pregnant, so that patient was not asked how many children she has.

7.8.4 INT.NONNEG

7.8.4.1 Description

A flavor that constrains INT.

NT.NONNEG constrains INT so that it has a value of 0 or greater.

7.8.4.2 Invariants

— the value shall be zero or greater if not nullFlavored, with no uncertainty.

OCL for Invariants:

```
inv "not negative": isNotNull implies value >= 0
```

7.8.5 INT.POS

7.8.5.1 Description

A flavor that constrains INT.NONNEG.

INT.POS constrains INT.NONNEG so that it has a value greater than 0.

7.8.5.2 Invariants

— the value shall be greater than zero if not nullFlavored, with no uncertainty.

OCL for Invariants:

```
inv "positive": isNotNull implies value > 0
```

7.8.6 CO (coded ordinal)

7.8.6.1 Description

Specializes QTY.

Represents data where coded values are associated with a specific order.

CO may be used for things that model rankings and scores, e.g. likert scales, pain, Apgar values, where there is a) implied ordering, b) no implication that the distance between each value is constant, and c) the total number of values is finite. CO may also be used in the context of an ordered code system. In this case, it may not be appropriate or even possible to use the value attribute, but CO may still be used so that models that make use of such code systems may introduce model elements that involve statements about the order of the terms in a domain.

The relative order of values in a code system need not be independently obvious in the literal representation of the CO. It these circumstances, is expected that an application will look up the ordering of these values from some definition of the code system.

Some of the code systems directly assign numerical value to the concepts that are suitable for some mathemetical operations.

Though it would generally make sense, applications should not assume that the translations of the code, if provided, have the same ordering as the CO. Translations shall not be considered when the ordering of the code system is determined.

7.8.6.2 ISO/IEC 11404 syntax

```
type CO = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  expression : ED,
  originalText : ED.TEXT,
  uncertainty : QTY,
  uncertaintyType : UncertaintyType,
  uncertainRange : IVL(QTY)
  value : Decimal,
  code : CD
)
```

7.8.6.3 Attributes

7.8.6.3.1 value : Decimal: A numerical value associated with the coded ordinal value.

The value may be constrained to an integer in some contexts of use. If code is nonNull, value shall only be nonNull if the code system explicitly assigns a value to the concept.

7.8.6.3.2 code : CD: A code representing the definition of the ordinal item.

7.8.6.4 Equality

Two nonNull CO values are equal if their codes are equal.

NOTE 1 CO values that have value alone with no code are never equal, as it is not clear whether they are comparable ordinals.

NOTE 2 Since the determination of CO equality is based upon the code, CO values can be equal to CD values.

7.8.6.5 Invariants

— there shall be a code or a value if not nullFlavored;

— no uncertainty.

OCL for invariants:

```
inv "must have a code or a value": isNotNull implies
    (code.isNotNull or value.oclIsDefined)
inv "uncertainty Type": uncertainty.oclIsUndefined and
      uncertainRange.oclIsUndefined
inv "No History or Update Mode":noUpdateOrHistory(code)
```

7.8.6.6 Operations

7.8.6.6.1 max(other : CO) : CO: The maximum of this and other.

NOTE If the value attribute is not specified, the applicable terminology might need to be consulted to determine the order of the two values. If no order is defined, the result will be NullFlavor NI.

7.8.6.6.2 min(other : CO) : CO: The minimum of this and other.

NOTE If the value attribute is not specified, the applicable terminology might need to be consulted to determine the order of the two values. If no order is defined, the result will be NullFlavor NI.

7.8.6.6.3 comparable(other : QTY):BL: This is false unless this and other have the same codeSystem, and then only if the codeSystem defines an order amongst the codes.

7.8.6.6.4 plus[+](other : QTY):QTY: This operation returns null unless this.comparable(other) is true, and the codeSystem defines the meaning of addition for these codes.

7.8.6.6.5 minus[-](other : QTY):QTY: This operation returns null unless this.comparable(other) is true, and the codeSystem defines the meaning of subtraction for these codes.

7.8.6.7 Examples

```
<example xsi:type="CO" value="1">
  <code code="1" codeSystem="2.16.840.1.113883.2.6.15.1.1">
    <displayName value="Poor"/>
  </code>
</example>
```

In this case, the value "poor“ is assigned a numerical value of 1.

7.8.7 REAL (real)

7.8.7.1 Description

Specializes QTY.

Fractional numbers. Typically used whenever quantities are measured, estimated or computed from other real numbers. The typical representation is decimal, where the number of significant decimal digits is known as the precision.

7.8.7.2 ISO/IEC 11404 syntax

```
type REAL = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  expression : ED,
  originalText : ED.TEXT,
  uncertainty : QTY,
  uncertaintyType : UncertaintyType,
  uncertainRange : IVL(QTY)
  value : Decimal
)
```

7.8.7.3 Attributes

7.8.7.4 value : Decimal: The value of the REAL.

This is an example of the primitive type wrapping pattern. See 6.3 for more details.

7.8.7.5 Equality

Two nonNull REAL are equal if they are not nullFlavored and have the same value, or if their uncertainRanges are not null and not nullFlavored and equal.

7.8.7.6 Invariants

- a value or an uncertain range shall be provided if not nullFlavored;
- cannot provide both a value and an uncertain range;
- uncertainty types shall be a REAL.

OCL for Invariants:

```
inv "value xor uncertainRange": not
   (value.oclIsDefined and uncertainRange.isNotNull)
inv "null or value": isNull xor (value.oclIsDefined or
    uncertainRange.isNotNull)
inv "uncertain types": uncertainRange.low.oclIsDefined
    implies uncertainRange.low.oclIsKindOf("REAL") and
    uncertainRange.high.oclIsDefined implies
    uncertainRange.high.oclIsKindOf("REAL") and
    uncertainty.oclIsDefined implies
    uncertainty.oclIsKindOf("REAL")
```

7.8.7.7 Operations

7.8.7.7.1 plus[+] (other : REAL) : REAL: The value of the addition of this and other..

7.8.7.7.2 minus[–] (other : REAL) : REAL: The value of the subtraction of other from this.

7.8.7.7.3 times[*] (other : REAL) : REAL: The value of the multiplication of this and other.

7.8.7.7.4 negated[–] : REAL: The negative value of this.

7.8.7.7.5 dividedBy[/] (other : REAL) : REAL: The value of this divided by other. If other is 0, then the result is NullFlavor NI.

7.8.7.7.6 abs() : REAL: The absolute value of this.

7.8.7.7.7 floor() : INT: The largest integer which is less than or equal to this.

7.8.7.7.8 ceiling() : INT: The smallest integer which is greater than or equal to this.

7.8.7.7.9 round() : INT: The integer which is closest to this. When there are two such integers, the largest one.

7.8.7.7.10 inverted() : REAL: The value of 1 divided by self..

7.8.7.7.11 max(other : REAL) : REAL: The maximum of this and other.

7.8.7.7.12 min(other : REAL) : REAL: The minimum of this and other.

7.8.7.7.13 power(other : REAL) : REAL : this raised to the power of other.

7.8.7.7.14 toInterval() : IVL(REAL) : Converts this value to an interval that expresses the range covered by the precision.

7.8.7.7.15 comparable(other : QTY):BL: Real numbers may always be compared.

7.8.7.8 Examples

7.8.7.8.1 Precision

```
<example xsi:type="REAL" value="23.0005"/>
```

The floating value 23.0005.

```
<example xsi:type="REAL" value="23.00"/>
```

The floating value 23.00.

7.8.7.8.2 Uncertainty

```
<example xsi:type="REAL" value="23" uncertaintyType="N">
  <uncertainty xsi:type="REAL" value="0.87"/>
</example>
```

The floating value 23. The uncertainty is known to be a normal distribution with a standard deviation of 0.87. Uncertainty always needs an xsi:type.

7.8.8 RTO (ratio)

7.8.8.1 Description

Specializes QTY.

A quantity constructed as the quotient of a numerator quantity divided by a denominator quantity.

Common factors in the numerator and denominator are not automatically cancelled out.

The RTO datatype supports titers (e.g. "1:128") and other quantities produced by laboratories that truly represent ratios. Ratios are not simply "structured numerics", particularly blood pressure measurements (e.g. "120/60") are not ratios.

NOTE 1 Ratios are different from rational numbers, i.e. in ratios common factors in the numerator and denominator never cancel out. A ratio of two real or integer numbers is not automatically reduced to a real number. This datatype is not defined to generally represent rational numbers. It is used only if common factors in numerator and denominator are not supposed to cancel out. This is only rarely the case. For observation values, ratios occur almost exclusively with titers. In most other cases, REAL should be used instead of the RTO.

NOTE 2 Since many implementation technologies expect generics to be collections or only have one parameter, RTO is not implemented as a generic in this International Standard. Constraints, at the point where the RTO is used, define which form of QTY is used.

7.8.8.2 ISO/IEC 11404 syntax

```
type RTO = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  expression : ED,
  originalText : ED.TEXT,
  uncertainty : QTY,
  uncertaintyType : UncertaintyType,
  uncertainRange : IVL(QTY)
  numerator : QTY,
  denominator : QTY
)
```

7.8.8.3 Attributes

7.8.8.3.1 numerator : QTY: The quantity that is being divided in the ratio.

7.8.8.3.2 denominator : QTY: The quantity that divides the numerator in the ratio.

The denominator shall not be zero.

7.8.8.3.3 Equality

Two nonNull RTOs are equal if their numerator and denominator are equal.

7.8.8.4 Invariants

- if the RTO is not nullFlavored, both a numerator and a denominator are required;
- uncertainty shall not be populated;
- neither numerator nor denominator may be of type TS.

OCL for Invariants:

```
inv "numerator and denominator required": isNull xor
     (numerator.isNotNull and denominator.isNotNull)
inv "no updateMode or History on RTO Attributes":
      noUpdateOrHistory(numerator) and
       noUpdateOrHistory(denominator)
inv "no uncertainty": uncertainty.oclIsUndefined and
      uncertainRange.oclIsUndefined
inv "no TS": (numerator.isNotNull implies not
     numerator.oclIsKindOf("TS")) and (denominator.isNotNull
     implies not denominator.oclIsKindOf("TS"))
```

7.8.8.5 Operations

7.8.8.5.1 comparable(other : QTY):BL: This and other can be be compared if both the numerator and denominator can be compared.

7.8.8.6 Examples

```
<example xsi:type="RTO">
        <numerator xsi:type="MO" value="103.00" currency="USD"/>
        <denominator xsi:type="PQ" value="1" unit="day"/>
</example>
```

US$103/day.

The inner xsi:type declarations are always required.

7.8.9 PQ (physical quantity)

7.8.9.1 Description

Specializes QTY.

A dimensioned quantity expressing the result of measuring.

7.8.9.2 ISO/IEC 11404 syntax

```
type PQ = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  expression : ED,
  originalText : ED.TEXT,
  uncertainty : QTY,
  uncertaintyType : UncertaintyType,
  uncertainRange : IVL(QTY)
  value : Decimal,
  codingRationale : CodingRationale,
  unit : characterstring,
  translation : Set(PQR)
)
```

7.8.9.3 Attributes

7.8.9.3.1 value : Decimal: the number which is multiplied by the unit to make the PQ or PQR value if not nullFlavored.

7.8.9.3.2 unit : Code: The unit of measure specified in the Unified Code for Units of Measure (UCUM).

UCUM defines two forms of expression, case sensitive and case insensitive. *PQ* uses the case sensitive codes. The codeSystem OID for the case sensitive form is 2.16.840.1.113883.6.8. The default value for unit is the UCUM code “1” (unity).

Equality of physical quantities does not require the values and units to be equal independently. Value and unit is only how we represent physical quantities. For example 1 m equals 100 cm. Although the units are different and the values are different, the physical quantities are equal. Therefore, one should never expect a particular unit for a physical quantity but instead allow for automated conversion between different comparable units.

The unit shall come from UCUM, which only specifies unambiguous measurement units. Sometimes it is not clear how some measurements in healthcare map to UCUM codes.

The general pattern for a measurement is *value* unit of **Thing**. In this scheme, the PQ represents the *value* and the unit, and the **Thing** is described by some coded concept that is linked to the PQ by the context of use. This maps obviously to some measurements, such as **Patient Body Temperature** of *37* Celsius, and *250* mg/day of **Salicylate**.

