INTERNATIONAL STANDARD

**ISO 13584-42**

Second edition
2010-12-15

# Industrial automation systems and integration — Parts library —

## Part 42: Description methodology: Methodology for structuring parts families

*Systèmes d'automatisation industrielle et intégration — Bibliothèque de composants —*

*Partie 42: Méthodologie descriptive: Méthodologie appliquée à la structuration des familles de pièces*



Reference number
ISO 13584-42:2010(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

## Figures



## Tables

# 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 13584-42 was prepared by Technical Committee ISO/TC 184, *Automation systems and integration*, Subcommittee SC 4, *Industrial data*.

This second edition of ISO 13584-42 constitutes a technical revision of ISO 13584-42:1998, which is provisionally retained in order to support continued use and maintenance of implementations based on it and to satisfy the normative references of other parts of ISO 13584. This second edition of ISO 13584-42 also incorporates the Technical Corrigendum ISO 13584-42:1998/Cor.1:2003.

ISO 13584 consists of the following parts, under the general title *Industrial automation systems and integration — Parts library:*

— *Part 1: Overview and fundamental principles*

— *Part 20: Logical resource: Logical model of expressions*

— *Part 24: Logical resource: Logical model of supplier library*

— *Part 25: Logical resource: Logical model of supplier library with aggregate values and explicit content*

— *Part 26: Logical resource: Information supplier identification*

— *Part 31: Implementation resources: Geometric programming interface*

— *Part 32: Implementation resources: OntoML: Product ontology markup language*

— *Part 35: Implementation resources: Spreadsheet interface for parts library* [Technical Specification]

— *Part 42: Description methodology: Methodology for structuring parts families*

— *Part 101: Geometrical view exchange protocol by parametric program*

— *Part 102: View exchange protocol by ISO 10303 conforming specification*

— *Part 501: Reference dictionary for measuring instruments — Registration procedure*

— *Part 511: Mechanical systems and components for general use — Reference dictionary for fasteners*

The structure of ISO 13584 is described in ISO 13584-1. The numbering of the parts of ISO 13584 reflects its structure:

— *Parts 10 to 19 specify the conceptual descriptions;*

— *Parts 20 to 29 specify the logical resources;*

— *Parts 30 to 39 specify the implementation resources;*

— *Parts 40 to 49 specify the description methodology;*

— *Parts 100 to 199 specify the view exchange protocols;*

— *Parts 500 to 599 specify the reference dictionaries.*

A complete list of parts of ISO 13584 is available from the following URL:

http://www.tc184-sc4.org/Titles/PLIB_Titles.htm

# Introduction

ISO 13584 is a collection of International Standards for the computer-interpretable representation and exchange of parts library data. The objective is to provide a neutral mechanism capable of transferring parts library data, independent of any application that is using a parts library data system. The nature of this description makes it suitable not only for the exchange of files containing parts, but also as a basis for implementing and sharing databases of parts library data.

ISO 13584 is organized as a series of parts, each published separately. The parts of ISO 13854 fall into one of the following series: conceptual descriptions, logical resources, implementation resources, description methodology, view exchange protocol, and reference dictionaries. The series are described in ISO 13584-1. This part of ISO 13584 is a part of the description methodology series.

This part of ISO 13584 provides rules and guidelines for standardization committees and for other information suppliers to create product ontologies. These product ontologies consist of hierarchies of characterization classes of parts built according to a common methodology intended to enable multi-supplier consistency. These rules pertain to the following: the method for grouping parts into characterization classes of parts to form a hierarchy; the method for associating part properties to characterization classes of parts, the dictionary elements that describe the classes and properties of parts.

This part of ISO 13584 refers as a normative reference to the data model that specifies the exchange of dictionary data. This EXPRESS specification was developed as a common model for ISO 13584 and IEC 61360, and is intended to be published as IEC 61360-2. For convenience, this common model is provided in this part of ISO 13584 as an informative annex that duplicates the normative content of IEC 61360-2. This part of ISO 13584 also provides the mapping of the concepts described here onto the common model. To understand Annex F, which contains a description of this model, knowledge of the EXPRESS language is required. The EXPRESS language is defined in ISO 10303-11:1994. No particular knowledge is required to understand the normative clauses of this part of ISO 13584.

This second edition of this part of ISO 13854 introduces the following modelling capabilities:

— the capability to model constraints on properties by restricting their domain of values;

— the capability to model and distinguish characterization classes and categorization classes;

— the capability to model aggregation and composition using a single resource mechanism;

— the capability to describe strings that carry external references;

— the capability to connect classes that belong to different class hierarchies.

This second edition of this part of ISO 13584 has removed the following:

— the capability to specialize item classes as feature classes, component classes or material classes.

NOTE The following changes ensure that a dictionary conforming with the first edition of this part of ISO 13584 conforms to this edition: (1) replace **feature_class**, **component_class** and **material_class** by **item_class** throughout the reference dictionary; (2) add to each new **item_class** class the **instance_sharable** attribute, the value of which being true for **component_class** and **material_class**, and false for **feature_class**; (3) add the places of a number of additional attributes.

# Industrial automation systems and integration — Parts library —

## Part 42:
## Description methodology: Methodology for structuring parts families

**IMPORTANT This part of ISO 13584 provides a specification intended to be implemented in software. Incompatibilities may result in machine-to-machine communication in the case of software developed on the basis of translations of this part of ISO 13584 into languages other than the official ISO languages. It is accordingly strongly recommended that any implementations be developed only on the basis of the texts in the official ISO languages.**



## 1 Scope

This part of ISO 13584 specifies the principles to be used for defining characterization classes of parts and properties of parts which provide for characterizing a part independently of any particular supplier-defined identification.

The rules and guidelines provided in this part of ISO 13584 are mandatory for the standardization committees responsible for creating standardized characterization hierarchies.

The use of these rules by suppliers and users is recommended as a methodology for building their own hierarchies.

The following are within the scope of this part of ISO 13584:

— the rules to group parts into leaf characterization classes of parts and non-leaf characterization classes of parts;

— the rules for the choice of the appropriate properties to be associated with the characterization classes of parts;

— the attributes to be provided by information suppliers to describe the characterization classes and properties of parts;

— the mechanisms for connecting characterization classes of parts to classification systems;

— the mechanisms for connecting characterization classes belonging to different characterization hierarchies;

— the specifications of those entities and attributes in the EXPRESS information model that provide for the exchange of such dictionary data.

— the description of any other object than part that can be characterized by a class belonging and a set of property-value pairs and to which the whole methodology defined in this part of ISO 13584 applies.

EXAMPLE Description by means of a dictionary compliant with this part of ISO 13584 can be used for describing any kind of products, as defined in Clause 3.

NOTE 1 The complete EXPRESS information model for the exchange of dictionary data, known as the common ISO13584/IEC61360 dictionary model, is defined in ISO 13584-25. Several levels of allowed implementations for the

common ISO13584/IEC61360 dictionary model, known as conformance classes, are also defined in ISO 13584-25. Conformance class 1 consists of the various schemes documented in this part of ISO 13584 (that duplicate information contained in IEC 61360-2), as well as the ISO13584_IEC61360_dictionary_aggregate_extension_schema documented in ISO 13584-25 (duplicated in IEC 61360-5 ). More advanced conformance classes, identified as conformance classes 2, 3 and 4, are documented in ISO 13584-25.

The following are outside the scope of this part of ISO 13584:

— properties of which values have an aggregate structure;

NOTE 2 An EXPRESS information model for the exchange of properties of which values have an aggregate structure is defined in ISO 13584-25.

— the description of the parts themselves;

— the descriptions of the functional models that can refer to some class of parts;

— the description of tables, program libraries and documents that can refer to some class of parts;

NOTE 3 EXPRESS resource constructs for the exchange of these information elements are defined in ISO 13584-24:2003.

— the description of the systems intended to manage parts libraries.

The structure of the information and the methodology defined in the ISO 13584 standard series enable the following:

— integration in the same data repository of different parts libraries originating from different information suppliers with uniform access mechanism provided by a dictionary;

— referencing another supplier library assumed to be available on the receiving system;

— referencing a standardized characterization hierarchy when such a hierarchy exists;

— definition by an end-user of a local categorization or search hierarchy, and the mapping of these hierarchies onto the supplier libraries available on its system.

## 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 8601, *Data elements and interchange formats — Information interchange — Representations of dates and times*

ISO 10303-11:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual*

IEC 61360-2:—[1)], *Standard data element types with associated classification scheme for electric components — Part 2: EXPRESS dictionary schema*

1) To be published. (Revision of IEC 61360-2:2004)

## 3 Terms and definitions

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

**3.1**
**abstract class**
class of which all members are also members of one of its subclasses

NOTE 1 Abstract classes are used when it is needed to group different kinds of objects in a class of a class inclusion hierarchy.

NOTE 2 In the common ISO13584/IEC61360 dictionary model, both abstract categorization classes and abstract characterization classes can be defined. The fact of being abstract is only a conceptual characteristic of a class. This characteristic is not explicitly represented in the model.

NOTE 3 Through inheritance, abstract characterization class allows to share, for example, some visible properties between different subclasses that correspond to different kinds of items.

**3.2**
**applicable property of a class**
applicable property necessarily possessed by each part that is member of a characterization class

NOTE 1 Each part that is member of a characterization class possesses an aspect corresponding to each applicable property of this characterization class.

NOTE 2 The above definition is conceptual, there is no requirement that all the applicable properties of a class should be used for describing each part of this class at the data model level.

NOTE 3 All the applicable properties of a superclass are also applicable properties for the subclasses of this superclass.

NOTE 4 Only properties defined or inherited as visible and imported properties of a class may be applicable properties.

NOTE 5 To facilitate integration of component libraries and electronic catalogues based on ISO 13584-24:2003 and ISO 13584-25, these parts of ISO 13584 request that only properties that are applicable to a class be used to characterize their instances in component libraries and electronic catalogues.

**3.3**
**attribute**
data element for the computer-sensible description of a property, a relation or a class

NOTE An attribute describes only one single detail of a property, of a class or of a relation.

EXAMPLE The name of a property, the code of a class, the measure unit in which values of a property are provided.

**3.4**
**basic semantic unit**
entity that provides an absolute and universally unique identification of a certain object of the application domain that is represented as a dictionary element

EXAMPLE 1 A dictionary compliant with this part of ISO 13584 provides for the identification of classes, properties, information sources and datatypes.

EXAMPLE 2 A dictionary compliant with ISO 13584-24:2003 provides for the identification of classes, properties, information sources, datatypes, tables, documents and program libraries.

EXAMPLE 3 In ISO 13584-511 , the class of the hexagon head bolts is identified by a BSU, the property thread tolerance grade is also identified by a BSU.

NOTE The content of a basic semantic unit may also be represented as an IRDI.

**3.5**
**characteristic of a product**
**product characteristic**
invariable *property*, characteristic of a *product*, whose value is fixed once the product is defined

NOTE 1 Changing the value of a characteristic of a product would mean changing the product.

EXAMPLE For a ball bearing, the inner diameter and the outer diameter are product characteristics.

NOTE 2 Adapted from ISO 13584-24:2003, definition 3.12.

**3.6**
**class**
abstraction of a set of similar products

NOTE 1 A product that complies with the abstraction defined by a class is called a class member.

NOTE 2 A class is an intentional concept that can take different extensional meanings in different contexts.

EXAMPLE The set of products used by a particular enterprise and the set of all ISO-standardized products are two examples of contexts. In these two contexts (the particular enterprise and ISO), the set of products that are considered as members of the *single ball bearing* class can be different, in particular because employees of each enterprise ignore a number of existing single ball bearing products.

NOTE 3 Classes are structured by class inclusion relationships.

NOTE 4 A class of products is a general concept as defined in ISO 1087-1. Thus, it is advisable that the rules defined in ISO 704 be used for defining the designation and definition attributes of classes of products.

NOTE 5 In the context of the ISO 13584 series, a class is either a characterization class, associated with properties and usable for characterizing products, or a categorization class, not associated with properties and not usable for characterizing products.

**3.7**
**class inclusion relationship**
relationship between classes that means inclusion of class members: if A is a superclass of A1 this means that, in any context, any member of A1 is also member of A

EXAMPLE 1 The set of products used by a particular enterprise and the set of all ISO-standardized products are two examples of contexts.

EXAMPLE 2 In any context, the class *capacitor* includes the class *electrolytic capacitor*.

NOTE 1 Class inclusion defines a hierarchical structure between classes.

NOTE 2 Class inclusion is a conceptual relationship that does not prescribe anything at the data representation level. Consequently, it does not prescribe any particular database schema or data model.

NOTE 3 In the model defined in this part of ISO 13584, the “is-a” relationship ensures class inclusion. This part of ISO 13584 recommends that the “case-of” relationship also ensure class inclusion.

NOTE 4 The class inclusion relationship is also called subsumption.

**3.8**
**class member**
product that complies with the abstraction defined by a class

**3.9**
**class valued property**
property that has one single value for a whole characterization class of products

NOTE 1 The value of a class valued property is not defined individually for every single product of a characterization class, but globally for the class itself.

NOTE 2 When all products from a characterization class of products have the same value for a particular property, defining this property as a class valued property permits to avoid duplication of the value for each instance.

NOTE 3 Class valued properties can also be used to capture some commonality between different characterization classes when such a commonality is not captured by the hierarchy structure (see example of RULE 4b).

**3.10**
**common ISO13584/IEC61360 dictionary model**
data model for product ontology, using the information modeling language EXPRESS, resulting from a joint effort between ISO/TC 184/SC 4/WG 2 and IEC SC3D

NOTE 1 Several levels of allowed implementations, known as conformance classes, are defined for the common ISO13584/IEC61360 dictionary model. Conformance class 1 consists of the various schemes documented in this part of ISO 13584 (that duplicate information contained in IEC 61360-2), more the **ISO13584_IEC61360_dictionary_aggregate_extension_schema** documented in ISO 13584-25 (duplicated in IEC 61360-5 ). Other conformance classes are documented in ISO 13584-25 (conformance classes 2, 3 and 4).

NOTE 2 In the ISO 13584 standard series, each particular product ontology addressing a particular product domain and based on the common ISO13584/IEC61360 dictionary model is called a reference dictionary for that domain.

**3.11**
**context dependent characteristic of product**
property of a *product* whose value depends on some *context parameters*

NOTE 1 For a given product, a context dependent characteristic is mathematically defined as a function whose domain is defined by some context parameters that define the product environment.

EXAMPLE For a *ball bearing*, the *life-time* is a context dependent characteristic that depends on the *radial load*, the *axial load* and the *rotational speed*.