However for some measurements that arise in healthcare, the scheme is not so obvious. Two classic examples are 5 Drinks of Beer, and 3 Acetominophen tablets. At first glance it is tempting to classify these measurments like this: *5* drinks of **Beer** and *3* **Acetominophen** tablets. The problem with this is that UCUM does not support units of "beer", "tablets" or "scoops".

The reason for this is that neither tablets nor scoops are proper units. What kind of tablets? How big is the glass? In these kinds of cases, the concept that appears to be a unit needs to be further specified before interoperability is established. If a correct amount is required, it is generally appropriate to specify an exact measurement with an appropriate UCUM unit. If this is not possible, the concept is not part of the measurement. UCUM provides a unit called unity for use in these cases. The proper way to understand these measurements as *3* 1 **Acetominophen** tablets, where 1 is the UCUM unit for unity, and the **Thing** has a qualifier. The context of use needs to provide the extra qualifying information.

7.8.9.3.3 codingRationale : CodingRationale: The reason that this PQ or PQR was provided. More than one reason may be given. For possible values, see 7.5.2.4.10 CD.codingRationale.

7.8.9.3.4 translation : Set(PQR): An alternative representation of the same physical quantity expressed in a different unit from a different unit code system and possibly with a different value.

It is not necessary for information processing entities to check and enforce that the translations are valid translations of the base unit, but they are allowed to do so, and to reject instances where the translations are not valid.

NOTE Translations are allowed to contain other representations in UCUM units, but there is generally no point to this as it is possible to convert from one UCUM form to another.

7.8.9.4 Equality

Two PQ values are equal if the value and units of their canonical forms are equal, or if their uncertainRanges are not null and not nullFlavored and equal. The attributes codingRationale, source, and any translations do not participate in the determination of equality.

7.8.9.5 Invariants

- a value or an uncertain range shall be provided if not nullFlavored;
- cannot provide both a value and an uncertain range;
- uncertainty types shall be a PQ;
- if uncertainties are provided, their canonical units shall match.

OCL for Invariants:

```
def: let unitsMatch (other : PQ) : Boolean =
        (canonical.unit = other.canonical.unit)

inv "null or value": isNull xor (value.oclIsDefined or
      uncertainRange.isNotNull)
inv "value xor uncertainRange": not (value.oclIsDefined and
      uncertainRange.isNotNull)
inv "uncertain types": uncertainRange.low.oclIsDefined implies
    uncertainRange.low.oclIsKindOf(PQ) and
   uncertainRange.high.oclIsDefined implies
    uncertainRange.high.oclIsKindOf(PQ) and
   uncertainty.oclIsDefined implies
    uncertainty.oclIsKindOf(PQ)
inv "uncertainties - canonicals":
   uncertainRange.low.oclIsDefined implies
    unitsMatch(uncertainRange.low) and
   uncertainRange.high.oclIsDefined implies
    unitsMatch(uncertainRange.high) and
   uncertainty.oclIsDefined implies unitsMatch(uncertainty)

context PQ::isDifference(other : QTY): Boolean
  post: other.oclIsKindOf(PQ) and
    canonical.unit.equals(other.oclAsTypeOf(PQ).canonical.unit)
```

7.8.9.6 Operations

7.8.9.6.1 canonical : PQ: The value converted to the form with canonical units. UCUM provides more information about canonical units.

7.8.9.6.2 comparable(other : QTY):BL: This and other can be compared if the units of the canonical forms are the same.

7.8.9.6.3 inverted():PQ: The inverted value of the PQ. Both value and unit must be inverted.

7.8.9.6.4 negated[–] : PQ: The negative value of this.

7.8.9.6.5 plus[+] (other : PQ) : PQ: The value of the addition of this and other; if the units do not match, NullFlavor NI.

7.8.9.6.6 minus[–] (other : PQ) : PQ: The value of the subtraction of other from this; if the units do not match, NullFlavor NI.

7.8.9.6.7 times[*] (other : PQ) : PQ: The value of the multiplication of this and other with appropriate changes to the units.

7.8.9.6.8 times[*] (other : REAL) : PQ: The value of the multiplication of this and other.

7.8.9.6.9 dividedBy[/] (other : PQ) : PQ: The value of this divided by other with appropriate changes to the units. If other is 0, then the result is nullFlavor NI.

7.8.9.6.10 dividedBy[/] (other : REAL) : PQ: The value of this divided by other. If other is 0, then the result is NullFlavor NI.

7.8.9.6.11 abs() : PQ: The absolute value of this.

7.8.9.6.12 max(other : PQ) : PQ: The maximum of this and other; if the units do not match, NullFlavor NI.

7.8.9.6.13 min(other : PQ) : PQ: The minimum of this and other; if the units do not match, NullFlavor NI.

7.8.9.6.14 toInterval() : IVL(PQ) : Converts this value to an interval that expresses the range covered by the precision of the value.

7.8.9.7 Examples

7.8.9.7.1 Plain value

```
<example xsi:type="PQ" value="1.1" unit="mg/mL"/>
```

1,0 mg/ml.

```
<example xsi:type="PQ" value="1.1" unit="mg/mL" codingRationale="R">
  <translation codingRationale="O" value="0.0011"
     codeSystem="2.16.840.1.113883.19.10" code="grams/litre"/>
</example>
```

1,0 mg/ml as a translation from the original measurement of 0,001 1g/l in a local code system into UCUM units.

7.8.9.7.2 Uncertain range

```
<doseQuantity xsi:type="PQ" units="1">
  <uncertainRange>
    <low xsi:type="PQ" value="1"/>
    <high xsi:type="PQ" value="2"/>
  </uncertainRange>
</doseQuantity>
```

This URG(PQ) specifies that patient should take 1 to 2 tablets. This might be part of a prescription order such as "By mouth, take 1 to 2 tablets every 4 to 6 hours when needed for severe pain to a maximum of 8 per day". The unit is "1" – the default UCUM unit, so this matches the units on the low and high in the uncertainRange.

7.8.9.7.3 Using expressions

```
<substanceAdministration>
   ...
   <doseQuantity xsi:type="PQ" nullFlavor="DER" unit="mL">
     <expression mediaType="application/hl7-factor+xml">
       <xml>
         <coefficient value="30" unit="mL/kg"/>
         <factor value="bodyMass"/>
       </xml>
     </expression>
   </doseQuantity>
   ...
   <derivedFrom>
     <localVariableName value="bodyMass"/>
     <monitoringObservation>
       <code code="29463-7" codeSystem="2.16.840.1.113883.11.16492">
          <displayName value="BODY WEIGHT:MASS:PT:^PATIENT:QN"/>
       </code>
     </monitoringObservation>
   </derivedFrom>
 </substanceAdministration>
```

This example uses an HL7 specific language to illustrate how the expression attribute is used.

The dose quantity of the substance that is being administered depends on the patient's body mass, 30 mg per kilogram of body mass. Rather than providing an actual value for the patient's body mass, since it may be unknown or may change, the maximum dose quantity is given in terms of an expression. Since no value attribute is provided, a nullFlavor shall be provided; DER is the appropriate choice when an expression is provided.

The body mass may be found by checking for any observations matching the LOINC code 29463-7; if no matching observations can be found, the outcome of the expression will be null or a nullFlavor, as the value cannot be known.

NOTE Factor is an HL7 specific language documented in the Abstract Data Types Specification.

7.8.10 PQ.TIME

7.8.10.1 Description

A flavor that constrains PQ.

PQ.TIME constraints PQ so that it shahll have units that describe a period of time.

7.8.10.2 Invariants

— the units shall be a measure of time ["such as, "s" (second), "min" (minute), "h" (hour), "d" (day), "wk" (week), "a" (year)].

OCL for Invariants:

```
inv "must be a unit of time": canonical.unit = "s"
```

7.8.11 PQR (physical quantity representation)

7.8.11.1 Description

Specializes CD.CV.

An extension of the coded value datatype representing a physical quantity using a unit from any code system. Used to show alternative representation for a physical quantity. The coded value represents the unit (usually in some other coding system than UCUM).

7.8.11.2 ISO/IEC 11404 syntax

```
type PQR = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  code : characterstring,
  codeSystem : characterstring,
  codeSystemName : characterstring,
  codeSystemVersion : characterstring,
  valueSet : characterstring,
  valueSetVersion : characterstring,
  displayName : ST,
  originalText : ED.TEXT,
  codingRationale : CodingRationale),
  translation : Set(CD),
  source : CD,
  value : Decimal
)
```

7.8.11.3 Attributes

7.8.11.3.1 value: Decimal : The magnitude of the measurement value in terms of the unit specified by this code.

7.8.11.4 Equality

Two PQR values are equal if their value, code and codeSystem are equal. The other attributes do not participate in the determination of equality.

7.8.11.5 Invariants

— a unit is required;

— no source;

— no originalText.

NOTE There is only one originalText, that for the physical quantity itself.

OCL for Invariants:

```
inv "null or value": isNull xor value.oclIsDefined
inv "no originalText": originalText.oclIsUndefined
inv "no updateMode or History on PQR": noUpdateOrHistory
inv "no translations": translation.isEmpty
inv "no source": source.oclIsUndefined
```

7.8.12 MO (monetary amount)

7.8.12.1 Description

Specializes QTY.

An MO is a quantity expressing the amount of money in some currency.

Currencies are the units in which monetary amounts are denominated in different economic regions. While the monetary amount is a single kind of quantity (money) the exchange rates between the different units are variable. This is the principle difference between PQ and MO, and the reason why currency units are not physical units.

7.8.12.2 ISO/IEC 11404 syntax

```
type MO = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  expression : ED,
  originalText : ED.TEXT,
  uncertainty : QTY,
  uncertaintyType : UncertaintyType,
  uncertainRange : IVL(QTY)
  value : Decimal,
  currency : characterstring
)
```

7.8.12.3 Attributes

7.8.12.3.1 value : Decimal: The value of the MO. MO values are usually precise to 0.01 (one cent, penny, paisa, etc.) or 1 (yen, forint, etc), though other precisions exist. ISO 4217 documents the appropriate precision for most currencies.

7.8.12.3.2 currency : Code: The currency unit as defined in ISO 4217.

7.8.12.4 Equality

Two MO values are equal if their value and currency attributes are equal, or if their uncertainRanges are not null and not nullFlavored and equal.

7.8.12.5 Invariants

- if not nullFlavored, a value or an uncertain range shall be present;
- if not nullFlavored, a currency shall be defined;
- cannot provide both a value and an uncertain range;
- uncertainty types shall be a PQ;
- if uncertainties are provided, their canonical units shall match.

OCL for Invariants:

```
def: let currencyMatches (other : MO) : Boolean =
     currency = other.currency

inv "null or currency": isNull xor currency.oclIsDefined
inv "null or value": isNull xor (value.oclIsDefined or
         uncertainRange.isNotNull)
inv "value xor uncertainRange": not (value.oclIsDefined and
         uncertainRange.isNotNull)
inv "uncertain types": uncertainRange.low.oclIsDefined implies
       uncertainRange.low.oclIsKindOf(MO) and
      uncertainRange.high.oclIsDefined implies
       uncertainRange.high.oclIsKindOf(MO) and
      uncertainty.oclIsDefined implies
       uncertainty.oclIsKindOf(MO)
inv "uncertainties - currencies":
   uncertainRange.low.oclIsDefined implies
     currencyMatches(uncertainRange.low) and
   uncertainRange.high.oclIsDefined implies
     currencyMatches(uncertainRange.high) and
   uncertainty.oclIsDefined implies
     currencyMatches(uncertainty)

context CO::isDifference(other : QTY): Boolean
  post: other.oclIsKindOf(MO)
```

7.8.12.6 Operations

7.8.12.6.1 plus[+] (other : MO) : MO: The value of the addition of this and other; if the currencies do not match, NullFlavor NI.

7.8.12.6.2 minus[–] (other : MO) : MO: The value of the subtraction of other from this; if the currencies do not match, NullFlavor NI.

7.8.12.6.3 times[*] (other : REAL) : MO: The value of the multiplication of this by other.

7.8.12.6.4 dividedBy[/] (other : REAL) : MO: The value of this divided by other. If other is 0, then the result is NullFlavor NI.

7.8.12.6.5 max(other : MO) : MO: The maximum of this and other.

7.8.12.6.6 min(other : MO) : MO: The minimum of this and other.

7.8.12.6.7 comparable(other : QTY):BL: This and other can be compared if their currencies are the same.

7.8.12.7 Examples

```
<example xsi:type="MO" value="42" currency="AUD"/>
```

A$42 Australian dollars.

7.8.13 TS (point in time)

7.8.13.1 Description

Specializes QTY.