NOTE 2 Adapted from ISO 13584-24:2003, definition 3.22.

**3.12**
**context parameter**
variable whose value characterizes the context in which a *product* is inserted

EXAMPLE 1 The *dynamic-load* applied to a *bearing* is a context parameter for this *bearing*.

EXAMPLE 2 The *ambient temperature* in which the *resistance* of a *resistor* is measured is a context parameter for this *resistor*.

NOTE 1 This definition supersedes the definition given in ISO 13584-24:2003, that was the following: “a variable of which the value characterizes the context in which it is intended to insert a *product*”.

NOTE 2 In the ISO 13584 standard series, a property value is represented as a data element type.

**3.13**
**data element type**
unit of data for which the identification, description and value representation have been specified

NOTE In the ISO 13584 standard series, a property value is represented as a data element type.

**3.14**
**dictionary data**
set of data that represents product ontologies possibly associated with product categorizations

NOTE 1 It is advisable that dictionary data be exchanged using some conformance class of the common ISO/IEC dictionary model.

NOTE 2 This definition of dictionary data supersedes the previous definition from the first edition of this part of ISO 13584 that was the following: “the set of data that describes hierarchies of characterization classes of products and properties of these products”.

**3.15**
**dictionary element**
set of attributes that constitutes the dictionary description of certain objects of the application domain

EXAMPLE 1 A dictionary compliant with this part of ISO 13584 provides for the description of classes, properties, information sources and datatypes.

EXAMPLE 2 A dictionary compliant with ISO 13584-24:2003 provides for the description of classes, properties, information sources, datatypes, tables, documents and program libraries.

**3.16**
**family of products**
set of products represented by the same characterization class

NOTE This definition supersedes the definition given in ISO 13584-24:2003, that was the following: "a simple or generic family of parts".

**3.17**
**feature**
aspect of a product that can be described by a characterization class and a set of property-value pairs

NOTE 1 In the real world, a feature instance only exists embedded within the product of which it is an aspect.

EXAMPLE 1 The head of a screw is a feature described by a head class and a number of head properties, which depends upon the head class. A screw head only exists when it belongs to a screw.

NOTE 2 Features are represented by means of **item_class** whose the **instance_sharable** attribute equals *false*.

NOTE 3 The **instance_sharable** attribute allows to specify the conceptual status of an item: either a stand-alone item (**instance_sharable** =*true*), or a feature (**instance_sharable** =*false*). It does not imply any constraint at the data representation level In the common ISO13584/IEC61360 dictionary model, representing several real world instances that share the same EXPRESS representation by a single EXPRESS entity, or by several EXPRESS entities is considered as implementation dependant. There exist no mechanism for specifying whether data value of a feature instance may or may not be shared.

EXAMPLE 2 The same instance of a screw head class can be referenced by several instances of a screw class. It means that there exists several screw heads, but that all these screw heads have the same characterization class and the same set of property values. The **instance_sharable** attribute allows to specify that changing this instance of the screw head class would change several instances of the screw class.

**3.18**
**imported property**
property defined in a class that is selected by another class of the same or of a different reference dictionary, by means of the case-of relationship, to become applicable to the latter class

NOTE 1 Only properties that are visible and/or applicable in a class can be imported from this class.

NOTE 2 Importation between classes of different reference dictionaries allows reusing properties, defined for example in a standard reference dictionary, without redefining them.

NOTE 3 Importation between classes of the same reference dictionary acknowledges the fact that some products can perform several functions, requiring the capability to import property from several higher level classes.

NOTE 4 When it is imported in a new class, a property keeps its original identifier, thus all the attributes do not need to be duplicated.

NOTE 5 An imported property is applicable for the class where it is imported.

**3.19**
**information**
facts, concepts or instructions

[ISO 10303-1:1994, definition 3.2.20]

**3.20**
**information model**
formal model of a bounded set of facts, concepts or instructions to meet a specified requirement

[ISO 10303-1:1994, definition 3.2.21]

**3.21**
**information supplier**
**supplier**
organization that delivers an ontology or a supplier library in the standard format defined in this ISO 13584 and that is responsible for its content

NOTE This definition of supplier supersedes the definition of information supplier from ISO 13584-1:2001 that was the following: “organization that delivers a supplier library in the standard format defined in this International Standard and is responsible for its content”.

**3.22**
**international registration data identifier**
internationally unique identifier for a certain object of the application domain as defined in ISO/IEC 11179-5

NOTE 1 Only international registration data identifiers compliant with ISO/TS 29002-5 are used in the context of the ISO 13584 standard series.

NOTE 2 An international registration data identifier may be used for representing the content of a basic semantic unit that identifies a dictionary element as a string.

NOTE 3 An international registration data identifier may also be used for identifying the content of an attribute of a dictionary element.

EXAMPLE The unit of measure of a property, a value of a property or a constraint over a property may be identified by an IRDI.

**3.23**
**is-a relationship**
class inclusion relationship associated with inheritance: if A1 *is-a* A, then each product belonging to A1 belongs to A, and all that is described in the context of A is automatically duplicated in the context of A1

NOTE 1 This mechanism is usually called “inheritance”.

NOTE 2 In the common ISO13584/IEC61360 dictionary model, the is-a relationship can only be defined between characterization classes. It is advisable that it defines a single hierarchy and it ensures that both visible and applicable properties are inherited.

**3.24**
**is-case-of relationship**
**case-of**
property importation mechanism: if A1 is *case-of* A, then the definition of A products also covers A1 products, thus A1 can import any property from A

NOTE 1 The goal of the *case-of* relationship is to allow connecting together several class inclusion hierarchies while ensuring that referenced hierarchies can be updated independently.

NOTE 2 There is no constraint that the case-of relationship is intended to define single hierarchies.

NOTE 3 In the common ISO13584/IEC61360 dictionary model, the case-of relationship can in particular be used in four cases: (1) to link a characterization class to a categorization class, (2) to import, in the context of some standardized reference dictionaries, some properties already defined in other standardized reference dictionaries, (3) to connect a user reference dictionary to one or several standardized reference dictionaries, (4) to describe a product using the properties of different classes: when products of class A1 fulfil two different functions, and are thus logically described by properties associated with two different classes, A and B, A1 can be connected by is-a to e. g., A, and by case-of to B.

NOTE 4 The EXPRESS resource constructs for modeling the case-of relationships are defined in Annex F of this part of ISO 13584.

**3.25**
**item**
thing that can be characterized by means of a characterization class to which it belongs and a set of property value pairs

NOTE 1 This definition supersedes the definition given in ISO 13584-24:2003, that was the following: “a thing that can be captured by a class structure and a set of properties”.

NOTE 2 In the ISO 13584 standard series, both products and features of products that correspond to composite properties are items.

**3.26**
**leaf characterization class**
characterization class that is not further specialized into more precise characterization classes

EXAMPLE *Countersunk flat head screw with cross recess (type Y)* and *hexagon socket head cap screw with metric fine pitch thread* are leaf characterization classes defined in ISO 13584-511 .

**3.27**
**non-leaf characterization class**
characterization class that is further specialized into more precise characterization classes

EXAMPLE *Externally-threaded component* and *metric threaded bolt/screw* are non-leaf characterization classes defined in ISO 13584-511 .

**3.28**
**non-quantitative data element type**
data element type that identifies or describes an object by means of codes, abbreviations, names, references or descriptions

**3.29**
**part**
material or functional product that is intended to constitute a component of different products

[ISO 13584-1:2001, definition 3.1.16]

**3.30**
**parts library**
computer-sensible product ontology and computer-sensible description of a set of products by means of references to this ontology

NOTE This definition supersedes the definition given in the first edition of this part of ISO 13584, which was the following: “identified set of data and possibly programs which can generate information about a set of parts”.

**3.31**
**product**
thing or substance produced by a natural or artificial process

NOTE In this part of ISO 13584, the term product is taken in its widest sense to include devices, systems and installations as well as materials, processes, software and services.

**3.32**
**product categorization**
**part categorization**
**categorization**
recursive partition of a set of products into subsets for a specific purpose

NOTE 1 Subsets which appear in a product categorization are called product categorization classes, or product categories.

NOTE 2 A product categorization is not a product ontology. It cannot be used for characterizing products.

NOTE 3 No property is associated with categorizations.

NOTE 4 Several categorizations of the same set of products are possible according to their target usage.

EXAMPLE The UNSPSC classification, defined by the United Nations, is an example of a product categorization that was developed for spend analysis.

NOTE 5 Using the *is-case-of* relationship, several product characterization class hierarchies can be connected to a categorization hierarchy to generate a single structure.

**3.33**
**product categorization class**
**part categorization class**
**categorization class**
class of products that constitutes an element of a categorization

EXAMPLE *Manufacturing Components and Supplies*, and *Industrial Optics* are examples of a product categorization class defined in the UNSPSC.

NOTE 1 No rule is given in this part of ISO 13584 about how to select categorization classes. This concept is introduced (1) to clarify its difference with characterization class, and (2) to explain that the same characterization class can be connected to any number of categorization classes.

NOTE 2 There is no property associated with a categorization class.

**3.34**
**product characterization**
**part characterization**
description of a product by means of a product characterization class to which it belongs and a set of property value pairs

EXAMPLE *Hexagon_head_bolts_ISO_4014 (Product grades = A, thread_type=M, length= 50, Diameter = 8)* is an example of a product characterization.

**3.35**
**product characterization class**
**part characterization class**
**characterization class**
class of products that fulfil the same function and that share common properties

NOTE Product characterization classes can be defined at various levels of details, thus defining a class inclusion hierarchy.

EXAMPLE *Metric threaded bolt/screw* and *hexagon head bolt* are examples of product characterization classes defined in ISO 13584-511. The first characterization class is included in the second one. *Transistor* and *bipolar power transistor* are examples of product characterization classes defined in IEC 61360-4-DB. The second one is included in the first one.

**3.36**
**product ontology**
**part ontology**
**ontology**
model of product knowledge, done by a formal and consensual representation of the concepts of a product domain in terms of identified characterization classes, of class relations and of identified properties

NOTE 1 Product ontologies are based on a class-instance model that allows one to recognize and to designate the sets of products, called characterization classes, that have a similar function (e.g., *ball bearing*, *capacitor*), but also to discriminate within a class the various subsets of products, called instances, that are considered as identical. It is advisable that the rules defined in ISO 1087-1 be used for formulating designation and definitions of characterization classes. Instances have no definitions. They are designated by the class to which they belong, and a set of property-value pairs.

NOTE 2 Ontologies are not concerned with words but with concepts, independent of any particular language.

NOTE 3 "Consensual" means that the conceptualization is agreed upon in some community.

NOTE 4 "Formal" means that the ontology is intended to be machine interpretable. Some level of machine reasoning is logically possible over ontology, e.g., consistency checking, making inferences.

NOTE 5 "Identified" means that each ontology characterization class and properties are associated with a globally unique identifier allowing one to reference this concept from any context.

NOTE 6 The data model for ontology recommended in this part of ISO 13584 is the common ISO13584/IEC61360 dictionary model, whose simplest version is documented in this part of ISO 13584. More complete versions are documented in ISO 13584-25 and IEC 61360-5 (conformance classes 1, 2, 3 and 4 of both documents).

NOTE 7 In this part of ISO 13584, each product ontology addressing a particular product domain compliant with the common ISO13584/IEC61360 dictionary model is called a reference dictionary for that domain.

EXAMPLE The reference dictionary for electric components, which is defined in IEC 61360-4-DB, is a product ontology for electric components compliant with the common ISO13584/IEC61360 dictionary model. It is agreed upon by all member bodies of IEC SC3D. A corporate reference dictionary is agreed upon by experts designated by management on behalf of the company.

**3.37**
**property**
defined parameter suitable for the description and differentiation of products

NOTE 1 A property describes one aspect of a given object.

NOTE 2 A property is defined by the totality of its associated attributes. The types and number of attributes that describe a property with high accuracy are documented in this part of ISO 13584.

NOTE 3 This part of ISO 13584 has identified three different kinds of properties: product characteristics, context parameters and context-dependent product characteristics.

NOTE 4 This definition of property supersedes the previous definition from the first edition of this part of ISO 13584 that was the following: "an information that can be represented by a data element type".

NOTE 5 In the ISO 13584 standard series, a property value is represented as a data element type.

**3.38**
**property data type**
allowed set of values of a property

**3.39**
**property definition class**
product characterization class in the context of which a product property is defined

NOTE In the common ISO13584/IEC61360 dictionary model, each product property has one property definition class that defines its domain of application. The property is only meaningful for this class, and all its subclasses, and it is said to be *visible* over this domain.

EXAMPLE In ISO 13584-511 , *wrenching height* has *nut* as its property definition class and *major diameter of external thread* has *metric external thread* as its property definition class.

**3.40**
**quantitative data element type**
data element type with a numerical value representing a physical quantity, a quantity of information or a count of objects

**3.41**
**reference dictionary**
product ontology compliant with the common ISO13584/IEC61360 dictionary model

NOTE In the ISO 13584 standard series, a product ontology that addresses a particular product domain, based on the common ISO13584/IEC61360 dictionary model, is called a reference dictionary for that domain.

**3.42**
**resource construct**
collection of EXPRESS language entities, types, functions, rules and references that together define a valid description of data

[ISO 13584-1:2001, definition 3.1.21]

NOTE This definition is adapted from the definition of resource construct in ISO 10303-1:1994, i.e., "the collection of EXPRESS language entities, types, functions, rules and references that together define a valid description of product data".

**3.43**
**subclass**
class that is one step below another class in a class inclusion hierarchy

NOTE In the common ISO13584/IEC61360 dictionary model, class inclusion hierarchies are defined by the *is-a* relationship. They can also be established by the *case-of* relationships.

**3.44**
**superclass**
class that is one step above another class in a class inclusion hierarchy

NOTE 1 In the common ISO13584/IEC61360 dictionary model, class inclusion hierarchies are defined by the *is-a* relationship. They can also be established by the *case-of* relationships.

NOTE 2 In the common ISO13584/IEC61360 dictionary model, a class has at most one superclass specified by means of an *is-a* relationship.

**3.45**
**supplier library**
parts library of which the information supplier is different from the library user

NOTE This definition supersedes the definition given in ISO 13584-1:2001, that was the following: "set of data, and possibly of programs, for which the supplier is defined and that describes in the standard format defined in this International Standard a set of products and/or a set of representation of products".

**3.46**
**visible property**
property that has a definition meaningful in the scope of a given characterization class, but that does not necessarily apply to the various products belonging to this class

NOTE 1 Meaningful in the scope of a given characterization class means that a human observer is able to determine, for any product of the characterization class, whether the property applies, and, if it applies, to which product aspect it corresponds.