A quantity specifying a point on the axis of natural time. A point in time is most often represented as a calendar expression.

7.8.13.2 ISO/IEC 11404 syntax

```
type TS = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  expression : ED,
  originalText : ED.TEXT,
  uncertainty : QTY,
  uncertaintyType : UncertaintyType,
  uncertainRange : IVL(QTY)
  value : characterstring
)
```

7.8.13.3 Attributes

7.8.13.3.1 value : String: The value of the TS. value is a string with the format "YYYY[MM[DD[HH[MM[SS[.U[U[U[U]]]]]]]][+|-ZZzz]" that conforms to the constrained ISO 8601 defined in ISO/IEC 8824 (all parts) (ASN.1) under Clause 32 (generalized time). The format should be used to the degree of precision that is appropriate.

7.8.13.4 Equality

Two nonNull TS values are only equal if their time and precision are equal, or if their uncertainRanges are not null and not nullFlavored and equal. If both TS value have timezones, the values should be be corrected for timezone before comparison. If neither TS value has a timezone, then they may be compared for equality without correction. If only one TS value has a timezone, then the equality is NullFlavor NI.

7.8.13.5 Invariants

- if the TS is not nullFlavored, a value or an uncertain range shall be present;
- if a value is present, at least a full year shall be specified;
- uncertainty types shall be a PQ with a unit of TIME;
- cannot provide both a value and an uncertain range.

OCL for Invariants:

```
def: let hasTimezone : Boolean = value.pos("+") > 0
    or value.pos("-") > 0
inv "value xor uncertainRange": not (value.oclIsDefined and
        uncertainRange.isNotNull)
inv "null or value": isNotNull implies (value.oclIsDefined
        or uncertainRange.isNotNull)
inv "uncertain types": uncertainRange.low.oclIsDefined implies
       uncertainRange.low.oclIsKindOf(PQ) and
      uncertainRange.high.oclIsDefined implies
       uncertainRange.high.oclIsKindOf(PQ) and
      uncertainty.oclIsDefined implies
       uncertainty.oclIsKindOf(PQ)
inv "uncertainties - units": uncertainRange.low.oclIsDefined
      implies uncertainRange.canonical.unit = "s" and
     uncertainRange.high.oclIsDefined implies
     uncertainRange.high.canonical.unit = "s" and
     uncertainty.oclIsDefined implies
     uncertainty.canonical.unit = "s"

context TS::isDifference(other : QTY): Boolean
  post: other.oclIsKindOf(TS) and canonical.unit = "s"
```

7.8.13.6 Operations

7.8.13.6.1 plus(+) (other : PQ) : TS: The value of the addition of this and other; if other.units are not a time, NullFlavor NI.

7.8.13.6.2 minus[–] (other : PQ) : TS: The value of the subtraction of other from this; if other.units are not a time, NullFlavor NI.

7.8.13.6.3 minus[–] (other : TS) : PQ: The value of the subtraction of other from this; the return value will have units that are a time.

7.8.13.6.4 max(other : TS) : TS: The maximum of this and other.

7.8.13.6.5 min(other : TS) : TS: The minimum of this and other.

7.8.13.6.6 toInterval() : IVL(TS): Converts this value to an interval that expresses the range covered by the precision.

7.8.13.6.7 precision() : Integer : The number of significant digits of the timestamp value.

7.8.13.6.8 comparable(other : QTY):BL: This and other can be compared if other is a TS.

7.8.13.7 Examples

7.8.13.7.1 Instant in time

```
<example xsi:type="TS" value="20031101234511+0500"/>
```

11:45 p.m. on 01 Nov 2003 at +5 from UTC (e.g. US eastern).

7.8.13.7.2 Birth date

```
<example xsi:type="TS" value="1945"/>
```

Patient was born in 1945. Month and day are unknown.

The outcome of toIVL() for this example would be the following interval:

```
<example xsi:type="IVL_TS" lowClosed="true" highClosed="false">
  <low value="194501010000.0000"/>
  <high value="194601010000.0000"/>
</example>
```

7.8.14 TS.DATE

7.8.14.1 Description

A flavor that constrains TS.

TS.DATE constrains TS so that it may only contain a date value.

7.8.14.2 Invariants

- no timezone;
- no hours, minutes, seconds or milliseconds.

OCL for Invariants

```
inv "Date": not hasTimezone and value.size <= 8
```

7.8.15 TS.DATE.FULL

7.8.15.1 Description

A flavor that constrains TS.DATE

TS.DATE.FULL constrains TS.DATE so that it shall contain reference to a particular day.

7.8.15.2 Invariants

- a full day shall be specified.

OCL for Invariants

```
inv "Full Date": value.size = 8
```

7.8.16 TS.DATETIME

7.8.16.1 Description

A flavor that constrains TS

TS.DATETIME constrains a TS so that its precision cannot be more precise than seconds.

7.8.16.2 Invariants

— milliseconds shall be blank.

OCL for Invariants

```
inv "DateTime": (value.size <= 14) or (hasTimezone and
    value.size <= 19)
```

7.8.17 TS.DATETIME.FULL

7.8.17.1 Description

A flavor that constrains TS.DATETIME.

TS.DATETIME.FULL constrains TS.DATETIME so that it shall contain reference to a particular second with a timezone.

7.8.17.2 Invariants

— a timezone is required;

— a full time, including seconds and timezone, is required.

OCL for Invariants

```
inv "Full DateTime": value.size = 19 and hasTimezone
```

7.8.18 TS.INSTANT

7.8.18.1 Description

A flavor that constrains TS.

TS.INSTANT constrains TS so that it shall contain reference to a particular instant of time, accurate to four decimal places on the second, with a timezone.

7.8.18.2 Invariants

— a timezone is required;

— a full time, including fractions of seconds to 4 decimal and timezone, is required.

OCL for Invariants

```
inv "Instant": value.size = 24 and hasTimezone
```

7.9 Collections of datatypes

7.9.1 Overview

These datatypes data types are collections of discrete elements (see Figure 8).

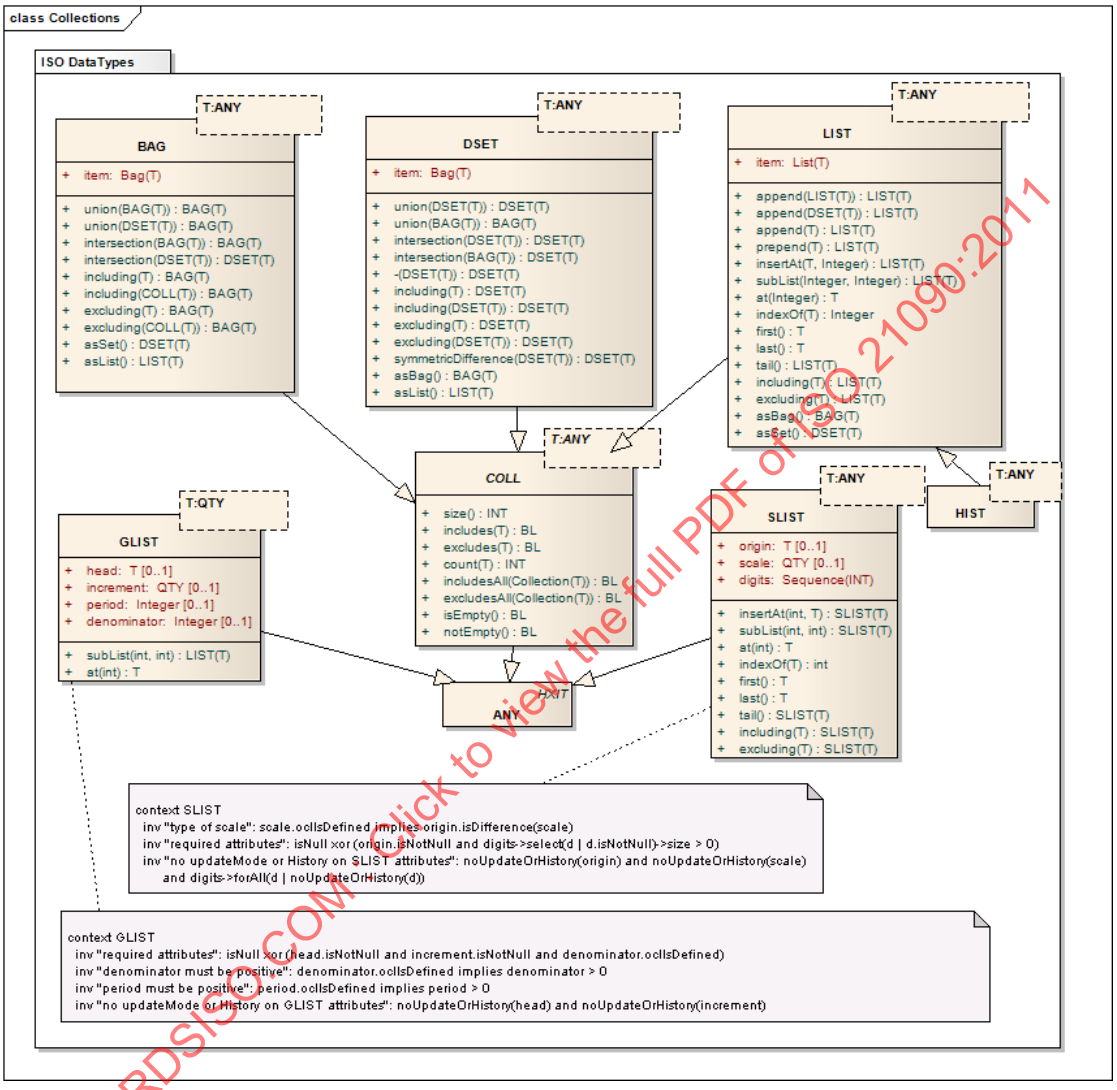

Figure 8 — Collection datatypes

7.9.2 COLL

7.9.2.1 Description

Abstract. Specializes ANY.

Parameter: T : ANY

A collection of values which can be enumerated using an iterator.

7.9.2.2 Operations

7.9.2.2.1 size() : INT: The number of elements in this collection.

7.9.2.2.2 includes(object : T) : BL: True if object is an element of this collection. The equals operation is used to evaluate whether object is an element of this collection. This is also known as "contains“: a collection contains object o if includes(o) returns true.

7.9.2.2.3 excludes(object : T) : BL: True if object is not an element of this collection. The equals operation is used to evaluate whether object is not an element of this collection.

7.9.2.2.4 count(object : T) : INT: The number of times that object occurs in the this collection. The equals operation is used to evaluate how many times object is in the collection.

7.9.2.2.5 includesAll(c2 : Collection(T)) : BL: True if this collection contains all the elements of c2.

7.9.2.2.6 excludesAll(c2 : Collection(T)) : BL: True if this collection contains none of the elements of c2.

7.9.2.2.7 isEmpty() : BL: True if this collection is the empty collection.

7.9.2.2.8 notEmpty() : BL: True if this collection is not the empty collection.

7.9.2.3 Equality

Equality is not defined for COLL as it is an abstract type.

7.9.3 DSET (discrete set)

7.9.3.1 Description

Specializes COLL.

Parameter: T : ANY.

A collection that contains distinct and discrete values in no particular order.

Valid (non-nullFlavored) discrete sets shall not contain duplicate items. The context of use shall define how elements are compared when checking set element uniqueness. By default, the uniqueness definition is based on the equality rules defined in this International Standard: Discrete sets shall not contain different values that are equal, and they shall not contain items that are null or have a nullFlavor, where the equality cannot be evaluated. When a discrete set is actually used, the context of use may specify an alternative definition for how uniqueness is evaluated. This alternative definition may allow for nullFlavored values in a proper set. Information processing entities providing alternative definitions for the uniqueness of a set shall make it clear in the conformance statement how such definitions are provided so that there is no ambiguity.

While proper (non-nullFlavored) sets do not contain values that do not meet the definition of uniqueness, discrete sets with a nullFlavor may contain elements duplicate values or values that have a nullFlavor. Discrete sets that are labelled as mandatory cannot have a nullFlavor and therefore cannot contain such values.

7.9.3.2 ISO/IEC 11404 syntax

```
type DSET (T : ANY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  item : Bag(T)
)
```

NOTE DSET specifies a set of discrete items. If the items are in an ordered but discrete domain (i.e. INT) and they represent a sequence of concurrent values such as 2,3,4,5, the same set can also be specified by a QSET such as IVL(2..5). However, for all other cases, there is no overlap between QSET and DSET.

7.9.3.3 Attributes

7.9.3.3.1 item : Bag(T): The contents of the set.

This is an example of the primitive type wrapping pattern. See 6.3 for more details.

The items are held in a bag because the context of use specifies exactly how uniqueness is specified for the DSET. The OCL kernel set applies a fixed definition of equality (the equality specified for the type) which may be less granular that the context of use. Though the internal structure allows duplicates, all the items in the set shall be unique according the definition provided by the context of use. If the context of use does not make this clear, the default behaviour is the equality definitions provided in this International Standard.

NOTE There is no support for the formal definition of the uniqueness constraints because of the cost of providing such a framework and the lack of apparent use for one.

7.9.3.4 Equality

Two nonNull DSETs are equal if they contain the same elements.

NOTE 1 The determination of element content is based on the same semantic equals as defined in this International Standard, so it is possible that a DSET(CD) can be equal to DSET(CS), for instance.

NOTE 2 It is possible for a DSET(INT) and a QSET(INT) to be equal, if they both contain the same elements.

7.9.3.5 Invariants

(none)

NOTE Rules about updateMode or history are applied to item where BAG(T) is used.

7.9.3.6 Operations

7.9.3.6.1 union(s : DSET(T)) : DSET(T): The SET containing all elements of this plus all the elements of other, with any duplicates removed. *union* is an alias for *including*.

7.9.3.6.2 union(bag : BAG(T)) : BAG(T): The SET containing all elements of this plus all the elements of other, with any duplicates removed

7.9.3.6.3 intersection(s : DSET(T)) : DSET(T): The intersection of this and s (i.e, the SET of all elements that are in both this and s).

7.9.3.6.4 intersection(bag : BAG(T)) : DSET(T): The intersection of this and bag.

7.9.3.6.5 minus[–](s : DSET(T)) : DSET(T): The elements of this, which are not in s.

7.9.3.6.6 including(object : T) : DSET(T): The SET containing all elements of this plus object if it is not already in the set. *including* is an alias for *union*.

7.9.3.6.7 including(other : DSET(T)) : DSET(T): The SET containing all elements of this plus all the elements of other, with any duplicates removed.

7.9.3.6.8 excluding(object : T) : DSET(T): The SET containing all elements of this without object.

7.9.3.6.9 excluding(other : DSET(T)) : DSET(T): The SET containing all elements of this with any elements of other removed .

7.9.3.6.10 symmetricDifference(s : DSET(T)) : DSET(T): The sets containing all the elements that are in this or s.

7.9.3.6.11 asList() : LIST(T): A sequence that contains all the elements from this, in an undefined order.

7.9.3.6.12 asBag() : BAG(T): The bag that contains all the elements from this.

7.9.3.7 Examples

7.9.3.7.1 Integer Sets

```
<example xsi:type="DSET_INT">
  <item value="3"/>
  <item value="6"/>
  <item value="9"/>
  <item value="11"/>
</example>
```

The set of integers (3,6,9,11). The set type specifies the type of the items in the set (xsi:type="DSET" is not correct as SET is a generic type).

```
<example xsi:type="DSET_INT">
  <item value="11"/>
  <item value="6"/>
  <item value="9"/>
  <item value="3"/>
</example>
```

This set is identical to the previous set; order has no significance in sets.

7.9.3.7.2 Problems with sets

```
<example xsi:type="DSET_TEL">
  <item value="tel:+15556667777" use="H"/>
  <item nullFlavor="UNK" use="WP"/>
</example>
```

A set of telephone numbers, with a known home telephone number, and an unknown work number. IMPORTANT: This is an illegal set: sets cannot contain nullValues.

```
<example xsi:type="DSET_TEL" nullFlavor="UNK">
  <item value="tel:+15556667777" use="H"/>
  <item nullFlavor="UNK" use="WP"/>
</example>
```

This is the same set properly represented. Because one of the values is unknown, the set itself is unknown.

In general, addresses and telephone numbers should not be modelled as DSETs for this reason, as once they are marked mandatory, unknown numbers can no longer be represented.

```
<example xsi:type="DSET_REAL">
  <item value="3.1"/>
  <item value="3.5"/>
</example>
```

Where the items are not discrete (PQ, MO, REAL), a discrete set is rather ambiguous. For example, is 3.103 a member of this set or not? For this reason, DSET should not be used with these types.

7.9.4 LIST (sequence)

7.9.4.1 Description

Specializes COLL.

Parameter: T : ANY.

A collection that contains discrete (but not necessarily distinct) values in a defined sequence. Values are also assigned an offset; the first value has the offset of zero.

The sequence is an ordered collection of values, but no particular order is inherently associated with the sequence. The meaning of the order of the items should be defined where a LIST is used. In some cases, the order is fixed (e.g. HIST), but in other cases, the order is not fixed: only the meaning associated with the order in the instance is defined (e.g. EN, AD).

7.9.4.2 ISO/IEC 11404 syntax

```
type LIST (T : ANY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  item : Sequence(T)
)
```

7.9.4.3 Attributes

7.9.4.3.1 item : Sequence(T): The contents of the sequence.

This is an example of the primitive type wrapping pattern. See 6.3 for more details.

7.9.4.4 Equality

Two lists are equal if (and only if) they are both empty or if they contain the same elements in the same order.

NOTE 1 The determination of element content is based on the same semantic equals as defined in this International Standard, so it is possible that a LIST(ED) can be equal to LIST(ST), for instance.

NOTE 2 SLIST equality is based on the equality of a sequence of values, so a SLIST<T> may be equal to a LIST<T>.

7.9.4.5 Invariants

(none)

NOTE Rules about updateMode or history are applied to item where BAG(T) is used.

7.9.4.6 Operations

7.9.4.6.1 append(s : LIST(T)) : LIST(T): The LIST consisting of all elements in this, followed by all elements in s.

7.9.4.6.2 append(s : DSET(T)) : LIST(T): The LIST consisting of all elements in this, followed by all elements in s in some arbitrary order. This is an alias for including.

7.9.4.6.3 append (object: T) : LIST(T): The LIST of elements, consisting of all elements of this, followed by object.

7.9.4.6.4 prepend(object : T) : LIST(T): The LIST consisting of object, followed by all elements in this.

7.9.4.6.5 insertAt(object : T, index : Integer) : LIST(T): The LIST consisting of this with object inserted at position index. If index is equal or greater than the length of the LIST, the value will be null.

7.9.4.6.6 subList(lower : Integer, upper : Integer) : LIST(T): The subLIST of this starting at number lower, up to and including element number upper. If lower or upper are equal or greater than the length of the LIST or less than 0, or lower is greater than upper, the value will be null.

7.9.4.6.7 at(i : Integer) : T: The i-th element of LIST. If i is equal or greater than the length of the LIST, the value will be null. The first element of the list has i = 0

7.9.4.6.8 indexOf(obj : T) : Integer: The index of object obj in the LIST or null if the item exists other than once (not at all, or multiple times).

7.9.4.6.9 first() : T: The first element in this or null if it is empty.

7.9.4.6.10 last() : T: The last element in this or null if it is empty.

7.9.4.6.11 tail() : LIST(T): The list with the first element removed or null if it is empty.

7.9.4.6.12 including(object : T) : LIST(T): The LIST containing all elements of this plus object added as the last element. This is an alias for append.

7.9.4.6.13 excluding(object : T) : LIST(T): The LIST containing all elements of this apart from all occurrences of object. The order of the remaining elements is not changed.

7.9.4.6.14 asBag() : BAG(T): The Bag containing all the elements from this, including duplicates, or a nullFlavor as appropriate.

7.9.4.6.15 asSet() : DSET(T): The Set containing all the elements from this, with duplicated removed.

7.9.4.7 Examples

```
<example xsi:type="LIST_INT">
  <item value="3"/>
  <item value="11"/>
  <item value="6"/>
  <item value="9"/>
</example>
```

A list of integers. The order is significant and must always be maintained.

7.9.5 GLIST (generated sequence)

7.9.5.1 Description

Specializes ANY.

Parameter: T : QTY.

A periodic or monotone sequence of values generated from a few parameters, rather than being enumerated. Used to specify regular sampling points for biosignals.

7.9.5.2 ISO/IEC 11404 syntax

```
type GLIST (T : ANY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  head: T,
  increment : QTY,
  denominator : integer,
  period : integer
)
```

7.9.5.3 Attributes

7.9.5.3.1 head : T: The first item in this sequence. This is the start-value of the generated list.

7.9.5.3.2 increment : QTY: The difference between one value and its previous different value.

EXAMPLE To generate the sequence (1; 4; 7; 10; 13; ...) the increment is 3; likewise to generate the sequence (1; 1; 4; 4; 7; 7; 10; 10; 13; 13; ...) the increment is also 3. The actual type QTY is dictated by the type of T. The value of increment shall be positive.

7.9.5.3.3 denominator : Integer: The integer by which the index for the sequence is divided, effectively the number of times the sequence generates the same sequence item value before incrementing to the next sequence item value.

EXAMPLE: To generate the sequence (1; 1; 1; 2; 2; 2; 3; 3; 3; ...) the denominator is 3.

The use of the denominator is to allow multiple generated sequences to periodically scan a multidimensional space. For example an (abstract) TV screen uses two such generators for the columns and rows of pixels. For instance, if there are 200 scan lines and 320 raster colunmns, the column-generator would have denominator 1 and the line-generator would have denominator 320.

7.9.5.3.4 period : Integer: If not null or nullFlavored, specifies that the sequence alternates, i.e. after this many increments, the sequence item values roll over to start from the initial sequence item value.

EXAMPLE: The sequence (1; 2; 3; 1; 2; 3; 1; 2; 3; ...) has period 3; also the sequence (1; 1; 2; 2; 3; 3; 1; 1; 2; 2; 3; 3; ...) has period 3.

The period allows to repeatedly sample the same sample space. The "waveform" of this periodic generator is always a "saw", just like the x-function of your oscilloscope.

7.9.5.4 Equality

GLIST is a list generator. Two GLISTs are equal if they specify the same sequence of elements.

Since GLISTs are infinite, and LISTs cannot be infinite, they can never been equal.

7.9.5.5 Invariants

- if the GLIST is not nullFlavored, all attributes but the period are required;
- denominator shall be positive;
- the period shall be positive.

OCL for Invariants:

```
inv "required attributes": isNull xor (head.isNotNull and
     increment.isNotNull and denominator.oclIsDefined)
inv "denominator must be positive": denominator.oclIsDefined
      implies denominator > 0
inv "period must be positive": period.oclIsDefined implies
      period > 0
inv "no updateMode or History on GLIST attributes":
     noUpdateOrHistory(head) and noUpdateOrHistory(increment)
```

7.9.5.6 Operations

7.9.5.6.1 subList(lower : Integer, upper : Integer) : LIST(T): A subLIST of this GLIST, starting at number lower, up to and including element number upper. If lower or upper are equal or greater than the length of the LIST or less than 0, or lower is greater than upper, the value will be null.

7.9.5.6.2 at(i : Integer) : T: The i-th element of LIST. If i is equal or greater than the length of the LIST, the value will be null.

7.9.5.7 Examples

```
<example xsi:type="GLIST_PQ" period="100" denominator="100">
  <head value="0" unit="V"/>
  <increment xsi:type="PQ" value="1" unit="mV"/>
</example>
```

The x-wave of a digital oscillograph scanning between 0 mV and 100 mV in 100 steps of 1 mV. The frequency is unknown from these data as we do not know how much time elapses between each step of the index.

```
<example xsi:type="GLIST_TS" denominator="1">
  <head value="20020729203000"/>
  <increment xsi:type="PQ" value="100" unit="us"/>
</example>
```

A timebase from June 29, 2002 at 8:30 p.m., with 100 µs between each steps of the index. If combined with the previous generator as a second sampling dimension this would now describe our digital oscilloscope's x-timebase as 1 mV per 100 µs. At 100 steps per period, the period is 10 ms, which is equal to a frequency of 100 Hz.