NOTE 2 The concept of a visible property allows sharing the definition of a property among product characterization classes where this property does not necessarily apply.

EXAMPLE The *non-threaded length* property is meaningful for any class of *screw* but it applies only to those screws that have a non-threaded part. It can be defined as visible at the *screw* level, while becoming applicable only in some subclasses.

NOTE 3 All the visible properties of a superclass that is a product characterization class are also visible properties for its subclasses.

NOTE 4 To facilitate integration of component libraries and electronic catalogues based on ISO 13584-24:2003 and ISO 13584-25, these parts of ISO 13584 request that only properties that are applicable to a class be used to characterize their instances in component libraries and electronic catalogues.

NOTE 5 This definition of a visible property supersedes the previous definition from ISO 13584-24:2003 that was the following: "a property that is defined for some class of products and that does not necessarily apply to the different products of this class of products".

# 4 Abbreviated terms

| | |
|---|---|
| BSU | Basic Semantic Unit |
| DET | Data Element Type |
| ICS | International Classification of Standards |
| IRDI | International Registration Data Identifier |
| MathML | XML schema for mathematical notations |
| SI | Système International d'Unités (International System of Units) |
| UNSPSC | United Nations Standard Products and Services Code |

# 5 Description of a hierarchy of characterization classes of products

## 5.1 Relationships between product categorization and product ontologies

This part of ISO 13584 provides the resource constructs for developing two kinds of product hierarchies:

— product categorizations, often called classifications, where sets of products are recursively split into subsets for organization purpose, such hierarchies are not associated with properties, and

— product ontologies, where characterization classes of products are specialized into more precise characterization classes for product characterization purposes, such classes are associated with properties.

Product categorizations are never used for characterizing products. Thus, changes in product categorizations do not affect product characterizations as they might be recorded in product structure or in a database. Contrariwise, changes in product ontologies affect all existing product characterizations. Product categorizations and product ontologies have a quite different life cycle. Product ontologies should be very stable when product categorizations may change over time, over space and over kinds of users without any consequence for product characterizations.

As a consequence, product categorizations and product ontologies shall never be mixed, nor connected by is-a relationships. When it proves useful to organize various ontologies into a single structure, or into several searching structures, characterization classes may be connected to categorization classes using the case-of relationship.

## 5.2 Relationships between classes

### 5.2.1 Class inclusion relationship

In object-oriented analysis, class inclusion relationship, also called subsumption, constitutes a basic operation of modeling domain knowledge. A class A1 is said to be subsumed by a class A if, in any context, every object belonging to A1 also belongs to A.

In most object-oriented languages, class inclusion relationship is represented by inheritance: if A subsumes A1, not only all instances of A1 belong to A, but also all properties defined for A also apply to A1. Moreover, this inheritance mechanism is the only mechanism which allows two classes to share the same properties. This identification of class inclusion and inheritance, which often leads to multiple inheritance, makes object-oriented models not modular and poorly reusable in contexts slightly different from the ones where they were designed for. To provide for modularity of product ontologies, two different representations of the class inclusion relationship are defined for the ontology model defined in the ISO 13584 series of standards.

— The usual is-a relationship, defined in this part of ISO 13584, allows single groups or organizations to build single subsumption hierarchies, either product ontologies or product categorizations, with simple inheritance of properties for *pro*duct ontologies.

— The case-of relationship, whose EXPRESS resource constructs are defined in Annex F of this part of ISO 13584, provides for modularity of ontologies. It does not imply any automatic inheritance, but it allows the subsumed class to explicitly import some or all of the properties which are visible and/or applicable for the subsuming class. This relationship allows in particular: (1) to link a characterization class to one or several categorization classes, (2) to import in the context of some standardized characterization hierarchies some properties already defined in other standardized characterization hierarchies, (3) to connect a user dictionary to one or several standardized characterization hierarchies (see Annex J), (4) to describe a product using the properties of different classes. When products of a class A1 fulfil two different functions, and thus should be described by properties associated with two different classes, A and B, A1 may be connected by is-a to e; g., A, and by case-of to B.

Note that class inclusion relationship may be used both for grouping several subclasses, that represent different kinds of objects, within a common abstract superclass, and for defining several specialization of a kind of object, defined by a particular superclass, into more specialized kinds of the same object, defined by specialized subclasses. In both cases, is-a relationship is used if inheritance is implied, case-of is used when no inheritance is implied.

### 5.2.2 Aggregation and composition

Products may be assemblies of other products. For instance a bolted assembly may consist of a bolt, a washer and a nut. Such a relationship if often called *aggregation*.

Some products may also be difficult to characterize using simple properties. They may be more easily described by several features of the product, each one described both by referencing a characterization class and by property-value pairs. Such a relationship between a characterization class and each of its feature classes is often called a *composition* relationship.

EXAMPLE 1 A screw is more easily characterized by separately specifying its head (belonging e. g., to *hexagonal head class*, with a *height* and a *distance across flats*), its thread (belonging e. g., to *metric thread class*) and its driving feature. The relationship between the screw class and each of its feature classes is a *composition* relationship.

Aggregation and composition need to be conceptually distinguished. Aggregation applies when a product that is included in another product may exist independently of the latter. The assembly has not necessarily the same life cycle as its components, and it is possible that a component changes without control of the assembly. Contrariwise, composition applies for features that have exactly the same life cycle as their embedding products, that may not exist without their embedding product and that may not change without changing the product to which they belong. Thus, distinction between composition and aggregation, and between feature and stand-alone product is clearly defined at the conceptual level.

But this conceptual difference should not necessarily be reflected at the data representation level.

EXAMPLE 2 An ISO-standardized screw thread can only exist as part of a screw. Thus, at the conceptual level, it is considered as a property of screws (composition). Nevertheless, ISO-standardized screw threads only exist in a limited set of dimensions. Thus, For an efficient data storage and exchange, it should be possible at the data level:

— to consider ISO-standardized screw threads as a set of predefined instances that may be e. g., stored in a table,

— to exchange this set of instances independently of any screw, and

— to describe a screw without embedding its thread property values, but just referencing a thread instance, possibly exchanged separately.

In the common ISO13584/IEC61360 dictionary model, composition and aggregation are considered as a conceptual difference without particular requirements on data representation. Both products and feature shall be modeled as **item_class**es, and both composition and aggregation relationships are represented by means of a property. The conceptual difference is established by a particular Boolean attribute of **item_class** called **instance_sharable**. An **item_class** of which **instance_sharable** equals *true* represents a stand-alone product that exists independently of any aggregation relationship. The same real world instance of such a

class may be referenced by any number of other real world instances. An **item_class** of which **instance_sharable** equals *false* is a feature class that represents a composite property. A real world instance of such a class may only exist when embedded in exactly one product. Nevertheless, it is implementation dependent to decide whether several real world instances of features modeled by the same set of property-values pairs are represented by several EXPRESS pieces of data or by the same piece of data in the data exchange file.

## 5.3 Simultaneous description of characterization classes of products and products properties

A characterization hierarchy shall be organized as one or several tree-structures with single inheritance. At each level of the hierarchy, the subclasses of a characterization class of products should be mutually exclusive, whenever possible.

— Roots of each hierarchy shall be a characterization class of products that includes all the products that are intended to be characterized by the characterization hierarchy.

— Below a root, each non-leaf characterization class shall be split into non-leaf characterization subclasses until leaf characterization classes of products are obtained. In choosing these characterization classes, the instantiation rule (RULE 3) is important. Each of these characterization classes may be used for product characterization purpose.

It is often recommended to start from the leaf characterization classes that correspond to those families of products which are currently of widespread use in the domain and such that for each class all its instances may be described by the same properties. Then non-leaf characterization classes are created progressively above these classes using the instantiation rule criterion defined in 6.1.3.

Both the standardized characterization hierarchies and any particular supplier hierarchy shall simultaneously define the following:

— (hierarchical) characterization classes of products;

— (non hierarchical) properties.

The meaning of each characterization class of products and of each property shall be such that a human observer shall be able to determine, for a given product:

— those characterization classes of products to which it belongs and those to which it does not belong;

— that aspect of the product to which every applicable property corresponds.

The simultaneous definition of characterization classes of products and of properties related to every characterization class of products improves their definition. The extent of a characterization class of products becomes clearer through the properties that apply to its products; the meaning of a property is explained by the characterization class of products defining its field of possible application.

By defining characterization classes of products through a hierarchy with inheritance, the definition of each property can be factored to the highest characterization class level for which this property is meaningful and may be applicable in various subclasses.

## 5.4 Applicable and visible properties

Thus, for defining a precise characterization hierarchy, two kinds of requirements shall be addressed.

— The precise definition of a property requires to refer in its definition to the kind of products for which the property is meaningful.

— The precise definition of a class requires to provide criteria for class membership.

In the ISO 13584 standard series, these two requirements are represented by means of two of class- property relationships.

— The domain for which a property is meaningful is specified by means of a class that constitutes its *property definition class*. The textual definition of the property contains in general references to this class. Thus, only products that are member of the property definition class of a property, or of any class included in that class, may have an aspect that can be mapped unambiguously onto the property. The property is said to be *visible*, i.e., meaningful, for this class and all its subclasses.

EXAMPLE 1 The *relubrication feature* defined within the *bearing* class as "device which allows lubricant to reach the rolling or sliding surface of a bearing" is clearly defined for any bearing. But it cannot be used unambiguously for characterizing any product that is not a *bearing*.

— The criteria for class membership is specified by means of a set of properties. Only those products that have an aspect corresponding to each of these properties are members of the class. All these properties that are criteria for class membership are said to be *applicable* to this class.

EXAMPLE 2 The *bore diametre* and the *outside diametre* are properties of a *rolling bearing*. A product for which such characteristics do not exist is not a *rolling bearing.*

All the products of a subclass being members of its superclasses, applicability is inherited through a class inclusion hierarchy.

EXAMPLE 3 The *bore diametre* and the *outside diametre* are properties that necessarily exist for all products belonging to a subclass of *rolling bearing* such that: *ball bearing* and *roller bearing.*

When a property *P* is defined with a class *C* as its property definition class, it may arrive that some products in *C* have no aspect that corresponds to P. Thus not all the visible properties of a class are necessarily applicable properties for that class. It is the reason why visible properties and applicable properties need to be distinguished.

EXAMPLE 4 A *relubrication feature* does not exist for sealed bearings for which lubrication is done one and for all. Nevertheless, sealed bearings are *bearing*s.

The root of the tree were a property is visible is to be selected when the property is defined because a property definition shall include the specification of the property definition class. Applicable properties shall be defined when a characterization class is defined. There exist two mechanisms for creating an applicable property in a characterization class:

— the first one is to turn on as applicable a property that is already specified as visible for this class, possibly through inheritance;

— the second one is to import the property from another class, possibly belonging to another hierarchy, through the case-of class to class relationship. The case-of relationship expressing the fact that the definition of the referenced class also covers the referencing class products, properties of the referenced class are meaningful for product of the referencing class.

NOTE 1 The main goal of the case-of relationship is to share properties across different hierarchies

NOTE 2 To facilitate integration of component libraries and electronic catalogues based on ISO 13584-24:2003 and ISO 13584-25, these parts of ISO 13584 request that only properties that are applicable to a class be used to characterize their instances in component libraries and electronic catalogues.

## 5.5 Purpose of a standardized characterization hierarchy

Due to the diversity of the real world, the characterization classes of products and the properties forming a standardized characterization hierarchy are not intended to exhaustively describe every product. They are only aimed at characterizing the various products of a domain by means of those characterization classes and those properties on which consensus has been reached. Among other capabilities, standardized characterization hierarchies should at least provide for:

— multi-supplier characterization;

— interchangeability between different products;

— multi-supplier search.

## 5.6 Use of the standardized characterization hierarchy

When describing his/her own products a user of the ISO 13584 standard series may choose:

— to make reference to standardized characterization classes of products and properties, when they are defined by the standardized characterization hierarchy;

— to define its own characterization hierarchy, and to connect it to one or several standardized characterization hierarchies using the case-of relationship defined in Annex F of this part of ISO 13584, importing from it all the properties that prove useful in its context.

In the latter case, the user shall connect each of his/her characterization classes of products to the lowest characterization class of the standardized characterization hierarchy that includes all the products of this characterization class. Then, the user may refine this characterization class with more specific classes as needed. The higher the level of the connection is, the less the user will be able to take advantage of the standardized characterization hierarchy and of the corresponding properties (see Annex J).

The first approach is simpler. The second approach allows to build a hierarchy more adapted to the particular needs of each particular user. Anyway, thanks to the importation mechanism defined by the case-of relationship, automatic integration of data from different sources may be performed for all those data that reference standard-defined properties.

## 5.7 Class valued property

Due to the diversity of the perspectives that may be adopted for a set of objects of the real world, different structures for a non-leaf characterization class of products may emerge (see example in section 6.1.4).

In this case, the structure that permits the maximum applicability of the factored properties shall be selected for the standardized characterization hierarchy. Each other perspective considered relevant is represented as a property that may only be assigned one single value to a characterization class of the tree. The assignment of the property to a characterization class may be defined at any level of the subtree. The value assigned this way is inherited by all the subclasses.

The data type of such a property is represented as a set of unique codes. Each code is associated with different human readable and translatable representations.

Querying particular values of this property at the level of some non-leaf characterization classes would allow to retrieve sets of subclasses of this class independently of the hierarchical decomposition of this class.

## 5.8 Compatibility between ISO 13584 and IEC 61360 standard series

The schemata presented in this part of ISO 13584 have been developed as a joint effort ISO TC184/SC4/WG2 and IEC SC3D to combine the requirements of IEC 61360 and ISO 13584 users. As explained in F.1.2, these schemata have been further extended to constitute the common ISO13584/IEC61360 dictionary model of which the schemata presented in this document are the simplest functional subset. This subset is documented both in IEC 61360-2 and in this part of ISO 13584, and it is allowed for use, both in the context of ISO 13584 and in the context of IEC 61360, for exchanging simple dictionary data.

The schemata presented in this document contain a number of entities defined as abstract supertypes without any subtype defined. These resources are mechanisms that provide for latter extensions. As specified by the ISO 10303-11:1994 entities defined as abstract supertypes cannot be directly used.

Two attributes defined in the schema documented in this part of ISO 13584 were specifically designed to address IEC 61360 particular requirements. These extensions are as follows:

a) in the **item_class** entity, the optional **value_code** attribute;

b) in the **value_domain** entity, the optional **terms** attribute.

These optional attributes are not recommended for use by ISO users. Nevertheless these attributes are part of the common ISO13584/IEC61360 dictionary model and they shall be processed by any implementation that claim conformance to either standard to ensure full interoperability between ISO 13584 and IEC 61360 dictionary data.