Other examples:

Head	Increment	Deno-minator	Period	Meaning
0	1	1	∞ (NullFlavor.PINF)	The identity-sequence where each item is equal to its index.
198706052000	2 h	1	∞ (NullFlavor.PINF)	Sequence starting on June 5, 1987 at 7 p.m. and incrementing every two hours: 9 p.m., 11 p.m., 1 a.m. (June 6), 3 a.m., 5 a.m. and so on.
0 V	1 mV	1	100	The x-wave of a digital oscillograph scanning between 0 mV and 100 mV in 100 steps of 1 mV. The frequency is unknown from these data as it is not known how much time elapses between each step of the index.
2002072920300	100 µs	1	∞ (NullFlavor.PINF)	A timebase from June 29, 2002 at 8:30 p.m. with 100 µs between each step of the index. If combined with the previous generator as a second sampling dimension, this would now describe the digital oscilloscope's x-timebase as 1 mV per 100 µs. At 100 steps per period, the period is 10 ms, which is equal to a frequency of 100 Hz.
0 V	1 mV	100	100	Combining this generator to the previous two generators could describe a three-dimensional sampling space with two voltages and time. This generator also steps at 1 mV and has 100 steps per period, however, it only steps every 100 index increments, so, the first voltage generator makes one full cycle before this generator is incremented. One can think of the two voltages as "rows" and "columns" of a "sampling frame". With the previous generator as the timebase, this results in a scan of sampling frames of 100 mV × 100 mV with a frame rate of 1 Hz.

7.9.6 SLIST (sampled sequence)

7.9.6.1 Description

Specializes ANY.

Parameter: T : QTY.

A sequence of sampled values scaled and translated from a list of integer values. Used to specify sampled biosignals.

7.9.6.2 ISO/IEC 11404 syntax

```
type SLIST (T : ANY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  origin : T,
  scale : QTY,
  digits : Sequence(INT)
)
```

7.9.6.3 Attributes

7.9.6.3.1 origin : T: The origin of the list item value scale. The physical quantity that a zero-digit in the sequence would represent.

7.9.6.3.2 scale : QTY: A ratio-scale quantity that is factored out of the digit sequence. The actual type QTY will be dictated by the type of T.

7.9.6.3.3 digits : Sequence(INT): A sequence of raw digits for the sample values. This is typically the raw output of an A/D converter.

7.9.6.4 Equality

SLIST specifies a LIST. Two SLISTs are equal if they specify the same sequence of values.

NOTE Because SLIST specifies a LIST, and equality is basd on the LIST that is specified, it is possible that a LIST<PQ> will be equal to a SLIST<PQ>.

7.9.6.5 Invariants

— scale shall be a difference from the origin;

— if the SLIST is not nullFlavored, an origin and at least one digit is required.

OCL for Invariants:

```
inv "required attributes": isNull xor (origin.isNotNull and
     digits->select(d | d.isNotNull)->size > 0)
inv "type of scale": scale.oclIsDefined implies
     origin.isDifference(scale)
inv "no updateMode or History on SLIST attributes":
noUpdateOrHistory(origin) and noUpdateOrHistory(scale)
     and digits->forAll(d | noUpdateOrHistory(d))
```

7.9.6.6 Operations

7.9.6.6.1 insertAt(object : T, index : Integer) : SLIST(T): The SLIST consisting of this SLIST with object inserted at position index. If index is equal to or greater than the length of the LIST, the value will be null. If object does not compare with the other objects in the list, the value will be null.

7.9.6.6.2 subList(lower : Integer, upper : Integer) : SLIST(T): The subLIST of this starting at number lower, up to and including element number upper. If lower or upper are equal or greater than the length of the LIST or less than 0, or lower is greater than upper, the value will be null.

7.9.6.6.3 at(i : Integer) : T: The i-th element of LIST. If i is equal or greater than the length of the LIST, the value will be null.

7.9.6.6.4 indexOf(obj : T) : Integer: The index of object obj in the LIST or null if the item exists more than once.

7.9.6.6.5 first() : T: The first element in this or null if it is empty.

7.9.6.6.6 last() : T: The last element in this or null if it is empty.

7.9.6.6.7 tail() : SLIST(T): The list with the first element removed or null if it is empty.

7.9.6.6.8 including(object : T) : SLIST(T): The LIST containing all elements of this plus object added as the last element.

7.9.6.6.9 excluding(object : T) : SLIST(T): The LIST containing all elements of this apart from all occurrences of object. The order of the remaining elements is not changed.

7.9.6.7 Examples

```
<example xsi:type="SLIST_PQ">
   <origin value='0' unit='uV'/>
   <scale xsi:type="PQ" value='2.5' unit='uV'/>
   <digit value="-4"/>
   <digit value="-13"/>
   <digit value="-18"/>
   <digit value="-18"/>
   <digit value="-18"/>
   <digit value="-17"/>
   <digit value="-16"/>
   <digit value="-16"/>
   <digit value="-16"/>
   <digit value="-16"/>
   <digit value="-16"/>
   <digit value="-17"/>
   <digit value="-18"/>
   <digit value="-18"/>
   <digit value="-1"/>
   <digit value="-17"/>
   <digit value="-16"/>
   <digit value="-16"/>
   <digit value="-16"/>
   <digit value="-15"/>
   <digit value="-13"/>
   <digit value="-11"/>
   <digit value="-10"/>
   <digit value="-10"/>
   <digit value="-9"/>
   <digit value="-6"/>
   <digit value="-4"/>
```

```
   <digit value="-5"/>
   <digit value="-5"/>
   <digit value="-3"/>
   <digit value="-2"/>
   <digit value="-2"/>
   <digit value="-1"/>
   <digit value="1"/>
   <digit value="2"/>
   <digit value="3"/>
   <digit value=""/>
   <digit value="7"/>
   <digit value="8"/>
   <digit value="9"/>
   <digit value="10"/>
   <digit value="11"/>
   <digit value="12"/>
   <digit value="13"/>
   <digit value="15"/>
   <digit value="17"/>
   <digit value="19"/>
   <digit value="21"/>
   <digit value="23"/>
   <digit value="25"/>
   <digit value="27"/>
   <digit value="29"/>
   <digit value="30"/>
   <digit value="30"/>
   <digit value="31"/>
   <digit value="34"/>
   <digit value="37"/>
   <digit value="40"/>
   <digit value="43"/>
   <digit value="45"/>
   <digit value="4"/>
   <digit value="46"/>
   <digit value="46"/>
   <digit value="46"/>
   <digit value="46"/>
   <digit value="47"/>
   <digit value="49"/>
   <digit value="51"/>
   <digit value="53"/>
   <digit value="55"/>
   <digit value="57"/>
   <digit value="59"/>
   <digit value="60"/>
   <digit value="59"/>
   <digit value="58"/>
   <digit value="58"/>
   <digit value="58"/>
   <digit value="57"/>
   <digit value="56"/>
   <digit value="56"/>
   <digit value="56"/>
   <digit value="57"/>
   <digit value="57"/>
   <digit value="5"/>
   <digit value="53"/>
   <digit value="50"/>
   <digit value="47"/>
   <digit value="45"/>
   <digit value="74"/>
   <digit value="51"/>
   <digit value="38"/>
   <digit value="33"/>
   <digit value="31"/>
   <digit value="28"/>
```

```
   <digit value="25"/>
   <digit value="21"/>
   <digit value="16"/>
   <digit value="14"/>
   <digit value="15"/>
   <digit value="13"/>
   <digit value="9"/>
   <digit value="7"/>
   <digit value="4"/>
   <digit value="1"/>
   <digit value="-1"/>
   <digit value="-3"/>
   <digit value="-4"/>
   <digit value="-6"/>
   <digit value="-10"/>
   <digit value="-12"/>
   <digit value="-13"/>
   <digit value="-12"/>
   <digit value="-12"/>
   <digit value="-17"/>
   <digit value="-18"/>
   <digit value="-18"/>
   <digit value="-18"/>
   <digit value="-19"/>
   <digit value="-20"/>
   <digit value="-21"/>
   <digit value="-20"/>
   <digit value="-20"/>
   <digit value="-20"/>
   <digit value="-20"/>
   <digit value="-2"/>
   ...
   <digit value="2"/>
   <digit value="1"/>
   <digit value="0"/>
   <digit value="0"/>
   <digit value="0"/>
   <digit value="1"/>
   <digit value="2"/>
   <digit value="2"/>
   <digit value="1"/>
   <digit value="1"/>
   <digit value="1"/>
   <digit value="0"/>
   <digit value="-1"/>
   <digit value="0"/>
   <digit value="1"/>
   <digit value="1"/>
   <digit value="1"/>
   <digit value="1"/>
   <digit value="2"/>
   <digit nullFlavor="UNK"/>
</example>
```


This example shows Lead II of an EKG tracing, with origin calibrated at 0 μV and with a scale factor of 2,5 μV. The last measurement failed (to show example of nullFlavor).

7.9.7 HIST (history)

7.9.7.1 Description

Specializes LIST.

Parameter: T : ANY.

A collection that set of items in historical order.

7.9.7.2 ISO/IEC 11404 syntax

```
type HIST (T : ANY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  item : Set(T)
)
```

NOTE The historical information pertains to the correctness of the information rather than the system's knowledge of the actual information. For further information, see 7.3.2.

7.9.7.3 Invariants

— all items in the list shall have at either validTimeLow or validTimeHigh non-null;

— the validTime periods on the list shall not overlap, and the items shall be ordered in ascending chronological order.

OCL for Invariants:

```
inv "validTime required": item->forAll(i |
      i.validTimeLow.oclIsDefined or
      i.validTimeHigh.oclIsDefined)
```

7.9.7.4 Examples

```
<example xsi:type="HIST_TEL">
  <item nullFlavor="UNK" use="WP H" validTimeHigh="199206"/>
  <item value="tel:+15552225543" use="H" validTimeLow="199206"
     validTimeHigh="199207"/>
  <item value="tel:+15556667777" use="H" validTimeLow="199207"/>
</example>
```

This specifies a known history of home telephone numbers.

7.9.8 BAG (bag)

7.9.8.1 Description

Specializes COLL.

Parameter: T : ANY.

An unordered collection of values, where each value can be contained more than once in the collection.

7.9.8.2 ISO/IEC 11404 syntax

```
type BAG (T : ANY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  item : Bag(T)
)
```

7.9.8.3 Attributes

7.9.8.3.1 item : Bag(T): The contents of the Bag.

This is an example of the primitive type wrapping pattern. See 6.3 for more details.

7.9.8.4 Equality

Two bags are equal if (and only if) they are both empty, or if they contain the same items, with the same number of each item present.

NOTE The determination of element content is based on the same semantic equals as defined in this International Standard, so it is possible that a BAG(CD) can be equal to BAG(CO), for instance.

7.9.8.5 Invariants

(none)

NOTE Rules about updateMode or history are applied to item where BAG(T) is used.

7.9.8.6 Operations

7.9.8.6.1 union(bag : BAG(T)) : BAG(T): The union of this and bag. If a value is contained multiple times, the total count of the value's occurance in the result of the union will the sum of the occurances in this and bag. This is an alias for including.

7.9.8.6.2 union(set : DSET(T)) : BAG(T): The union of this and set. If a value is contained in both the bag and the set, the total count of the value's occurance in the result of the union will be one greater than the number of occurances in this. This is an alias for including.

7.9.8.6.3 intersection(bag : BAG(T)) : BAG(T): The intersection of this and bag.

7.9.8.6.4 intersection(set : DSET(T)) : DSET(T): The intersection of this and set.

7.9.8.6.5 including(object : T) : BAG(T): The bag containing all elements of this plus object. If object is already in this, it will occur another extra time in the result.

7.9.8.6.6 including(coll : COLL(T)) : BAG(T): The bag containing all elements of this plus any elements in coll. This is an alias for union.

7.9.8.6.7 excluding(object : T) : BAG(T): The bag containing all elements of this apart from all occurrences of object.

7.9.8.6.8 excluding(coll : COLL(T)) : BAG(T): The bag containing all elements of this apart from any objects that are found in coll.

7.9.8.6.9 asList() : LIST(T): A sequence that contains all the elements from this in an undefined order.

7.9.8.6.10 asSet() : DSET(T): The set containing all the elements from this, with duplicates removed.

7.9.8.7 Examples

```
<example xsi:type="BAG_TEL">
  <item value="tel:+15556667777" use="H"/>
  <item nullFlavor="UNK" use="WP"/>
</example>
```

A bag of telephone numbers, with a known home telephone number and an unknown work number.

```
<example xsi:type="BAG_TEL">
  <item nullFlavor="UNK" use="WP"/>
  <item value="tel:+15556667777" use="H"/>
</example>
```

This is not equal to the previous bag of telephone numbers; although order is not important, the work telephone number has a nullFlavor and equality cannot be evaluated.

7.10 Continuous set datatypes

7.10.1 Overview

These datatypes provide support for collections of data (see Figure 9).

7.10.2 QSET (continuous set)

7.10.2.1 Description

Abstract; specializes ANY.

Parameter: T : QTY.

An unordered set of distinct values that are quantities.