# 6 Rules for creating hierarchies of characterization classes of products

To simplify the analysis when creating standardized characterization hierarchies or supplier hierarchies, the following rules, that allow the splitting of the analysis into two phases, should be applied. The first set of rules refers to the choice of the characterization class hierarchy. The second set of rules refers to the association of properties with characterization classes of products. In fact the process should be iterative. Identification of new properties might lead to reconsidering some design choices done during class hierarchy development, and vice versa.

## 6.1 Choice of characterization class hierarchy

### 6.1.1 Field of application

RULE 1 – Field covered by the hierarchy.

The definition of the root class of a characterization hierarchy shall clearly identify all the products that may be characterized using this characterization hierarchy.

It is recommended to develop standardized characterization hierarchies for each product domain addressed by ISO or IEC standards.

The root of such hierarchies, and/or possibly some other nodes, may be connected with the International Classification of Standards (ICS) to clarify the domain covered by the hierarchy.

NOTE Connection with the ICS, and possibly with other categorizations may be formally expressed using the case-of relationship defined in Annex F of this part of ISO 13584.

### 6.1.2 Upper section of the class hierarchy

RULE 2 – Organization of a product domain.

The role of the upper section of a standardized characterization hierarchy is to organize the various aspects of a product domain using class inclusion relationships and to factorize properties. This may lead to define abstract classes that gather items that are different in nature, such that products classes and feature classes provided that some visible properties need to be shared at these levels. Such classes are said to be abstract because all members of such classes are necessarily also members of some of their subclasses.

When there is no need to share some properties across items that are different in nature, such that products classes and feature classes, separate hierarchies may be defined.

### 6.1.3 Lower section of the class hierarchy

RULE 3 – Instantiation rule.

Below the upper section of the hierarchy, a characterization class of products shall be created only:

— when it is possible and advisable to order or to search for a product by characterizing it as a member of this characterization class, or

— when a user could reasonably choose a product of such a characterization class to represent a significant state (phase) of his design process.

EXAMPLE To group screws for metal, wood and sheet metal within one characterization class "screws", is contradictory to the rule of instantiation. During a design process a designer would never select a screw without knowing if this screw was a machine screw or a wood screw. Therefore, it is necessary to define below the upper section "bolts/screws/studs" different characterization classes of products for the different types of screws.

### 6.1.4 Multiple perspectives on the class hierarchy

It may appear that at some point in the class hierarchy, several point of views can be used to define a substructure for that hierarchy.

RULE 4a – Maximum applicability

When the application of the instantiation rule (RULE 3) leads to different possible structures of some characterization classes depending upon a point of view, the structure that enables the maximum applicability of the factorized properties shall be selected.

RULE 4b – Class valued properties

The other points of views considered essential for selection purposes – not selected however for the structure according to the maximum applicability rule (RULE 4a) – shall be represented by means of class valued properties (cf. paragraph 5.7). Those are intended to be queried by the user to select a set of characterization classes that correspond to the other points of view.

RULE 4c – Class valued assignment level

A class valued property shall be assigned a value at the level of a characterization class where that property value applies for *every* leaf characterization class of the corresponding subtree and does not apply for the immediate parent characterization class.

EXAMPLE The characterization class of circular bearings may split up into ball, needle and conical bearings or into sealed bearings and non-sealed bearings. The first perspective is selected for structuring purposes (RULE 4a). A class valued property, named "is_sealed", that is intended to take a constant BOOLEAN value in the lower characterization classes is defined as class valued property at the circular bearing characterization class level (RULE 4b). That property is assigned a value in all the classes whose all bearings are either sealed (*is_sealed* = *TRUE*) or unsealed (*is_sealed* = *FALSE*).

## 6.2 Association of properties

### 6.2.1 Properties to be considered

RULE 5 – Choice of properties.

As a minimum, those properties that characterize a non-leaf characterization class of products and that can be used when searching products for every sub-class of this non-leaf characterization class of products, shall be associated as visible or as visible and applicable as appropriate with the standardized characterization hierarchy.

Those properties that are not used (or rarely used) for searching purposes can be added as appropriate.

### 6.2.2 Semantic identification of properties

The following rule gives two criteria for deciding when two properties have the same semantics.

RULE 6 – Semantic identification

Two characteristic properties of two different products shall be factored to a higher level of the hierarchy, if and only if, they are considered to have the same semantic meaning. Such a decision shall be justified by satisfying one of the two following criteria:

— interchangeability criterion: the two products can, in some circumstances, be interchangeable and when interchanged the two characteristic properties shall have identical values;

— criterion of homogeneity in processing: the two properties play an identical role with respect to some process, automatic or non automatic, that might be performed over a set of products.

Otherwise, two different properties shall be defined.

EXAMPLE 1 The properties thread diameter of hexagon head screws and cylindrical head screws fit the interchangeability criterion.

EXAMPLE 2 The properties mass (calculation of the total mass of a product), designation (for part lists), diameter of conical attachment of tools (for automatic tool exchange) or the outside diameter of cylindrical electronic components (for automatic placement) fit the criterion of homogeneity in automatic processing.

EXAMPLE 3 A thread diameter of a screw for wood or of a screw for metal is characteristic properties that are not represented as the same property.

### 6.2.3 Factoring rule

It is not possible in every case to define a hierarchy in which:

— every property having the same semantics in several subclasses is factorized (i.e. defined only once and inherited) as a unique property at the level of the ascendant characterization class;

— any property defined at the level of a non-leaf characterization class of products applies actually (i.e. has a value) to every product in every sub-class.

To create an intermediate characterization class in order to factorize a property may be prevented by the instantiation rule (RULE 3) that is used for defining the characterization class hierarchy.

The following rule provides guidelines for systematically factorizing of properties having the same semantics while maintaining the instantiation rule.

RULE 7 – Applicability of inherited properties.

Two properties having the same semantics (RULE 6) in two characterization classes shall be factorized as a unique property defined at the level of the common ascendant characterization class. If this property does not apply to some subclasses, it shall be defined as a visible property. This property may be specified as applicable in the various subclasses where it is visible. When a property is specified as applicable, its definition shall be such that for any sub-class there is no doubt about its applicability, and when it applies, there is no doubt about the product aspect it corresponds to.

EXAMPLE 1 The property "material", the definition of which would be "material from which the product is made" cannot be factored; we do not know for example to which property it corresponds for a turning tool with carbide brazed tip.

EXAMPLE 2 The property "single material", whose definition would be "property only applicable to products made from one single material; the value of this property is the code of the single material", corresponds with the semantic identification rule (RULE 7).

# 7 Dictionary elements that describe properties of products

Properties of a product are classified into:

— product characteristics;

— context dependent characteristics;

— context parameters.

## 7.1 Mapping of properties onto the common ISO13584/IEC61360 dictionary model

In the common ISO13584/IEC61360 dictionary model, a property is mapped onto a **property_BSU** that carries its identification and a **property_DET** that provides its description. Note that **property_DET** is abstract, thus its subtypes have to be used.

A product characteristic is mapped onto the **non_dependent_P_DET** subtype of **property_DET**, a context dependent characteristic is mapped onto the **dependent_P_DET** subtype of **property_DET** and a context parameter is mapped onto the **condition_DET** subtype of **property_DET**.NOTE Context parameters and context dependent characteristic shall be represented at least when they prove useful for selection purposes.

The characterization class that defines the visibility of a property is mapped onto the **property_BSU.name_scope** attribute. The applicability of a property is defined by the class(es) to which it is applicable through the **class.described_by** attribute of this(these) class(es).

In this section, the main attributes linked directly or indirectly to a property are described.

## 7.2 Attributes

Each product property is described by a dictionary element that contains the following attributes:

— code;

— definition class;

— data type;

— preferred name;

— short name;

— preferred letter symbol;

— synonymous letter symbols;

— synonymous name;

— property type classification;

— definition;

— source document of definition;

— note;

— remark;

— unit;

— condition;

— formula;

— value format;

— date of original definition;

— date of current version;

— date of current revision;

— version number;

— revision number;

— is deprecated;

— is deprecated interpretation;

— administrative data.

The entry for each attribute is structured in the following way (entries may be omitted if not applicable):

— Name of the attribute;

— Obj: Objective;

— Descr: Description;

— Oblig: Obligation;

— Trans: Need for translation;

— For: Representation format or maximal number and type of characters if it is a string;

— Mapp: Mapping onto the attributes used in the common ISO13584/IEC61360 dictionary model;

— Expl: Example.

NOTE For internal purposes within a company, other attributes and/or other codes for some attributes can be defined but they cannot be used for exchange purposes.

### 7.2.1 Code

Obj: To identify a property within a characterization class of products and to allow an absolute identification within the data dictionary when it is associated with the information supplier code and with a version number.

Descr: A basic semantic unit associated with the property.

Oblig: Mandatory.

Trans: No translation.

Mapp: **property_DET\dictionary_element.identified_by\basic_semantic_unit.code**

### 7.2.2 Definition Class

Obj: To specify for which characterization class the property is defined.

Descr: The code of the characterization class that is the root of the tree where this property is visible.

Oblig: Mandatory.

Trans: No translation.

Mapp: **property_DET\dictionary_element.identified_by\property_BSU.name_scope**

### 7.2.3 Data Type

Obj: To specify the data type of the property. The data type describes the set of values that may be assigned to a property.

Descr: A type conforming to the type system specified in the common ISO13584/IEC61360 dictionary model that specifies the data type of the property.

Oblig: Mandatory.

Trans: No translation.

Mapp: **property_DET.domain**

NOTE **property_DET.domain** refers to **data_type** which again will be subtyped for the various possible data types. Thus, the data type is directly attached to the property. However, an identification mechanism using a **named_type** and a **data_type_BSU** is also available for the cases where the same data type is intended to be used for different properties. When a **data_type** is intended to be used by different properties, the definition of this **data_type** as a **named_type** provides for separate up-dating of the data-type and of the properties that refer to it.

### 7.2.4 Preferred Name

Obj: To give a name of the property (in full length where possible). It is used for communication and understanding.

Descr: The preferred name is identical to the name that is used in International Standards if available. If the preferred name in the International Standard is longer than the maximum length allowed for this attribute it shall be meaningfully abbreviated.

Oblig: Mandatory.

Trans: To be translated.

For: 255, alphanumeric.

Mapp: **property_DET\class_and_property_elements.names\item_names.preferred_name**

Expl: Threaded diameter

### 7.2.5 Short Name

Obj: To give a name of the property for representation in limited space.

Descr: It is a meaningful abbreviation of the preferred name. If standardized abbreviations exist they should be used. It can be identical to the preferred name or letter symbol.

Oblig: Optional.

Trans: To be translated, in the case that the short name is identical to the letter symbol, it shall be the same in all languages.

For: 30, alphanumeric.

Mapp: **property_DET\class_and_property_elements.names\item_names.short_name**

Expl: thread_diam.

### 7.2.6 Preferred Letter Symbol

Obj: To allow a shorter name of the property. When it exists, it is used in place of the short name for representation in tables, formula, drawings etc...

Descr: The letter symbol is unique within a characterization class of products. It shall be derived from International Standards. The preferred letter symbol is always provided in a text representation. It may also be provided in a MathML-Text representation.

EXAMPLE ISO 80000/IEC 80000 (formerly ISO 31), IEC 60027 , IEC 60748 and product standard may be source of letter symbols.

Oblig: Optional.

Trans: No translation.

For: Alphanumeric and MathML format (if provided).

Mapp: **property_DET.preferred_symbol**

### 7.2.7 Synonymous Letter Symbol

Obj: To allow further shorter name of the property. It is used for representation in tables, formula, drawings etc…

Descr: The letter symbol is unique within a characterization class of products. It shall be derived from International Standards.

EXAMPLE Examples of such standards are ISO 80000/IEC 80000 (formerly ISO 31), IEC 60027 , IEC 60748 or product standards.

None, one or more synonym letter symbols are allowed. The synonym letter symbol is always provided in a text representation. It may also be provided in an MathML representation.

Oblig: Optional.

Trans: No translation.

For: Alphanumeric and MathML format (if provided).

Mapp: **property_DET.synonymous_symbols**

### 7.2.8 Synonymous Name

Obj: Synonyms to the preferred name may be provided to facilitate transition from names used for local or historical reasons.

Descr: Alternative designation that differs from the given preferred name but represents the same concept. None, one or more synonymous names are allowed.

Oblig: Optional.

Trans: To be translated.

For: 255, alphanumeric.

Mapp: **property_DET\class_and_property_elements.names\item_names .synonymous_names**

### 7.2.9 Property Type Classification

Obj: To classify the different properties defined in order to make large collections of property definitions more manageable.

Descr: The whole set of properties are divided into subsets according to the categories defined in ISO 80000/IEC 80000 (formerly ISO 31). The property type classification attribute is the reference to the ISO 80000/IEC 80000 category that is relevant for the property.

Oblig: Optional.

Trans: No translation.

For: One capital letter and two digits.

Mapp: **property_DET.DET_Classification**

NOTE A survey of the main classes and categories of properties in ISO 80000/IEC 80000 is given in Annex A. The ISO 80000/IEC 80000 classification of quantitative measure is given in Annex H. The ISO 80000/IEC 80000 classification of non quantitative properties, also called identifications and indicators, is given in Annex A.

### 7.2.10 Definition

Obj: To describe the meaning of the property.

Descr: Statement that describes the meaning of the property and permits its differentiation from all other properties. It shall be a definition in the sense that it shall be complete and unambiguous. All significant words are free from homonym and synonymy.

Oblig: Mandatory.

Trans: To be translated.

For: Unlimited alphanumeric string.

Mapp: **property_DET\class_and_property_elements.definition**

### 7.2.11 Source Document of Definition

Obj: A reference to the source document from which the property definition was derived.

Descr: As a minimum the reference shall be given with the document number and date of issue of the document.

Oblig: Optional.

Trans: To be translated.

NOTE When the document is represented as an identified_document. The name of the document may be provided in various languages. In an exchange conforming to ISO 13584-25 or to ISO 13584-35 (OntoML) the documents themselves may be exchanged in various languages.

For: Alphanumeric. An identifier of the document.

Mapp: **property_DET\class_and_property_elements.source_doc_of_definition\ identified_document.document_identifier**

### 7.2.12 Note

Obj: To provide further information on any part of the definition, which is essential for the understanding of that definition.

Descr: It shall be copied from the definition in the source document into the definition of the property.

Oblig: Optional.

Trans: To be translated.

For: Unlimited alphanumeric string.

Mapp: **property_DET\class_and_property_elements.note**

### 7.2.13 Remark

Obj: Explanatory text to further clarify the meaning of the usage of the property.

Descr: Free text remarks. It shall not influence the meaning.

Oblig: Optional.

Trans: To be translated.

For: Unlimited alphanumeric string.

Mapp: **property_DET\class_and_property_elements.remark**

### 7.2.14 Unit

Obj: Prescription of the default unit, and possibly of alternative units in which the value of a quantitative property is expressed.

Descr: A formal model of one or several units, and/or one or several references to a dictionary of units from which the formal model of units may be obtained.

NOTE 1 Identifiers providing for referencing a dictionary of units allow to download the formal model from a resolution service compliant with ISO/TS 29002-20 .

Oblig: Mandatory (for quantitative data).

Trans: No translation.

For: Units are represented as specified in ISO 10303-41 using, if required, the extensions specified in the common ISO13584/IEC61360 dictionary model. Mathematical strings may also be provided. A mathematical string is always provided in a text representation. It may also be provided in MathML representation. Reference to a dictionary of units is done by identifiers compliant with ISO 29002 standard series.

Mapp: **property_DET.domain\int_measure_type.unit** or
**property_DET.domain\real_measure_type.unit** or
**property_DET.domain\rational_measure_type.unit**
and/or
**property_DET.domain\int_measure_type.unit_id** or
**property_DET.domain\real_measure_type.unit_id** or
**property_DET.domain\rational_measure_type.unit_id**
and/or
**property_DET.domain\int_measure_type.alternative_units** or
**property_DET.domain\real_measure_type.alternative_units** or
**property_DET.domain\rational_measure_type.alternative_units**
and/or
**property_DET.domain\int_measure_type.alternative_unit_ids** or

**property_DET.domain\real_measure_type.alternative_unit_ids** or
**property_DET.domain\rational_measure_type.alternative_unit_ids**

NOTE 2 The data model allows alternative units, but when alternative units are allowed, they are listed in the property definition and each property value represented according to one alternative unit is associated with its corresponding unit.

### 7.2.15 Condition

Obj: To formally identify the context parameter on which a context dependent characteristic depends.

Descr: To give the code(s) of the adequate context parameter(s).

Oblig: Mandatory for context dependent characteristics.

Trans: No translation.

For: A reference to a set of basic semantic units.

Mapp: **property_DET\dependent_P_DET.depends_on**

### 7.2.16 Formula

Obj: Rule or statement in mathematical form expressing semantics of a quantitative property. A formula shall not change any essential information of the meaning of that definition.

Descr: It is a mathematical expression of the property definition.

Oblig: Optional.

Trans: No translation.

For: A mathematical string (a mathematical string is always provided in a text representation; it may also be provided in a MathML text representation).

Mapp: **property_DET.formula**

### 7.2.17 Value Format

Obj: Specification of the length and pattern of the recommended presentation for displaying the value of a property whose data type is either **string_type** or any of its subtype, a **number_type** or any of its subtype or a collection of such values defined either by a **level_type** or an aggregate data type: **list_type**, **set_type**, **bag_type**, **array_type** or **set_with_subset_constraint_type**. If present this attribute provides for guidance to the system about how string and numeric values should be displayed. This guidance shall not contradict the data type and the possible constraints on the property value. The value of this attribute should be compatible with the data type of the property: it should not change this data type, else it should be ignored. The value of this attribute should also be compatible with the possible property constraints defined on the property values by means of the constraint schema provided in this edition of this part of ISO 13584, else it should be ignored.

NOTE 1 **list_type**, **set_type**, **bag_type**, **array_type** and **set_with_subset_constraint_type** are defined in ISO 13584-25.

Descr: When specified, the value format shall be defined according one of the height definitions below. The syntax of these formats is defined in Annex D using a subset of the Extended Backus-Naur Form (EBNF) defined in ISO/IEC 14977.

a) Non-quantitative data value formats: five definitions. They are intended for properties having a non-quantitative type data-type, i. e., **string_type** or one of its subtype, or collections of such values.

Alphabetic Value Format (A),

Mixed Characters Value Format (M),

Number Value Format (N),

Mixed Alphabetic or Numeric Characters Value Format (X),

Binary Value Format (B).

b) Quantitative data value formats: three definitions. They are intended for properties having numeric values, i.e., **number_type** or one of its subtype, or collections of such values.

Integer Value Format (NR1);

Real Numbers with Decimal-Mark Value Format (NR2),

Real numbers with Decimal-Mark and Exponent-Mark Value Format (NR3),

Rational Value Format (NR4).

Oblig: Optional.

Trans: No translation.

For: 80, alphanumeric.

Mapp: **property_DET.domain\simple_type.value_format**

NOTE 2 In case of named types, the value format refers to the value of the data type referenced by the **referred_type** attribute of this **named_type**.

### 7.2.18 Date of Original Definition

Obj: To show when the property was defined by the information supplier and thus when it was declared as valid by this supplier. This date will never be changed and can be used for verification purpose.

Descr: The entry shall be in accordance with ISO 8601.

Oblig: Optional.

Trans: No translation.

For: 10, alphanumeric.

Mapp: **property_DET\dictionary_element.time_stamps\dates.date_of_original_definition**

Expl: 1967-08-20.

### 7.2.19 Date of Current Version

Obj: To show the date when the current version was defined.

Descr: The entry shall be in accordance with ISO 8601.

Oblig: Optional.

Trans: No translation.

For: 10, alphanumeric.

Mapp: **property_DET\dictionary_element.time_stamps\dates.date_of_current_version**

### 7.2.20 Date of Current Revision

Obj: To show the date of the last revision number change.

Descr: The entry shall be in accordance with ISO 8601.

Oblig: Optional

Trans: No translation.

For: 10, alphanumeric.

Mapp: **property_DET\dictionary_element.time_stamps\dates.date_of_current_revision**

### 7.2.21 Version Number

Obj: To characterise each version of a property. A new version number of a property shall be created whenever a change in some attribute that describes this property influences its use.

NOTE 1 No change is allowed that affects the meaning of a property.

NOTE 2 The changes in a property that affect its version number are defined in Clause 9.

Descr: A string that contains a natural number to indicate the different versions of a property during the life cycle. Version numbers shall be issued in ascending order. A new version of the property shall be generated according to the rules given in Clause 9.

Oblig: Mandatory.

For: 10, numerical.

Trans: No translation.

Mapp : **property_DET\dictionary_element.identified_by\basic_semantic_unit.version**

### 7.2.22 Revision Number

Obj: To characterise each revision of the same version of a property. A new revision number of a property shall be created when a change in some attribute that describes this property influence neither its meaning nor its use.

NOTE 1 No change is allowed that affects the meaning of a property.

NOTE 2 The changes in a property that affect its revision number are defined in Clause 9.

Descr: A string that contains a natural number used for administrative control of a property. Consecutive revision numbers shall be issued in ascending order for each value of the version of a property. Per property, unique by its identifier, only one revision number is current at any time. A new revision number of the property shall be generated according to the definitions given in Clause 9. When a new version is issued, the revision is set to '0'.

Oblig: Mandatory.

For: 3, numerical.

Trans: No translation.

Mapp: **property_DET\dictionary_element.revision**

### 7.2.23 Is Deprecated

Obj: To specify whether a property may still be used or shall no longer be used for new characterizations.

NOTE 1 Deprecated properties may be maintained in a reference dictionary to allow users to interpret characterizations defined with previous versions of this dictionary.

Descr: An optional Boolean that specifies, when true, that a particular property has possibly been used in the past but shall no longer be used for new characterizations.

NOTE 2 When this attribute does not exist, the property is not deprecated.

Oblig: Optional.

For: A Boolean value.

Trans: No translation.

Mapp: **property_DET\dictionary_element.is_deprecated**

### 7.2.24 Is Deprecated Interpretation

Obj: To specify for a deprecated property the deprecation rationale and how instance values of the deprecated property should be interpreted.

Descr: A note that is mandatory when a property is deprecated.

Oblig: Optional.

For: A translatable text.

Trans: To be translated.

Mapp: **property_DET\dictionary_element.is_deprecated_interpretation**

### 7.2.25 Administrative data

Obj: To record information about the life cycle of a property.

Descr: An entity that represents the source language in which a property description was defined, the list of languages in which the property description is translated and the possible status that defines the life cycle state of the property.

Oblig: Optional.

For: An **administrative_data** entity.

Trans: No translation.

Mapp: **property_DET\dictionary_element.administration**

# 8 Dictionary elements that describe classes of products

## 8.1 Mapping of classes onto the common ISO13584/IEC61360 dictionary model

A characterization class is mapped in the common ISO13584/IEC61360 dictionary model onto a **class_BSU** that carries its identification and an **item_class** dictionary element that provides its description. Both instances of products and aspects of products are mapped onto **item_class**.

EXAMPLE 1 The head of a screw is a feature. In ISO 13584-511 , it is described by means of the *head* **item_class**.

The **item_class** entity includes an **instance_sharable** attribute that specifies the conceptual status of an item: either a stand-alone item (**instance_sharable** = *true*), or a feature (**instance_sharable** = *false*). Value of this attribute does not imply any constraint at the data representation level.

EXAMPLE 2 The same instance of a screw head class may be referenced by several instances of a screw class. It means that there exists several screw heads, but that all these screw heads have the same characterization class and the same set of property values.

NOTE 1 Two subtypes of **item_class**, named **component_class** and **material_class**, were defined within the dictionary model of the first edition of this part of ISO 13584 and IEC 61360-2. These subtypes are deprecated, and they are removed from this edition of this part of ISO 13584.

NOTE 2 The following changes ensure that a dictionary conforming with the first edition of this part of ISO 13584 conforms to this edition: (1) replace **component_class** and **material_class** by **item_class** throughout the reference dictionary; (2) add to each new **item_class** class the **instance_sharable** attribute, the value of which being *true*.

NOTE 3 Another subtype of **item_class**, named **feature_class,** was provided in ISO 13584-24:2003. This subtype is also deprecated and its usage is not recommended in new implementations of this part of ISO 13584 and IEC 61360-2.

NOTE 4 The following changes ensure that a dictionary conforming with the first edition of this part of ISO 13584 conforms to this edition: (1) replace **feature_class** by **item_class** throughout the reference dictionary; (2) add to each new **item_class** class the **instance_sharable** attribute, the value of which being *false*.

A categorization class is mapped in the common ISO13584/IEC61360 dictionary model onto a **class_BSU** that carries its identification and a **categorization_class** dictionary element that provides its description. Only a subset of the attributes of characterization classes is also used for categorization class. A particular attribute to categorization class, called **categorization_superclasses**, allows to specify its superclasses in the case-of class inclusion hierarchy.

NOTE 5 The case-of relationship for characterization classes also allows to import properties into the case-of class. Thus, this relationship is represented by means of a different attribute. This is defined in Annex F of this part of ISO 13584.

In this section, the main attributes linked directly or indirectly to a class are described.

## 8.2 Attributes

Each characterization class of products is described by a dictionary element that contains the following attributes:

— code;

— superclass;

— preferred name;

— short name;

— synonymous name;

— visible types;

— applicable types;

— class valued properties;

— visible properties;

— applicable properties;

— class value assignment;

— definition;

— source document of definition;

— note;

— remark;

— simplified drawing;

— date of original definition;

— date of current version;

— date of current revision;

— version number;

— revision number;

— constraints;

— is deprecated;

— is deprecated interpretation;

— administrative data.

Moreover, the dictionary element that describes a categorization class of products also contains the following attribute:

— **categorization_class_superclasses**.

The entry for each attribute is structured in the following way (entries may be omitted if not applicable):

— Name of the attribute;

— Obj: Objective;

— Descr: Description;

— Oblig: Obligation;

— Trans: Need for translation;

— For: Representation format or maximal number and type of characters if it is a string;

— Mapp: Mapping onto the resource constructs used in the common ISO13584/IEC61360 dictionary model;

— Expl: Example.

### 8.2.1 Code

Obj: To identify a characterization class or a categorization class of products and to allow an absolute identification within the data dictionary when associated with the information supplier code and with a version number.

Descr: A basic semantic unit associated with the characterization class of products.

Oblig: Mandatory.

Trans: No translation.

Mapp: **item_class\dictionary_element.identified_by\basic_semantic_unit.code**

### 8.2.2 Superclass

Obj: To reference the immediate unique parent characterization class by the is-a relationship of a characterization class.

Descr: A basic semantic unit of the immediate parent characterization class of the current characterization class.

Oblig: Optional (if it does not exist, the class has no superclass).

Trans: No translation.

For: A class basic semantic unit.

Mapp: **item_class\class.its_superclass**

NOTE 1 When defining a standardized characterization hierarchy, one or several node(s) may have no superclass. These nodes are the root or roots of the hierarchy.

NOTE 2 It is a recommended practice that roots of standardized characterization hierarchies, and/or possibly some other nodes, reference some nodes of the ISO/IEC International Classification for Standards (ICS) predefined tree by the *case-of* relationship to clarify the target domain addressed by each characterization hierarchies, or sub-hierarchies. Other categorization hierarchies may be referenced by the same means as appropriate.

NOTE 3 When a class of the ICS is referenced, the class code to be used is the code defined for that class in the referenced ICS, where dots are replaced by underscores. The version of the class to be used equals '001'. As specified in ISO 13584-26 , the supplier code to be used for the first edition of the ICS, published in 1992, is "112/1///_00_1", and the supplier code to be used for the sixth edition, published in 2005, is "112/1///_00_6". Dictionary data suppliers are encouraged to use the more recent editions of the ICS. Other editions are to be identified by a supplier code defined as "112/1///_00_<ICS edition number>", where <ICS edition number> is the integer edition number of the referenced ICS.

### 8.2.3 Preferred Name

Obj: To give a meaningful description of a characterization class or a categorization class of products (in full length where possible). It is used for communication and understanding.

Descr: The preferred name is identical to the name that is used in International Standards if available. If the preferred name in the International Standards is longer than the maximum length allowed for this attribute it shall be meaningfully abbreviated.

Oblig: Mandatory.

Trans: To be translated.

For: 255, alphanumeric.

Mapp: **class\class_and_property_elements.names\item_names.preferred_name**

Expl: Screw threads.

### 8.2.4 Short Name

Obj: To give a name to a characterization class or a categorization class of products for representation in limited space.

Descr: It is a meaningful abbreviation of the preferred name. If standardized abbreviations exist they should be used. It can be identical to the preferred name.

Oblig: Optional.

Trans: To be translated, in the case that the short name is identical to the letter symbol, it will be the same in all languages.

For: 30, alphanumeric.