Any ordered type can be the basis of a QSET; it does not matter whether the base type is discrete or continuous. If the base datatype is only partially ordered, all elements of the QSET shall be elements of a totally ordered subset of the partially ordered datatype (for example, PQ is only ordered when the units are consistent. Every value in a QSET(PQ) shall have the same canonical unit).

QSET is an abstract type. A working QSET is specified as an expression tree built using a combination of operator (QSI, QSD, QSU, QSP) and component types (QSC, QSS and IVL; and for TS, PIVL and EIVL).

QSETs shall not contain null or nullFlavored values as members of the set.

7.10.2.2 ISO/IEC 11404 syntax

```
type QSET (T : ANY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  originalText : ED.TEXT
)
```

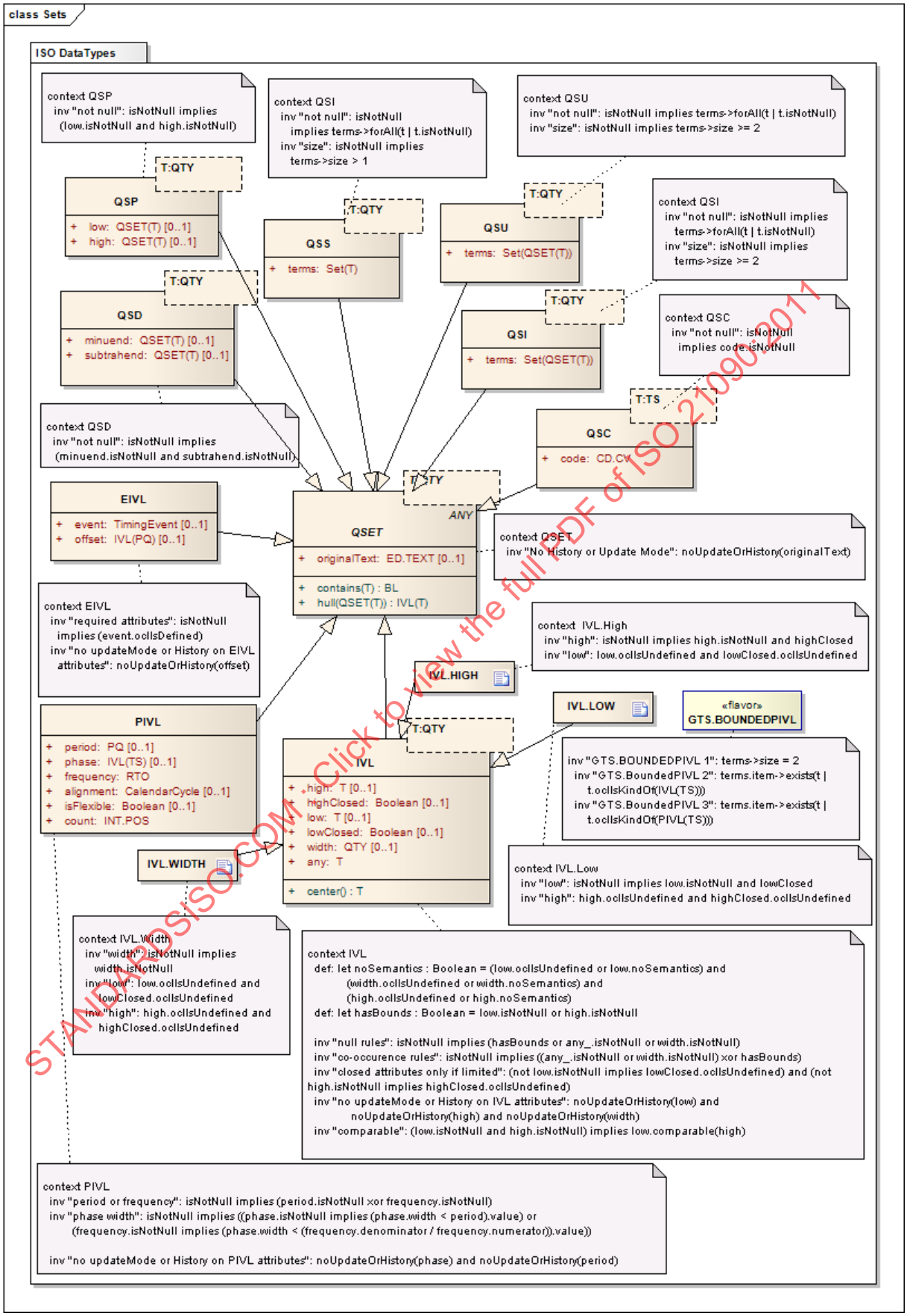

Figure 9 — Continuous set datatypes

7.10.2.3 Attributes

7.10.2.3.1 originalText : ED.TEXT: The text representation from which the QSET was encoded, if such a representation is the source of the QSET.

Original text can be used in a structured user interface to capture what the user saw as a representation of the set on the data input screen, or in a situation where the user dictates or directly enters text, it is the text entered or uttered by the user.

It is valid to use a QSET derived datatype to store only the text that the user entered or uttered. In this situation, original text exists without a valid value. The originalText is not a substitute for a valid value. If the actual content of the *QSET* is not valid, then the *QSET* shall be nullFlavored, irrespective of whether originalText has a value or not.

The original text shall be an excerpt of the relevant information in the original sources, rather than a pointer or exact reproduction. Thus, the original text shall be represented in plain text form. In specific circumstances, when clearly described the context of use, the originalText may be a reference to some other text artefact for which the resolution scope is clearly described.

NOTE The details of the link in the originalText.reference between different artifacts of medical information (e.g. document and coded result) is outside the scope of this International Standard and can be further proscribed in specifications that use this International Standard.

7.10.2.4 Equality

The notional equality determination for QSET and all its descendants, except IVL, is based on set membership: two QSETs are equal if they contain the same members. However, QSETs are used to build expression trees that can become quite complex. It is not feasible to determine whether two different QSET expression trees describe the same set of elements, so the determination for whether two QSETs are equal is the default equality test defined in ANY.

This equality test applies to all QSET specializations except for IVL, and is not specified for the other specializations.

7.10.2.5 Invariants

OCL for invariants:

```
  inv "No History or Update Mode":
noUpdateOrHistory(originalText)
```

7.10.2.6 Operations

7.10.2.6.1 contains (x: T): BL: True if the QSET contains the value x.

7.10.2.6.2 hull (x: QSET(T)): IVL(T) : The convex hull of this set with x, which is the smallest interval that is a superset of this and x. See Figure 10.

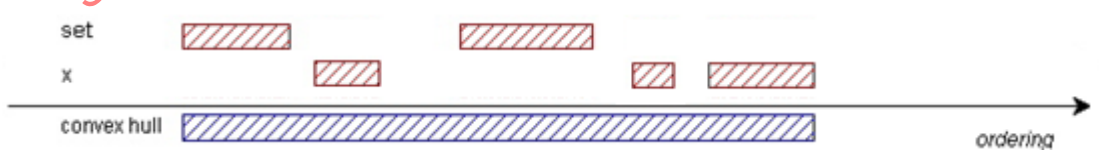

Figure 10 — Convex hull

NOTE The convex hull of a *QSET* can less formally be called the "outer bound interval". Thus, the convex hull of a *QSET* describes the absolute beginning and end of a schedule. For some set specifications, where there is infinite repetition [e.g. a PIVL(TS)] the convex hull has infinite bounds. The term "schedule" is used throughout this subclause in its general sense, that of an organized series of values. The more common meaning of the term "schedule", that of a time-based plan of events, is exactly a *QSET*(TS).

7.10.2.7 Examples

7.10.2.7.1 QSET(TS)

The type QSET(TS) is also known as GTS: general timing specification.

The first example specifies every other Tuesday in the season from the (USA holidays) Memorial Day to Labor (Labour) Day in the years 2002 and 2003. This is built as an expression of the intersection between three sets:

— every other Tuesday;

— the years 2002 and 2003;

— the season between Memorial Day and Labor (Labour) Day.

```
<example xsi:type="QSI_TS">
  <!-- intersection, because it is a QSI -->

  <!-- every other Tuesday -->
  <term xsi:type='PIVL_TS' alignment='DW'>
    <phase lowClosed='true' highClosed='false'>
      <low value='20001202'/>
      <high value='20001203'/>
    </phase>
    <period value='2' unit='wk'/>
  </term>

  <!-- 2002 and 2003 -->
  <term xsi:type='IVL_TS' lowClosed='true' highClosed='false'>
    <low value='20020101'/>
    <high value='20040101'/>
  </term>

  <!-- season between Memorial Day and Labor Day -->
  <!-- periodic hull between Memorial day and Labor Day -->
  <term xsi:type='QSP_TS'>
    <low xsi:type="QSI_TS">
      <!-- memorial day: intersection of last week of May and mondays -->
      <term xsi:type='PIVL_TS'>
        <phase highClosed='false'>
          <low value='19870525'/>
          <high value='19870601'/>
        </phase>
        <period value='1' unit='a'/>
      </term>
      <term xsi:type='PIVL_TS'>
        <phase highClosed='false'>
          <low value='19870105'/>
          <high value='19870106'/>
        </phase>
        <period value='1' unit='wk'/>
      </term>
    </low>
    <high xsi:type="QSI_TS">
      <!-- labor day :  intersection of first week of Sept and mondays -->
      <term xsi:type='PIVL_TS'>
        <phase highClosed='false'>
          <low value='19870901'/>
          <high value='19870908'/>
        </phase>
        <period value='1' unit='a'/>
      </term>
      <term xsi:type='PIVL_TS'>
```

```
        <phase highClosed='false'>
          <low value='19870105'/>
          <high value='19870106'/>
        </phase>
        <period value='1' unit='wk'/>
      </term>
    </high>
  </term>
</example>
```

7.10.3 QSU (QSET Union)

7.10.3.1 Description

Specializes QSET.

Specifies a QSET as a union of other sets.

7.10.3.2 ISO/IEC 11404 syntax

```
type QSU (T : ANY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  originalText : ED.TEXT,
  terms : Set(QSET(T))
)
```

7.10.3.3 Attributes

7.10.3.3.1 terms : Set(QSET(T)): A list of other QSETs that are involved in the union.

7.10.3.4 Invariants

— a nonNull QSU may only contain nonNull QSETs;

— at least two sets shall be specified.

OCL for invariants:

```
inv "not null": isNotNull implies terms->forAll(t |
t.isNotNull)
inv "size": isNotNull implies terms->size >= 2
```

7.10.4 QSI (QSET intersection)

7.10.4.1 Description

Specializes QSET.

Specifies a QSET as an intersection of other sets.

7.10.4.2 ISO/IEC 11404 syntax

```
type QSI (T : ANY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  originalText : ED.TEXT,
  terms : Set(QSET(T))
)
```

7.10.4.3 Attributes

7.10.4.3.1 terms : Set(QSET(T)): A list of other QSETs that are involved in the intersection.

7.10.4.4 Invariants

— a nonNull QSI may only contain nonNull QSETs;

— at least 2 sets shall be specified.

OCL for invariants:

```
 inv "not null": isNotNull implies terms->forAll(t |
t.isNotNull)
 inv "size": isNotNull implies terms->size >= 2
```

7.10.5 QSD (QSET difference)

7.10.5.1 Description

Specializes QSET.

Specifies a QSET as the difference between two sets.

7.10.5.2 ISO/IEC 11404 syntax

```
type QSU (T : ANY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  originalText : ED.TEXT,
  minuend : QSET(T)
  subtrahend : QSET(T)
)
```

The difference is the second set subtracted from the first.

7.10.5.3 Attributes

7.10.5.3.1 minuend : QSET(T): The set from which the second set is subtracted.

7.10.5.3.2 subtrahend: QSET(T): The set that is subtracted from the first set.

7.10.5.4 Invariants

— a nonNull QSD may only contain nonNull QSETs.

OCL for invariants:

```
 inv inv "not null": isNotNull implies (minuend.isNotNull and
subtrahend.isNotNull)
```

7.10.6 QSP (QSET periodic hull)

7.10.6.1 Description

Specializes QSET.

Specifies a QSET as the periodic hull between two sets as shown in Figure 11.