Mapp: **class\class_and_property_elements.names\item_names.short_name**

### 8.2.5 Synonymous Names

Obj: Synonyms to the preferred name of a characterization class or a categorization class may be provided to facilitate transition from names used for local or historical reasons.

Descr: Alternative designation that defers given preferred name but represents the same concept. None, one or more synonymous names are allowed.

Oblig: Optional.

Trans: To be translated.

For: 255, alphanumeric.

Mapp: **class\class_and_property_elements.names\item_names.synonymous_names**

### 8.2.6 Visible Types

Obj: To define the new named types that may be referred to, as their data type, by the visible properties of this characterization class, or of any of its subclasses (visible type).

Descr: Different properties may have the same data type of values.

EXAMPLE A set of codes that identify materials.

Such data types may be defined as named types, independently of any property. They may be, later on, referred to as their data type by different properties. The definition of named type shall contain the code of the named type, the version number of the named type, and the data type of the named type specified by using the resource constructs of the common ISO13584/IEC61360 dictionary model. For the **non_quantitative_code_type** and **non_quantitative_int_type** the data type shall be specified as a set of **dic_value**s each one consisting of an unique code and a set of names (possibly translated)**.**

Oblig: Optional.

Trans: In case of **non_quantitative_code_type** and **non_quantitative_int_type** value names are to be translated

For: **data_type_BSU**s and **data_type_element**s

Mapp: **item_class\dictionary_element.identified_by**
**class_BSU.added_visible_data_types**

NOTE 1 Visible types are inherited.

NOTE 2 Visible types that are not also applicable types may only be referenced to define the data type of the visible properties of the characterization class (or of any of its subclasses).

EXAMPLE A **named_type** "material" may be defined at the level of some characterization class as a **non_quantitative_code_type**. The set of values of this named type consists of a set of codes associated with different (translated) names. This named type may be referenced in some (sub-) characterization class to define the data type of the property that corresponds to the material the products of the characterization class consist of. It may be referenced in some other (sub-) characterization class to define the data type of the property that corresponds to the coating of the products of the characterization class.

### 8.2.7 Applicable Types

Obj: To define which visible types are allowed as data type for the properties applicable to the characterization class (or any of its subclasses).

Descr: The list of the code of the defined or inherited visible types of the characterization class that become applicable types for the characterization class (and any of its subclasses).

Oblig: Optional.

Trans: No translation.

For: A list of **data_type_BSU**s.

Mapp: **item_class\class.defined_types**.

### 8.2.8 Class Valued Properties

Obj: To define which properties may be assigned a single value in the product characterization class where it is declared as class valued, or in any of its subclasses.

Descr: A class valued property enables representation of (1) different perspectives on the set of products that belongs to the current characterization class, and (2) properties that takes the same value for all products belonging to a particular class. When a class valued property is defined at the level of some characterization class, the dictionary user may query all the subclasses of this characterization class for which this class valued property is assigned some value.

Oblig: Optional.

Trans: No translation.

For: A list of **property_BSU**s.

Mapp: **item_class.sub_class_properties**

NOTE Class valued properties are inherited. They do not appear in the **sub_class_properties** list of any subclass.

### 8.2.9 Visible Properties

Obj: To specify the new properties that are defined at the level of this characterization class and may be thus specified as applicable to the characterization class or any of its subclasses (visible properties).

Descr: The new properties (with respect to inheritance) that the products belonging to the non-leaf or leaf characterization class of products may or may not possess according to the subclasses they belong to.

For: A set of **property_BSU**s.

Oblig: Mandatory (possibly empty).

Mapp: **item_class\dictionary_element.identified_by\ class_BSU.added_visible_properties**

### 8.2.10 Applicable Properties

Obj: To specify the new properties that are specified as applicable for that characterization class and for any of its subclasses.

Descr: The new properties (with respect to inheritance) that the products belonging to the non-leaf or leaf characterization class of products shall possess. The properties of this list shall be visible for the characterization class, i.e., they shall be defined as visible either by this characterization class or by a characterization class higher in the hierarchy. The LIST order shall correspond to the default presentation order of the properties in any case where such an order shall be defined.

EXAMPLE To display the properties of some classes on a screen.

For: List of **property_BSU**s.

Oblig: Mandatory (possibly empty).

Mapp: **item_class\class.described_by**

### 8.2.11 Class Constant Values

Obj: To define the values assigned to some class valued properties declared in the characterization class or in any characterization class higher in the hierarchy.

Descr: Class valued properties are intended to be queried by the user to select a set of characterization classes for which such a property has a given value. Once a value is assigned to a class valued property, this value is inherited and shall not be redefined.

For: A set of **class_value_assignment**s.

Oblig: Optional.

Mapp: **item_class.class_constant_values**

### 8.2.12 Definition

Obj: To clarify the meaning of a characterization class or a categorization class by giving its intention textually.

Descr: Statement that describes the meaning of the characterization class and permits its differentiation from all other characterization classes. It shall be a definition in the sense that it shall be complete and unambiguous. All significant words are free from homonymy (homonym) and synonymy.

Oblig: Mandatory.

Trans: To be translated.

For: **class\class_and_property_elements.definition**

### 8.2.13 Source Document of Definition

Obj: A reference to the source document from which a characterization class or a categorization class definition was derived.

Descr: As a minimum the reference shall be given with the document number and date of issue of the document.

Oblig: Optional.

Trans: No translation.

For: Alphanumeric. An identifier of the document.

Mapp: **class\class_and_property_elements.source_doc_of_definition\identified_document.document_identifier**

NOTE In an exchange conforming to ISO 13584-24:2003, the document itself may be exchanged.

### 8.2.14 Note

Obj: To provide further information on any part of the definition, which is essential for the understanding of that definition.

Descr: Free text statement.

Oblig: Optional.

Trans: To be translated.

For: Unlimited alphanumeric string.

Mapp: **item_class\class_and_property_elements.note**

### 8.2.15 Remark

Obj: Explanatory text to further clarify the meaning of the usage of a characterization class or a categorization class of products.

Descr: Free text remarks. It shall not influence the meaning.

Oblig: Optional.

Trans: To be translated.

For: Unlimited alphanumeric string.

Mapp: **item_class\class_and_property_elements.remark**

### 8.2.16 Simplified Drawing

Obj: To provide a visualization, on request of the user, that shows an image of a characterization class or a categorization class of products.

Descr: A drawing including at least the reference coordinate system of the product (that is to be used for all the representations of this product), and the letter symbols of the main applicable properties.

Oblig: Optional.

Trans: No translation.

For: A graphics entity.

Mapp: **item_class.simplified_drawing**

NOTE A graphics entity may be exchanged as an **external_graphics** entity described in this part of ISO 13584. This **external_graphics** may be represented as **http_file**s defined in ISO 13584-24:2003 and referenced by the **ISO13584_IEC61360_dictionary_schema** documented in Annex F.

### 8.2.17 Date of Original Definition

Obj: To show when a characterization class or a categorization class of products was defined by the information supplier and thus when it was declared as valid by this supplier. This date will never be changed and can be used for verification purpose.

Descr: The entry shall be in accordance with ISO 8601.

Oblig: Optional.

Trans: No translation.

For: 10, alphanumeric.

Mapp: **class\dictionary_element.time_stamps.date_of_original_definition**

Expl: 1967-08-20

### 8.2.18 Date of Current Version

Obj: To show the date when the current version was defined.

Descr: The entry shall be in accordance with ISO 8601.

Oblig: Optional.

Trans: No translation.

For: 10, alphanumeric.

Mapp: **class\dictionary_element.time_stamps.date_of_current_version**

### 8.2.19 Date of Current Revision

Obj: To show the date of the last revision number change.

Descr: The entry shall be in accordance with ISO 8601.

Oblig: Optional.

Trans: No translation.

For: 10, alphanumeric.

Mapp: **class\dictionary_element.time_stamps.date_of_current_revision**

### 8.2.20 Version Number

Obj: To characterize each version of a characterization class or a categorization class. A new version number of a class shall be defined when ever a change in the attributes that describes this class influences its use.

NOTE 1 No change is allowed that affects the meaning of a class.

NOTE 2 The changes in a class that affect its version number are defined in Clause 9.

Descr: A string that contains a natural number to indicate the different versions of a characterization class during the life cycle. Version numbers shall be issued in ascending order. A new version number of a characterization class shall be generated according to the definitions given in Clause 9.

Oblig: Mandatory.

For: 10, numeric.

Trans: No translation.

Mapp: **item_class\dictionary_element.identified_by\basic_semantic_unit.version**

### 8.2.21 Revision Number

Obj: To characterize each of a version of a characterization class or a categorization class. A new revision number of a class shall be defined when ever a change in the attributes that describes this class influences neither its meaning nor its use.

NOTE 1 No change is allowed that affects the meaning of a class.

NOTE 2 The changes in a class that affect its revision number are defined in Clause 9.

Descr: A string that contains a natural number used for administrative control of a class. Consecutive revision numbers shall be issued in ascending order for each value of the version number of a class. Per class, unique by its identifier, only one revision number is current at any time. A new revision number of the class shall be generated according to the definitions given in Clause 9. When a new version is issued, the revision is set to '0'.

Oblig: Mandatory.

For: 3, numeric.

Trans: No translation.

Mapp: **item_class\dictionary_element.revision**

### 8.2.22 Constraints

Obj: To restrict the target domains of values of some properties of a characterization class to some subsets of their inherited domains of values.

NOTE 1 These constraints apply to the class where they are defined, and to all its is-a subclasses and case-of classes.

NOTE 2 Each constraint may be associated with an identifier compliant with ISO/TS 29002-5. This identifier consists of a supplier identifier, called RI, a data identifier, called DI, and a version identifier, called VI.

Descr: A set of **constraint**s.

Oblig: Optional.

For: A set of **constraint**s.

Trans: No translation.

Mapp: **item_class\class.constraints**

### 8.2.23 Instance Sharable

Obj: To specify whether an item class is a stand-alone item, or a dependent item that may only exit, in real world, as a component of another item.

NOTE This attribute does not prescribe any specific implementation at the data representation level.

Descr: A Boolean attribute that is defined for an **item_class**.

Oblig: Optional.

For: A Boolean.

Trans: No translation.

Mapp: **item_class.instance_sharable**

### 8.2.24 Categorization Class Superclasses

Obj: To define which categorization classes are one step above a categorization class in a case-of class inclusion hierarchy.

NOTE This attribute applies only to categorization classes.

Descr: Class basic semantic units of the immediate parent categorization classes in a case-of class inclusion hierarchy.

Oblig: Optional.

For: A set of **class_BSU**s.

Trans: No translation.

Mapp: **categorization_class.categorization_class_superclasses**

### 8.2.25 Is Deprecated

Obj: To specify whether a class may still be used or shall no longer be used for new characterizations.

NOTE 1 Deprecated classes may be maintained in a reference dictionary to allow users to interpret characterizations defined with previous versions of this dictionary.

Descr: An optional Boolean that specifies, when true, that a particular class has possibly been used in the past but shall no longer be used for new characterizations.

NOTE 2 When this attribute does not exist, the class is not deprecated.

Oblig: Optional.

For: A Boolean value.

Trans: No translation.

Mapp: **class\dictionary_element.is_deprecated**

#### 8.2.26 Is Deprecated Interpretation

Obj: To specify for a deprecated class the deprecation rationale and how instance values of the deprecated class should be interpreted.

Descr: A note that is mandatory when a class is deprecated.

Oblig: Optional.

For: A translatable text.

Trans: To be translated.

Mapp: **class\dictionary_element.is_deprecated_interpretation**

#### 8.2.27 Administrative Data

Obj: To record information about the life cycle of a class.

Descr: An entity that represents the source language in which a class description was defined, the list of languages in which the class description is translated and the possible status that defines the life cycle state of the class.

Oblig: Optional.

For: An **administrative_data** entity.

Trans: No translation.

Mapp: **class\dictionary_element.administration**

## 9 Dictionary Change Management Rules

This clause defines the rules for organizing, controlling and tracking changes on reference dictionaries.

Given a particular version $O_t$ of a reference dictionary, and a set of product descriptions based on version $O_{t'}$ of this dictionary, the goal of these rules are (1) to permit to decide whether the available $O_t$ dictionary allows to interpret correctly the available product descriptions, based on $O_{t'}$, (2) when it is not the case, to decide which part of the $O_t$ reference dictionary should be updated to allow to interpret correctly the available product descriptions.

NOTE The whole discussion on the dictionary management rules presented in this clause is based on product characterization. Categorization classes being not used in product characterization, they are not concerned in the discussion. Thus, in this clause "class" means "characterization class", and "dictionary element" means "all dictionary elements but categorization classes". The rules for categorization class are defined in Rule 8.

### 9.1 Principle of ontological continuity

The role of a domain ontology in the ISO 13584 series, called a reference dictionary for this domain, is to allow:

— exchange of unambiguous information about products between business partners, and

— storage of stable product characterizations in various persistent repositories.

The method used is to encode each product by a characterization that consists of:



— the characterization class to which the product belongs, and

— a set of property-value pairs, where the properties are selected among the properties that are applicable to this characterization class.

EXAMPLE When a product is represented, using the ISO 13584-511 reference dictionary for fasteners, as a *cap nut* with a *nominal diameter* of 5 and a height of nut of 4; for short:

*cap nut* (*nominal diameter* = 5; *height of nut* = 4),

this characterization represents a product that is an "hexagon nut closed at one side by a flat cap" whose "nominal thread diameter" is 5 mm and whose "overall height of nut" is 4 mm.

NOTE 1 In the computer-to-computer exchange format, both the characterization class and the properties appearing in a characterization are represented using codes that include the version number of these dictionary elements. In the above example, characterization is represented by class and property preferred names to make them understandable.

NOTE 2 In this ontology-based approach, each product is represented as an ontology instance.

The fundamental assumption on which this encoding is based is that:

— in an exchange, both the sender and the receiver must associate the same meaning to the same characterization, and

— a characterization recorded at time = t must be interpreted with the same meaning at time = t+1, even if the reference dictionary evolves between time t and t+1.

In general there are two solutions which allow respecting this fundamental assumption:

— In the case where the reference dictionary has changed between t and t+1 and where there are no restrictions on the allowed changes, a reference dictionary user needs to be able to access both the reference dictionary available at time t and the reference dictionary available at t+1, and thus all the various versions of the reference dictionary.

— Another solution is retained by the dictionary change management rules defined in this part of ISO 13584, which allows to use only one reference dictionary version, namely the more recent version of all its dictionary elements.

This solution, called the *principle of ontological continuity*, restricts the allowed changes to those which ensure that any product characterization defined at time = t with the reference dictionary existing at this time must still keep the same meaning when interpreted with the reference dictionary existing at time = t+1. Consequently the meaning of a dictionary element introduced at some time will be retained in the future.