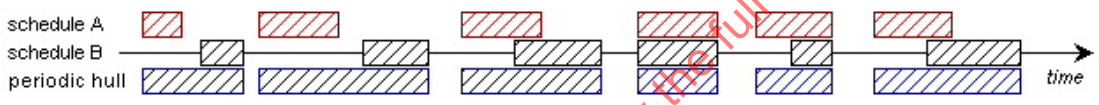

Figure 11 — Periodic hull

A periodic hull may be generated by comparing two sets that interleave. For QSET values **A** and **B** to interleave, the occurrence intervals of both groups can be arranged in pairs of corresponding occurrence intervals. It must further hold that for all corresponding occurrence intervals a $\subseteq$ A and b $\subseteq$ B, A starts before B starts (or at the same time) and B ends after A ends (or at the same time).

The interleaves-relation holds when two schedules have the same average frequency, and when the second schedule never "outpaces" the first schedule. That is, no occurrence interval in the second schedule may start before its corresponding occurrence interval in the first schedule.

7.10.6.2 ISO/IEC 11404 syntax

```
type QSP (T : ANY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  originalText : ED.TEXT,
  low : QSET(T)
  high : QSET(T)
)
```

7.10.6.3 Attributes

7.10.6.3.1 low : QSET(T): The set used as the basis for the periodic hull operation.

7.10.6.3.2 high: QSET(T): The set that is used as the parameter for the periodic hull operation.

7.10.6.4 Invariants

— a nonNull QSP may only contain nonNull QSETs.

OCL for invariants:

```
inv inv "not null": isNotNull implies (low.isNotNull and
high.isNotNull)
```

7.10.7 QSS (QSET Set)

7.10.7.1 Description

Specializes QSET.

Specifies a QSET as an enumeration of simple values. This is a shortcut form for specifying the same values as singleton intervals.

7.10.7.2 ISO/IEC 11404 syntax

```
type QSS (T : QTY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  originalText : ED.TEXT,
  terms : Set(T)
)
```

7.10.7.3 Attributes

7.10.7.3.1 terms : Set(T): a list of values that are in the set. The set is actually constructed as the union of the intervals implied by the precision implicit in the definition of T. For some types of QTY, this is either trivial (INT) or ambiguous (RTO) and QSS does not really make sense for these type. QSS is a useful type for TS in particular.

7.10.7.4 Invariants

— a nonNull QSS may only contain nonNull values;

— at least 1 value shall be specified.

OCL for invariants:

```
inv "not null": isNotNull implies terms->forAll(t |
t.isNotNull)
inv "size": isNotNull implies terms->size >= 1
```

7.10.7.5 Examples

```
<example xsi:type='QSS_TS'>
   <term value='20071101'/>
   <term value='20071106'/>
</example>
```

The union of the intervals that cover November 1, 2007 and November 6, 2007.

7.10.8 QSC (coded QSET)

7.10.8.1 Description

Specializes QSET.

Specifies a QSET as an coded value that describes a predefined QSET(TS).

7.10.8.2 ISO/IEC 11404 syntax

```
type QSC (T : ANY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  originalText : ED.TEXT,
  code : CD.CV
)
```

7.10.8.3 Attributes

7.10.8.3.1 code : CD.CV: a predefined code that fully and unambiguously describes a set of times.

The possible set of codes that are allowed for use in this attribute should be described in conformance statements. HL7 defines the set of codes described below in GTSAbbrevation, and all information processing entities claiming direct conformance with this standard shall support the codes AM, PM, BID, TID, QID, JB and JE if this type is supported.

Code System GTSAbbreviation. OID: 2.16.840.1.113883.5.1022 (Required Codes, OID for this value set 2.16.840.1.113883.1.11.10720)		
1	AM	Every morning at institution-specified times.
1	PM	Every afternoon at institution-specified times.
1	BID	Two times a day at institution-specified time.
1	TID	Three times a day at institution-specified time.
1	QID	Four times a day at institution-specified time.
1	JB	Regular business days (Monday to Friday excluding holidays).
1	JE	Regular weekends (Saturday and Sunday excluding holidays).

In addition to the core codes described above, HL7 has also defined additional codes for the designated official or unofficial holidays of source countries:

Code System GTSAbbreviation. OID: 2.16.840.1.113883.5.1022 (Additional Holiday Codes, OID for this value set 2.16.840.1.113883.1.11.10725)		
1	JH	Holidays
2	JHCHR	Christian Holidays (Roman/Gregorian "Western" Tradition.)
3	JHCHRXME	Christmas Eve (December 24)
3	JHCHRXMS	Christmas Day (December 25)
3	JHCHRNEW	New Year's Day (January 1)
3	JHCHREAS	Easter Sunday. The Easter date is a rather complex calculation based on astronomical tables describing full moon dates. Details can be found at [http://www.assa.org.au/edm.html, and http://aa.usno.navy.mil/AA/faq/docs/easter.html]. The Christian Orthodox Holidays are based on the Julian calendar.
3	JHCHRGFR	Good Friday is the Friday immediately before Easter Sunday
3	JHCHRPEN	Pentecost Sunday is seven weeks after Easter (the 50th day of Easter).
2	JHNUS	United States National Holidays (public holidays for federal employees established by USA Federal law 5 U.S.C. 6103.)
3	JHNUSMLK	Dr. Martin Luther King, Jr. Day, the third Monday in January.
3	JHNUSPRE	Washington's Birthday (Presidential Day) the third Monday in February
3	JHNUSMEM	Memorial Day, the last Monday in May
3	JHNUSMEM5	Friday before Memorial Day Weekend
3	JHNUSMEM6	Saturday of Memorial Day Weekend
3	JHNUSIND	Independence Day (fourth of July)
3	JHNUSIND5	Alternative Friday before the fourth of July Weekend [5 U.S.C. 6103(b)].
3	JHNUSIND1	Alternative Monday after the fourth of July Weekend [5 U.S.C. 6103(b)].
3	JHNUSLBR	Labor (Labour) Day, the first Monday in September.
3	JHNUSCLM	Columbus Day, the second Monday in October.
3	JHNUSVET	Veteran's Day, November 11
3	JHNUSTKS	Thanksgiving Day, the fourth Thursday in November
3	JHNUSTKS5	Friday after Thanksgiving
2	JHNNL	The Netherlands National Holidays
3	JHNNLQD	Queen's day (April 30)
3	JHNNLLD	Liberation day (May 5 every five years)
3	JHNNLSK	Sinterklaas (December 5)

NOTE 1 This table is not complete; neither does it include religious holidays other than Christian [of the Gregorian (Western) tradition] nor national holidays in countries other than the USA and the Netherlands. While other jurisdictions might define their own code systems, they are welcome to submit their codes to HL7 and or ISO for inclusion in this code system.

NOTE 2 Holidays are locale-specific. Exactly which religious holidays are subsumed under JH depends on the locale and other tradition. For global interoperability, using constructed QSET expressions is safer than named holidays. However, some holidays that depend on moon phases (e.g. Easter, Ramadan) or ad-hoc decree cannot be easily expressed in a QSET other than by using *QSC*.

NOTE 3 Information processing entities might define their own set of codes to be supported by creating an appropriate value set. The value set can be referenced in the QSC code if required.

7.10.8.4 Invariants

— a code shall be provided.

OCL for invariants:

```
inv "not null": isNotNull implies code.isNotNull
```

7.10.8.5 Examples

```
<example xsi:type='QSC_TS' >
  <code code="JHCHRXMS" codeSystem="2.16.840.1.113883.5.1022"/>
</example>
```

All Christmas days.

7.10.9 IVL (interval)

7.10.9.1 Description

Specializes QSET.

Parameter: T : QTY.

A set of consecutive values of an ordered base datatype.

Any ordered type can be the basis of an IVL; it does not matter whether the base type is discrete or continuous. If the base datatype is only partially ordered, all elements of the IVL shall be elements of a totally ordered subset of the partially ordered datatype. For example PQ is considered ordered. However the ordering of PQs is only partial; a total order is only defined among comparable quantities (quantities of the same physical dimension). While *IVL*s between 2 m and 4 m exist, there is no *IVL* between 2 m and 4 s.

7.10.9.2 ISO/IEC 11404 syntax

```
type IVL (T : ANY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  originalText : ED.TEXT,
  low : T,
  lowClosed : boolean,
  high : T,
  highClosed : boolean,
  width : QTY,
  any : T
)
```

7.10.9.3 Attributes

7.10.9.3.1 low : T: This is the low limit. If the low limit is not known, a nullFlavor may be specified.

The low limit shall not be positive infinity.

7.10.9.3.2 lowClosed : Boolean: Whether low is included in the IVL (is closed) or excluded from the IVL (is open).

7.10.9.3.3 high : T: This is the high limit. If the high limit is not known, a nullFlavor may be specified.

The high limit shall not be negative infinity, and shall be higher than the low limit if one exists.

7.10.9.3.4 highClosed : Boolean: Whether high is included in the IVL (is closed) or excluded from the IVL (is open).

7.10.9.3.5 width : QTY: The difference between high and low boundary. Width is used when the size of the Interval is known, but the actual start and end points are not known. The actual type QTY will be dictated by the type of T.

7.10.9.3.6 any : T: Specifies that some particular value lies within the interval.

This should be used when it is not known when something starts, or ends, but it is known that it was happening at a given time. This is relatively common for observations (i.e. of disease processes), procedure, and scheduling. In these cases, neither high nor low is known, though the width may also be known.

7.10.9.4 Equality

Unlike other QSET specializations, IVL equality is determined based on set membership. Two IVL values are equal if they contain the same members.

NOTE 1 For IVLs, there are two special cases. Highs are considered equal if they are both positive infinity, and lows are considered equal if they are both negative infinity.

NOTE 2 If two intervals have the same width and the bounds are not known, they are not considered equal.

NOTE 3 The same applies where the interval is known by a contained value (any): such intervals are never considered equal.

NOTE 4 Because equality is determined by set membership, it is possible for DSET(INT) and IVL(INT) to be equal. For example, the DSET(INT) 2,3,4 is equal to the IVL(INT) 2..4.

7.10.9.5 Invariants

- either the IVL is nullFlavored, has a width, ANY, or has (low and/or high). (ANY and/or width) and (low and/or high) cannot be mixed;
- lowClosed and highClosed can only be used if low or high are used;
- low and high shall be comparable.

OCL for Invariants:

```
  def: let hasBounds : Boolean = low.isNotNull or high.isNotNull
  def: let noSemantics : Boolean = (low.oclIsUndefined or
       low.noSemantics) and (width.oclIsUndefined or
       width.noSemantics) and (high.oclIsUndefined or
       high.noSemantics)

  inv "null rules": isNonNull implies (hasBounds or any_.isNotNull
       or width.isNotNull)
  inv "co-occurence rules": isNotNull implies ((any_.isNotNull
       or width.isNotNull) xor hasBounds)
  inv "closed attributes only if limited":
     (not low.isNotNull implies lowClosed.oclIsUndefined) and
     (not high.isNotNull implies highClosed.oclIsUndefined)
  inv "no updateMode or History on IVL attributes":
              noUpdateOrHistory(low) and
              noUpdateOrHistory(high) and noUpdateOrHistory(width)
  inv "comparable": (low.isNotNull and high.IsNotNull) implies
              low.comparable(high)
```

7.10.9.6 Examples

7.10.9.6.1 Integer interval

```
<example xsi:type='IVL_INT'>
   <low value='2'/>
   <high value='4'/>
</example>
```

A simple interval of INT between 2 and 4. This is exactly the same set of values as specifying the DSET (2,3,4)

7.10.9.6.2 Physical quantity interval

```
<example xsi:type='IVL_PQ' lowClosed='true' highClosed='false'>
   <low value='2.8' unit='m'/>
   <high value='4.6' unit='m'/>
</example>
```

An interval of PQ between 2,8 m, inclusive, and 4,6 m, exclusive.

7.10.9.6.3 Timestamp interval

```
<example xsi:type='IVL_TS'>
   <low value='200012041000'/>
   <high value='200012041030'/>
</example>
```

An interval of TS on December 4, 2000, between 10 a.m. and 10:30 a.m.

7.10.9.6.4 Operation record

```
<example xsi:type='IVL_TS'>
   <width xsi:type='PQ' value='2' unit='h'/>
   <any value='200012041000'/>
</example>
```

The operation took 2 h, and was occurring at 10 a.m. on December 4, 2000. Width requires an xsi:type since its type is abstract (QTY)

7.10.10 IVL.LOW

7.10.10.1 Description

A flavor that constrains IVL.

IVL.LOW constrains IVL so that low is provided and lowClosed is true. All other properties are prohibited.

7.10.10.2 Invariants

- low and lowClosed shall be populated;
- high and highClosed shall be null.