NOTE 3 Over its lifetime, a reference dictionary description may contain small errors, like typos. It may also need to be refined, for instance to take into account technology improvements. Finally, it also arrives, in some cases, that reference dictionary definitions contain conceptual errors where the meaning of classes and/or properties needs to be changed.

The dictionary change management rules documented in this clause classifies the various changes that may be needed during the lifetime of reference dictionaries. It also specifies how each change should be represented to ensure that the same meaning is always associated with the same existing characterization.

## 9.2 Revisions and Versions

The impact of a change depends upon its impact on existing and future characterizations. We first define what means the fact that a characterization conforms to a reference dictionary.

Let:

— $O_t$ be the reference dictionary O at time t;

— $C_t$ be the classes of the reference dictionary O at time t;

— $P_t$ be the properties of reference dictionary O at time t;

— applicable_properties$_t$ be the function that associates to each class of $C_t$ its applicable properties in $P_t$ at time t;

NOTE 1 applicable_properties$_t$(Class_$c_t$) represents all properties declared by the **described_by** attribute of the **class** entity that defines Class_$c_t$, more, if Class_$c_t$ is a case-of class, the properties imported by the **imported_properties** of Class_$c_t$, more all the applicable properties of the possible superclasses of Class_$c_t$.

— domain$_t$ be the function that associates to each property $Pi_t$ in $P_t$ its domain of values at time t.

NOTE 2 domain$_t$($Pi_t$) represents the domain of values declared by the **domain** attribute of the **property_def** that defines Pi at time t.

A characterization $x_t$ conforms to the reference dictionary $O_t$ if and only if $x_t$ may be represented as an instance of $O_t$. This means that:

— $x_t$ belongs to one class of $C_t$, say Class_$c_t$;

— $x_t$ is characterized by values of some properties, say $P1_t$ , $P2_t$ ,..., $Pn_t$;

— $P1_t$ , $P2_t$ ,..., $Pn_t$ belongs to $P_t$;

— $P1_t$ , $P2_t$ ,..., $Pn_t$ are properties that are applicable to Class_$c_t$;

NOTE 3 $P1_t$ , $P2_t$ ,..., $Pn_t$ may be the set of all properties applicable to Class_$c_t$, it may also be any subset of this set.

— for each property: $P1_t$ , $P2_t$ ,..., $Pn_t$, the value assigned to this property belongs to the domain of values of that property at time t, as defined by the function: domain$_t$($Pi_t$), i = 1..n.

Formally, $x_t$ conforms to the reference dictionary $O_t$ if a dictionary user may encode it:

$$x_t = \text{Class\_}c_t\ (P1_t = v1,\ P2_t = v2, \ldots,\ Pn_t = vn)$$

With:

$$\text{Class\_}c_t \in O_t,\ P1_t \in O_t\ ,\ P2_t \in O_t,\ \ldots,\ Pn_t \in O_t$$

$$\wedge\ P1_t \in \text{applicable\_properties}_t(\text{Class\_}c_t) \wedge P2_t \in \text{applicable\_properties}_t(\text{Class\_}c_t)$$

$$\wedge \ldots \wedge Pn_t \in \text{applicable\_properties}_t(\text{Class\_}c_t)$$

$$\wedge\ v1 \in \text{domain}_t(P1_t) \wedge v2 \in \text{domain}_t(P2_t) \wedge \ldots \wedge vn \in \text{domain}_t(Pn_t)$$

The set of all the characterizations $x_t$ that conform to $O_t$, called the *population* $Pop_t$ of $O_t$, is defined as follows:

$$Pop_t = \text{all } x_t \text{ such that } x_t \text{ conforms to } O_t$$

We said that:

— $Pop_t$ and $x_t$ *conform* to $O_t$, and

— $O_t$ *interprets* $Pop_t$ and $x_t$.

These definitions allow to classify the various changes of reference dictionaries.

The first kind of changes in a reference dictionary are those changes that do not modify at all the set of characterizations that may be defined by this reference dictionary, i.e., the population of the reference dictionary. It is the case, for instance, when a typo is corrected, when new translations are added or when the definition of a class is redrafted to make its content clearer without changing its meaning. In this case, $Pop_t$, the population of $O_t$ at time = t, is identical to $Pop_{t+1}$, the population of $O_{t+1}$ at time = t+1. This means that:

— any characterization $x_t$ defined from $O_t$ also *conforms* to $O_{t+1}$. Thus, $O_{t+1}$ is *backward compatible* with $O_t$ since it allows to *interpret* all its instances, moreover

— any characterization $x_{t+1}$ defined from $O_{t+1}$ also *conforms* to $O_t$. Thus $O_{t+1}$ is also *upward compatible* with $O_t$ since $O_t$ allows to *interpret* all $O_{t+1}$ instances.

In case of a change for which backward and upward compatibility are existing, it is not necessary to record if the characterization x was built at time = t or at time = t + 1. Thus this change would not require to change the version numbers that were already assigned to the various dictionary elements at time = t. This change is called a *revision change*, and it will be traced by increasing either the **revision** attribute of the dictionary element that was modified if the change affects the description in the source language in which the dictionary element was defined, and/or one or several **translation_revision** attributes corresponding to the other languages in which the dictionary element is translated if the change affects the description in the corresponding language.

NOTE 4 Revision numbers are not recorded in the identifiers of dictionary elements. The characterization using the dictionary elements will allow to use $O_t$ as well as $O_{t+1}$ for interpretation.

NOTE 5 Each dictionary element has a revision attribute. Each dictionary element that is translated has an **administrative_data** element that specifies both the **source_language** language in which the **dictionary_element** was initially defined, and, for each translation, a **translation_revision** attribute

The second kind of changes in a reference dictionary are those changes that refine the reference dictionary and allow to define new characterizations. New classes are introduced, new properties are introduced, and new property values are added to their domain of values. To ensure the ontological continuity principle, no class, property or value should be removed. The reference dictionary $O_{t+1}$ defined after the change shall remain able to interpret $Pop_t$. $O_{t+1}$ is still *backward compatible* with $O_t$ and it allows to interpret all its instances. But it is no longer *upward compatible* because some characterizations that conform to $O_{t+1}$ do not conform to $O_t$.

In case of changes for which only backward compatibility is existing, a characterization x that was built at time = t+1 and depends upon the modified dictionary elements should express this clearly in its representation. This change is called a *version change* and it will be traced by increasing the **version** of the dictionary element that was modified, and of all the other dictionary elements that were also modified as a consequence.

NOTE 6 Version numbers are recorded in the identifiers of dictionary elements, the version of each element used in a characterization will prevent to use $O_t$ for trying to interpret a characterization based on $O_{t+1}$ specific versions.

Table 1 summarizes the differences between version and revision.

**Table 1 — Revision and version**

| | **Backward compatibility** $Pop_t$ *conforms to* $O_{t+1}$ | **Upward compatibility** $Pop_{t+1}$ *conforms to* $O_t$ |
|---|---|---|
| Revision | Yes | Yes |
| Version | Yes | No |

## 9.3 Correction of errors

In case of errors in a reference dictionary which is already used to define product characterizations, the need is to correct the reference dictionary errors but also to provide a mechanism that allows reference dictionary users to understand and process the error correction. For each data set that contains product characterization, processing errors means (1) recognizing which characterizations are in errors and (2) defining how erroneous characterizations should be corrected to be in line with the corrected reference dictionary.

When the dictionary elements which are erroneous have not yet been used for creating product characterizations or if in a closed user environment, the characterizations can be corrected concurrently with the reference dictionary. It is the responsibility of the reference dictionary supplier to decide how to remove erroneous elements from the current reference dictionary, and how to perform the reference dictionary correction.

In the dictionary change management rules defined in this part of ISO 13584, an open user environment is assumed, where all possible characterizations are not accessible to the dictionary supplier, and correction cannot be performed together with the reference dictionary. In such an environment, a mechanism called "deprecation" has to be used.

Deprecation means that:

— to ensure backward compatibility, the erroneous dictionary elements and/or the erroneous property values remain in the reference dictionary to ensure backward compatibility, but

— all the erroneous elements are associated with a **is_deprecated** attribute with a true value, the meaning of which being: "this dictionary element or value shall no longer be used for new characterizations", and

— an attribute associated with each **is_deprecated** attribute, called **is_deprecated_interpretation**, is used to specify how a characterization that references deprecated elements should be changed to be in line with the up-dated reference dictionary.

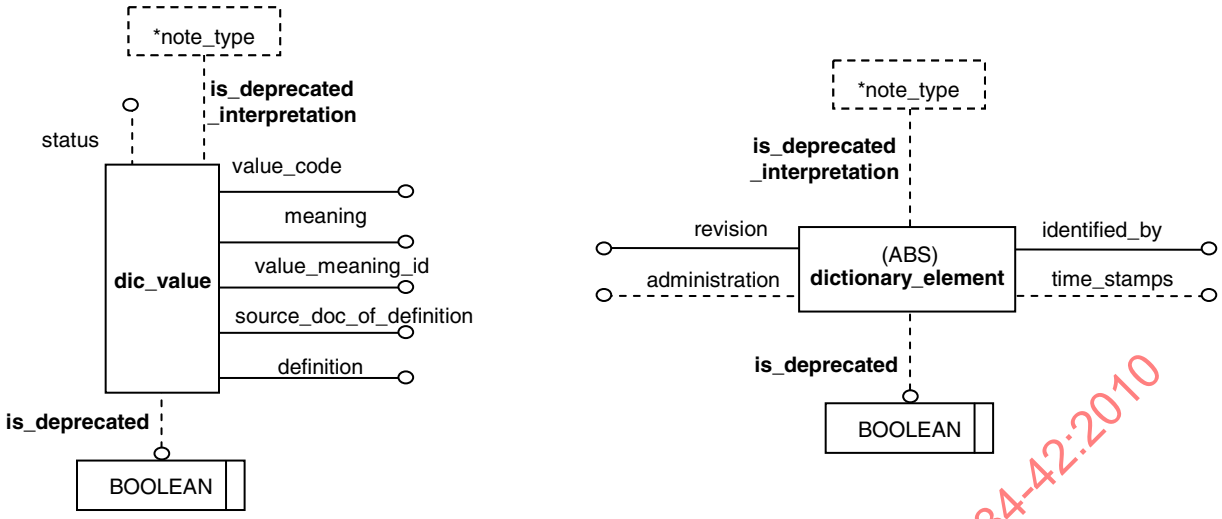
NOTE 1 The specification in **is_deprecated_interpretation** may be either informal, to explain to a reference dictionary user how the corresponding data should be processed, or formal to direct a computer how to correct automatically the data.

NOTE 2 in the current specification of the dictionary change management rules, no formal language is defined for representing the content of **is_deprecated_interpretation**. It is the intent of the team that developed the common ISO13584/IEC61360 dictionary model to consider the development of such a language.

EXAMPLE 1 If in a class C1 an applicable property P1 whose value was supposed to be expressed in meters, is replaced by a property P2 which has the same meaning but whose value shall be expressed in microns, (1) the P1 **is_deprecated** attribute is set to true, and (2) its **is_deprecated_interpretation** attribute could be set to: "the value of this property shall now be expressed in microns and recorded in property P2".

EXAMPLE 2 In example 1 above, the value of the **is_deprecated_interpretation** attribute of P1 could be represented, if this approach has been agreed by the community that uses the reference dictionary, as an expression using the syntax of the EXPRESS language, and representing the values of properties by the property identifiers. In this case the content could be set to: "P2 := P1 * 1 000".

Figure 1 presents a planning model of the representation of this mechanism in the common ISO13584/IEC61360 dictionary model, together with the definition of the relevant attributes.




Attribute definitions:
**is_deprecated**: an optional Boolean. When true, it specifies that the **dic_value/dictionary_element** shall no longer be used.
**is_deprecated_interpretation**: specifies the deprecation rationale and how instance values of the deprecated element should be interpreted.

**Figure 1 — Information model of deprecated elements**

## 9.4 Rules for change management

This subclause provides rules for managing changes in reference dictionaries.

### 9.4.1 Criteria for classifying a change

The impact of a change in a dictionary element onto the populations of characterizations that are interpretable by $O_t$ and/or $O_{t+1}$ provides a criteria for classifying the change impact as a revision change, a version change or an error correction. This clause describes the how each change should, at least, be recorded according to its impact to ensure that the receiver of an exchange file that contains item characterizations will understand whether its current dictionary allows to interpret the exchange file or not.

These rules define the minimal requirements. But a reference dictionary supplier may always decide to update the version of a dictionary element when the rules request only the updating of the revision, or to deprecate a modified element when the rules request only the updating of its revision or of its version.

**Rule 1: Revision Change**

If after changing an entity (class, datatype, properties, name, definition…) $Ent_t$ of the reference dictionary $O_t$ into $Ent_{t+1}$, (1) the new reference dictionary $O_{t+1}$ may interpret all the characterizations that might be defined by $O_t$, and (2) it does not allow to define any new characterization, the change is a revision change of the dictionary element changed by the change of the $Ent_t$ entity.

If the description of the dictionary element changed is only defined in a single language, or if it is translated and the change affects the description in the source language in which it was defined, the change should increase the value of the **revision** attribute of the dictionary element modified by the change. If the change of Ent also affects the translation in any other languages in which the dictionary element is translated, the corresponding translations should be changed, and the change should increase the values of the **translation_revision** attribute of the corresponding translations.

EXAMPLE 1 In a dictionary that is available only in a single language, if one changes the definition of a class without changing the characterizations it can interpret, the revision attribute of the class must be increased.

EXAMPLE 2 In a dictionary whose source language is English and that is translated in German and French, if one changes the French definition of a class without changing the characterizations it can interpret, the **translation_revision** attribute of the French translation should be increased.

EXAMPLE 3 In a dictionary whose source language is English and that is translated in German and French, if one changes the value of the **figure** attribute of a class without changing the characterizations the class can interpret, then the **revision** attribute of the class should be increased. If the **figure** value is a **graphic_files** that is not language-dependent, and thus applies to all language descriptions, the German and French **translation_revision** should not be increased because the translated part was not changed.

EXAMPLE 4 If one adds a visible property to a class without making it applicable in the class or any of its subclass, then no characterization may be described by this new property. No direct attribute of the class being modified, neither the revision nor the version of the class needs to be updated.

**Rule 2: Version Change**

If after changing an entity (class, datatype, properties, …) $Ent_t$ of the reference dictionary $O_t$ into $Ent_{t+1}$, (1) the new reference dictionary $O_{t+1}$ may interpret all the characterizations that might be defined by $O_t$, but (2) it also provides new characterizations that cannot be interpreted by $O_t$, the change should increase the version of $Ent_t$.