OCL for Invariants:

```
inv "low": isNotNull implies low.isNotNull and lowClosed
inv "high": high.oclIsUndefined and highClosed.oclIsUndefined
```

7.10.11 IVL.HIGH

7.10.11.1 Description

A flavor that constrains IVL.

IVL.HIGH constrains IVL so that high is provided and highClosed is true. All other properties are prohibited.

7.10.11.2 Invariants

- low and lowClosed shall be null;
- high and highClosed shall be populated.

OCL for Invariants:

```
inv "high": isNotNull implies high.isNotNull and highClosed
inv "low": low.oclIsUndefined and lowClosed.oclIsUndefined
```

7.10.12 IVL.WIDTH

7.10.12.1 Description

A flavor that constrains IVL.

IVL.WIDTH constrains IVL so that width is mandatory and low, lowClosed, high and highClosed are prohibited.

7.10.12.2 Invariants

- width shall be populated;
- low and lowClosed shall be null;
- high and highClosed shall be null.

OCL for Invariants:

```
inv "width": isNotNull implies width.isNotNull
inv "low": low.oclIsUndefined and lowClosed.oclIsUndefined
inv "high": high.oclIsUndefined and highClosed.oclIsUndefined
```

7.10.13 PIVL (PeriodicInterval)

7.10.13.1 Description

Specializes QSET.

An interval of time that recurs periodically. PIVL has two properties, phase and period/frequency. phase specifies the "interval prototype" that is repeated on the period/frequency.

7.10.13.2 ISO/IEC 11404 syntax

```
type PIVL (T : ANY) = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  originalText : ED.TEXT,
  phase : IVL(TS),
  period : PQ,
  frequency : RTO,
  alignment : CalendarCycle,
  isFlexible : boolean,
  count : INT.POS
)
```

7.10.13.3 Attributes

7.10.13.3.1 phase : IVL(TS): A prototype of the repeating interval, specifying the duration of each occurrence and anchors the PIVL sequence at a certain point in time. phase also marks the anchor point in time for the entire series of periodically recurring intervals. If count is null or nullFlavored, the recurrence of a PIVL has no beginning or ending, but is infinite in both future and past.

The width of the phase shall be less than or equal to the period.

7.10.13.3.2 period : PQ: A time duration specified as a reciprocal measure of the frequency at which the PIVL repeats.

7.10.13.3.3 frequency : RTO: The number of times the *PIVL* repeats (numerator) within a specified time period (denominator). The numerator is an integer, and the denominator is a PQ.TIME.

Only one of period and frequency should be specified. The form chosen should be the form that most naturally conveys the idea to humans, i.e. every 10 min (period) or twice a day (frequency).

7.10.13.3.4 alignment : CalendarCycle: If and how the repetitions are aligned to the cycles of the underlying calendar (e.g. to distinguish every 30 d from "the 5th of every month".) A non-aligned PIVL recurs independently from the calendar. An aligned PIVL is synchronized with the calendar.

If populated, the value of this attribute shall be taken from the HL7 CalendarCycle code system. The current values are:

CalendarCycle Enumeration. OID: 2.16.840.1.113883.5.9			
1	CY	year	
1	MY	month of the year	
1	CM	month (continuous)	
1	CW	week (continuous)	
1	WM	week of the month	
1	WY	week of the year	
1	DM	day of the month	
1	CD	day (continuous)	
1	DY	day of the year	
1	DW	day of the week (begins with monday)	
1	HD	hour of the day	
1	CH	hour (continuous)	
1	NH	minute of the hour	
1	CN	minute (continuous)	
1	SN	second of the minute	
1	CS	second (continuous)	

ISO/IEC 11404 Syntax for alignment attribute

```
type CalendarCycle = enumeration (CY, MY, CM, CW, WY, DM, CD, DY, DW,
HD, CH, NH, CN, SN, CS)
```

7.10.13.3.5 isFlexible : Boolean: Indicates whether the exact timing is up to the party executing the schedule e.g. to distinguish "every 8 h" from "3 times a day".

NOTE This is sometimes referred to as "institution specified timing".

7.10.13.3.6 count : INT.POS: The number of times the period repeats in total. If count is null or nullFlavored, then the period repeats indefinitely both before and after the anchor implicit in the phase.

7.10.13.4 Invariants

— if PIVL is not nullFlavored, only one of period and frequency may be specified;

— the width of the phase shall be less or equal to the period.

OCL for invariants:

```
inv "no updateMode or History on PIVL attributes":
     noUpdateOrHistory(phase)and noUpdateOrHistory(period)
inv "no updateMode or History on PIVL attributes":
     noUpdateOrHistory(phase) and noUpdateOrHistory(period)
inv "phase width": isNotNull implies
  ((phase.isNotNull implies phase.width < x.period) or
  (frequency.isNotNull implies phase.width <
         (frequency.denominator / frequency.numerator)))
```

7.10.13.5 Examples

7.10.13.5.1 Twice a day

```
<example xsi:type='PIVL_TS' isFlexible='true'>
   <period value='12' unit='h'/>
</example>
```

Twice a day (BID). The actual time is at the discretion of the institution.

This can also be represented using the alternative representation by frequency:

```
<example xsi:type='PIVL_TS' isFlexible='true'>
   <frequency>
     <numerator xsi:type="INT" value='2'/>
     <denominator xsi:type="PQ" value="1" unit='d'/>
   </frequency>
</example>
```

This also represents twice a day (BID). In simple cases, such as twice a day, the two forms are easily interconvertable, and humans find either form acceptable. While it is always possible to convert between period and frequency, human readers have a strong preference for one form or another depending on the actual numbers:

```
<example xsi:type='PIVL_TS' isFlexible='true'>
   <frequency>
     <numerator xsi:type="INT" value='7'/>
     <denominator xsi:type="PQ" value="1" unit='d'/>
   </frequency>
</example>
```

This means to do something seven times a day. The period based reference to this is not so easy to read:

```
<example xsi:type='PIVL_TS' isFlexible='true'>
   <period value='3.428571428571428571428571428571' unit='h'/>
</example>
```

While this example may seem contrived, examples like this arise in clinical practice around the world.

7.10.13.5.2 Twice a day for ten minutes

```
<example xsi:type='PIVL_TS'>
   <phase>
      <width xsi:type="PQ" value='10' unit='min'/>
   </phase>
   <period value='12' unit='h'/>
</example>
```

Twice a day (every 12 h) for 10 min.

7.10.13.5.3 Every September

```
<example xsi:type='PIVL_TS' alignment='MY'>
   <phase highClosed='true' lowClosed='false'>
      <low value='198709'/>
      <high value='198710'/>
   </phase>
   <period value='1' unit='a'/> <!-- a means year in UCUM -->
</example>
```

This example is slightly more complex and shows the month of September that recurs every year (In 1987, this form is irrelevant since the periodic interval recurs every year past and future.)

7.10.13.5.4 Every other Saturday.

```
<example xsi:type='PIVL_TS' alignment='DW'>
   <phase highClosed='true' lowClosed='false'>
      <low value='20001202' />
      <high value='20001203' />
   </phase>
   <period value='2' unit='wk'/>
</example>
```

7.10.13.5.5 Every 4 h to 6 h.

```
<example xsi:type='PIVL_TS'>
   <period value='5' unit='h' uncertaintyType='U'>
      <uncertainty value='0.57735' unit='h'/>
   </period>
</example>
```

7.10.14 EIVL (Event-Related Periodic Interval of Time)

7.10.14.1 Description

Specializes QSET.

Specifies a periodic interval of time where the recurrence is based on activities of daily living or other important events that are time-related but not fully determined by time.

EXAMPLE "One hour after breakfast" specifies the beginning of the interval one hour after breakfast is finished. Breakfast is assumed to occur before lunch, but is not determined to occur at any specific time.

7.10.14.2 ISO/IEC 11404 syntax

```
type EIVL = class (
  validTimeLow : characterstring,
  validTimeHigh : characterstring,
  controlInformationRoot : characterstring,
  controlInformationExtension : characterstring,
  nullFlavor : NullFlavor,
  updateMode : UpdateMode,
  flavorId : Set(characterstring),
  originalText : ED.TEXT,
  event : TimingEvent,
  offset : IVL(PQ)
)
```

7.10.14.3 Attributes

7.10.14.3.1 event : TimingEvent: A code for a common (periodic) activity of daily living based on which the event-related periodic interval is specified. Events that qualify for being adopted in the domain of this attribute must satisfy the following requirements:

— the event commonly occurs on a regular basis;

— the event is being used for timing activities;

— the event is not entirely determined by time.

If these criteria are not met, the relationship of the event and its time must be communicated using structures outside the datatypes defined in this International Standard.

If populated, the value of this attribute shall be taken from the HL7 TimingEvent code system. The current values are:

TimingEvent Enumeration. OID: 2.16.840.1.113883.5.139			
1	HS	HS	The hour of sleep
1	WAKE	WAKE	upon waking
1	AC	AC	before a meal (from the latin *ante cibus*)
2	ACM	ACM	before breakfast (from the latin *ante cibus matutinus*)
2	ACD	ACD	before lunch (from the latin *ante cibus diurnus*)
2	ACV	ACV	before dinner (from the latin *ante cibus vespertinus*)
1	IC	IC	between meals (from the latin *inter cibus*)
2	ICM	ICM	between breakfast and lunch
2	ICD	ICD	between lunch and dinner
2	ICV	ICV	between dinner and the hour of sleep
1	PC	PC	after a meal (from the latin *post cibus*)
2	PCM	PCM	after breakfast (from the latin *post cibus matutinus*)
2	PCD	PCD	after lunch (from the latin *post cibus diurnus*)
2	PCV	PCV	after dinner (from the latin *post cibus vespertinus*)
1	C	C	meal (from the latin *cibus*)
2	CM	CM	breakfast (from the latin *cibus matutinus*)
2	CD	CD	lunch (from the latin *cibus diurnus*)
2	CV	CV	dinner (from the latin *cibus vespertinus*)

ISO/IEC 11404 Syntax for event attribute

```
type TimingEvent = enumeration (HS, WAKE, AC, ACM, ACD, ACV, IC, ICM, ICD,
ICV, PC, PCM, PCD, PCV, C, CM, CD, CV)
```

7.10.14.3.2 offset : IVL(PQ): An interval of elapsed time (duration, not absolute point in time) that marks the offsets for the beginning, width and end of the EIVL measured from the time each such event actually occurred.

EXAMPLE If the specification is "one hour before breakfast for 10 min", code is CM, IVL.low of offset is –1 h and the IVL.high of offset is –50 min.

The offset shall be null if the event code specifies "before", "after" or "between meals". The offset shall be nonNull if the EIVL is nonNull and the event code is C, CM, CD or CV. The offset may or may not be null or nullFlavored for the event codes HS and WAKE.

7.10.14.4 Invariants

— if EIVL is not nullFlavored, event shall be specified.

OCL for invariants:

```
inv "required attributes": isNotNull implies
(event.oclIsDefined)
inv "no updateMode or History on EIVL attributes":
noUpdateOrHistory(offset)
```


7.10.14.5 Examples

```
<example xsi:type='EIVL_TS' event='CM'>
   <offset>
      <low value='-1' unit='h'/>
      <high value='-50' unit='min'/>
   </offset>
</example>
```

One hour before breakfast for 10 min.

```
<example xsi:type='EIVL_TS' event='CV'>
   <offset>
      <low value='30' unit='min'/>
      <high value='30' unit='min'/>
   </offset>
</example>
```

Thirty minutes after dinner.

7.10.15 GTS.BOUNDEDPIVL

7.10.15.1 Description

A flavor that constrains QSI.

GTS.BOUNDEDPIVL constrains QSI(TS) so that it only allows an intersection of IVL(TS) and PIVL(TS).

7.10.15.2 Invariants

- there shall be two terms;
- one term shall be an IVL(TS);
- the other term shall be a PIVL(TS);
- the IVL width shall be null (i.e. either a low or high or both shall be provided).

OCL for Invariants:

```
inv "GTS.BOUNDEDPIVL 1": terms->size = 2
inv "GTS.BoundedPIVL 2": terms.item->exists(t |
    t.oclIsKindOf(IVL(TS)))
inv "GTS.BoundedPIVL 3": terms.item->exists(t |
    t.oclIsKindOf(PIVL(TS)))
```


7.11 Uncertainty datatypes

These datatypes provide support for uncertain values. The support provided here, along with the support provided for uncertainty on QTY, provides support for quantitative uncertainty, not with the medical kinds of uncertainty encountered in clinical practice such as “likely to be x”, or a differential diagnoses (see Figure 12).