NOTE 1 Constraints have an impact on those item characterizations that fulfil the constraints. Thus, change of constraints should be represented by an increase of version of the class that contains these constraints. But change of constraint does not change the set of characterization that may be interpreted by a dictionary. Thus, when constraints are modified in a class, the set of item characterizations that fulfil the constraints may become broader or narrower without violating the ontological continuity principle.

EXAMPLE 5 If one adds an applicable property to a class, thus (1) all characterizations defined by the previous reference dictionary may still be interpreted (without using the new applicable property), but (2) some characterizations may also be defined by the new reference dictionary that could not be interpreted by the previous one (those that use the new applicable property). Thus the version of the class must be increased.

EXAMPLE 6 If one adds a new alternative unit to the real measure type of some property, thus all characterizations defined by the previous reference dictionary may still be interpreted (without using the new alternative unit), but some characterizations may also be defined by the new reference dictionary that could not be interpreted by the previous one (by using the new alternative unit). Thus the version of the class must be increased.

**Rule 3: Error correction**

If after changing a dictionary element (class, datatype, properties, …) $Ent_t$ of the reference dictionary into $Ent_{t+1}$, the new reference dictionary $O_{t+1}$ is not able to interpret all the characterizations that might be defined by $O_t$, the change would not be backward compatible and is violating the principle of ontological continuity.

For allowing error correction, it is needed to (1) identify those dictionary elements that should be modified and assign the value true to their **is_deprecated** attribute, (2) define new entities which would correct the errors, and (3) describe in the **is_deprecated_interpretation** attribute of the deprecated elements, why the element was deprecated and how a characterization that references deprecated elements should be changed to be in line with the up-dated reference dictionary.

EXAMPLE 7 If one corrects the unit of the real measure type of some property, then all products characterized by this property that are already recorded somewhere should be described differently. The new reference dictionary would not be backward compatible and this change could not be done that way. Thus (1) the corresponding property should be deprecated, (2) a new property (with a different identifier) should be created and made visible and applicable where the previous one was so, and (3) in the **is_deprecated_interpretation** of the previous property it should be specified that its value should be e.g., "divided by 1000, and then put as value to the new property".

EXAMPLE 8 If one adds some new context parameters to those from which a context dependent property depends on, thus all characterizations that involve this property should be described differently after the change. The new reference dictionary would no longer be able to interpret some previous characterizations. Thus (1) deprecate the old context dependent property, (2) create and introduce a new one, and (3) explain the deprecation rational, and possibly, if the previous context dependent property was supposed to be measured at a fixed value of the new context parameter, how values of the deprecated context dependent property could be converted into value of the new property.

NOTE 2 In this example, it would also be possible to keep in the reference dictionary both the previous and the new context dependent property.

Figure 2 summarizes how to classify a change.

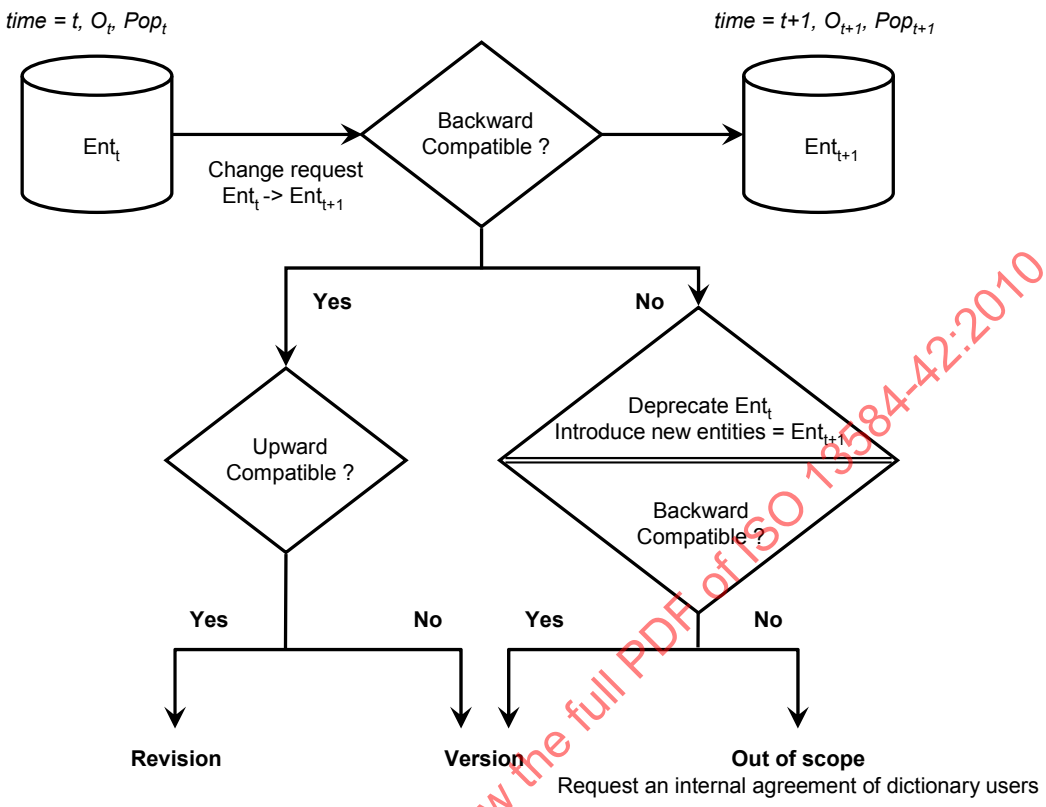



**Figure 2 — Classifying a dictionary change**

### 9.4.2 Dependency and the propagation of changes

In a reference dictionary, each dictionary element may exist only in a single version. Thus, when the version number of a dictionary element of a reference dictionary is increased, all the dictionary elements that reference this dictionary element should be changed to reference its new version. Indeed, such a change should be traced at the level of the identifiers of all the referencing dictionary elements in order to be sure that, when a dictionary element identifier is replaced by its dictionary element description, it contains the correct internal references. Thus every change in the version of a dictionary element referenced within another dictionary element must be represented by a new version of the latter.

The oneness of each dictionary element applies within a reference dictionary, but not between several dictionaries. When class C1 of reference dictionary D1 references class C2 of reference dictionary D2 by the case-of relationship, it is the responsibility of the D1 reference dictionary supplier to decide which classes are referenced, in which version, and possibly which properties are imported. Thus, if the dictionary supplier of D2 changes the version of C2, it is the responsibility of the D1 dictionary supplier to decide if and when the new version of C2 will be referenced in C1. The old reference may be kept. But if the version of the reference is increased, then the version of C1 shall be increased.

This is summarized in the four rules below

**Rule 4: No propagation between reference dictionaries**

When a class C1 of the reference dictionary D1 references a class C2 of the reference dictionary D2 by the case-of relationship, and when the version of class C2 is upgraded, it is the responsibility of the D1 dictionary supplier to decide if and when C1 will reference the new version of C2. If this is done, the version of C1 shall be increased.

EXAMPLE 1 C1 import, through case-of, property P1 and P2. A new applicable property P3 is added to C2. The supplier of D1 is not interested in P3. C1 may continue referencing the previous version of C2.

**Rule 5: Version propagation**

Increasing the version number of any dictionary element that is referenced by other dictionary elements of the same reference dictionary must be propagated to them.

NOTE 1 Same reference dictionary means identified by the same information supplier.

NOTE 2 When a dictionary element references another dictionary element, this reference is done through a BSU that includes the version of the target dictionary element. Thus, if the version of the target dictionary element is changed, the content of the source dictionary element should also be changed to record the correct (new) reference. This change induces that new characterization could be described (indirectly) by the source dictionary element and that its version should be changed.

EXAMPLE 2 Changing the version of a **named_type**, for instance to extend its domain of values, change also the domain of values of any property that reference it as its **domain**. Thus the version of these properties should also be updated. This would also change the set of characterizations that may be described by the classes where these properties become applicable by the **described_by** attribute.

EXAMPLE 3 Change of the version number of a class leads to a change of the version number of its sub-classes and of the subclasses of this class, and so on.

EXAMPLE 4 If the version of the definition class of a property is increased, the version of this property must also be increased.

NOTE 3 This rule ensure that when the version number of the characterization class used for characterizing an item in an exchange file is smaller than or equal to its version in the local dictionary of the receiving system, This system is able to interpret correctly this characterization, whatever be its complexity.

**Rule 6: Computation of new version values**

For each particular change, all the propagated changes shall be computed together and the version number of each entity shall be increased at most once. It is also allowed to gather a number of different changes to compute new versions of a set of dictionary elements.

**Rule 7: Circulation of new version**

It is the responsibility of the dictionary supplier that provides reference dictionaries to decide how and when updates should be distributed.

EXAMPLE 5 A reference dictionary may be associated with a server compliant with ISO/TS 29002-20 that makes available each update as soon as it has been validated.

EXAMPLE 6 A reference dictionary may be distributed by releases. Every year a new release integrates all the updates elaborated during the year. In this case, all modified dictionary elements may have only one version increased.

### 9.4.3 Management of categorization classes

Categorization classes having no impact on item characterization, the above rules cannot apply to them.

**Rule 8: Versioning of categorization classes**

Increasing versions of categorization classes are requested when one or several of their superclasses, referenced by the **categorization_class_superclasses**, attribute, are changed. All other changes may be recorded by revision increasing.

Change of versions of categorization classes are not propagated to characterization classes that reference them. Such changes are only recorded as revision changes.

#### 9.4.4 Management of dictionary version and revision

During a file exchange of item characterizations based on a dictionary, the following rule ensures that in a file exchange, just by checking the dictionary version, the file receiver may know whether his/her available version of the dictionary allows to interpret the file

**Rule 9: Versions and revisions of a dictionary**

When an updated dictionary is distributed according to rule 6:

— if the **version** of any dictionary element defined in this dictionary has been incremented, and/or if a new dictionary element has been introduced, the **version** of the dictionary shall be incremented;

EXAMPLE A new dictionary element is introduced in the dictionary when a new subclass of an existing class is introduced, or when a new visible type is defined.

— if the **revision** of any dictionary element defined in this dictionary has been incremented, but the version of the dictionary is not changed, then the **revision** of the dictionary shall be incremented;

— if the **version** of the dictionary is incremented, its **revision** shall be reset to '0'.

### 9.5 Dictionary Changes and Attributes

#### 9.5.1 System maintained attributes

Dictionary change management rules are restricted to attributes that are available for change triggered by user change requests. System maintained attributes are therefore out of scope for reference dictionary changes, since they are modified automatically as consequence of another change:

— Revision change: if the change affects the description in the source language in which the dictionary element was described, **revision** is incremented and **date_of_current_revision** is updated with the current time. If the change affects the description in one, or several, of other languages in which the description of the dictionary element is translated, **translation_revision** corresponding to this language is incremented and **date_of_current_translation_revision** corresponding to this language is updated with the current time.

— Version change: **version** is incremented and **date_of_current_version** is updated with the current time. **revision** is set to the value defined as the minimum of **revision** attribute and **date_of_current_revision** is updated with the modification time.

— Creation of a new dictionary element: a new dictionary element is created with a new **code** and **date_of_original_definition** is set to the current time. **Version** is set to the value defined as the minimum of **version** attribute and **date_of_current_version** is updated with the current time. **Revision** is set to the value defined as the minimum of revision number and **date_of_current_revision** is updated with the current time.

#### 9.5.2 Attributes available for textual change

The role of the terminological attributes of dictionary elements, such that names, definition, note, icon, is to explain which kind of products are characterized by a particular reference dictionary class, and which kind of product characteristic is represented by each particular property.

To ensure backward compatibility for textual changes of terminological attributes, such changes should not reduce the meaning of the class or of the property, even if it may precise its meaning. But a textual change may enlarge the definition of the class, for instance to take into account the development of new products.

Thus, a textual change requires, at least, a new revision number. But it is the responsibility of the dictionary supplier to decide whether this change should also change the version of the dictionary element to ensure that dictionary users will access to the terminological attributes that were used when some characterization was defined. It is also the responsibility of the dictionary supplier to decide whether a new dictionary element

should be defined associated with a new code because the new definition seems to be not backward compatible with the previous one.

## 9.6 Constraints on the evolution of reference dictionaries

In this clause, we summarize the constraints for each kind of concept (classes, relation between classes, properties and characterizations) during reference dictionary evolution.

**Permanence of the classes**

Existence of a characterization class could not be denied across evolution. Because each characterization class allows to define some characterizations, any class existing at time t, shall still exist at t', with t' > t.

NOTE 1 To make the change management model more flexible, a class may become obsolete. It will then be marked as ”deprecated”, and possibly replaced by another class. But it will continue belonging to the newer versions of the reference dictionary.

This principle allows that the most recent reference dictionary will be able to interpret all the earlier-defined characterizations. It is the responsibility of each reference dictionary user to decide if, and until when, deprecated elements will be kept in each user dictionary.

The problem of permanence is different for categorization classes. Since categorization classes are not used for defining characterizations, these classes may be suppressed or modified without creating a backward compatibility problem.

**Permanence of properties**

Similarly, all properties existing at time t shall still exist at t', t' > t. A property may also become obsolete but neither its existence, nor its value for a particular item may be modified. The value domain of a property may evolve. Taking into account the backward compatibility requirement, a value domain can only increase, certain values being eventually marked as deprecated.

**Permanence of the class-subclass relationship**

The class-subclass relationship is the relation between a class, and all its subclasses, direct or obtained by transitivity. The class-subclass relationship supports inheritance between the superclass and the subclasses. Requirement for permanence of a particular class-subclass relationship between two classes C1, as superclass, and C2, as subclass, depends upon the consequences of this relation for the characterizations defined by the subclass:

— if C2 does not inherit from C1 any element (property, type, value,...) that may be used in a characterization, then the C1-C2 relationship may be suppressed. Consequences in version and revision number are defined by the dictionary change management rules.

— if C2 inherits from C1 some elements (property, type, value,...) that may be used in a characterization of a C2 instance, then the C1-C2 relationship shall not be suppressed.

Note that this constraint allows large evolution of the class-subclass relationship hierarchy, for example by intercalating intermediate classes between two classes linked by a class-subclass relation.

**Permanence of Characterizations**

The fact that a property P is applicable to a class C at time t requests that P remains applicable to C at t', t' > t.

NOTE 2 This does not require at all that the same applicable properties are always used to describe the instances of the same class. Properties used to characterize an item do not depend on reference dictionary evolutions. It depends mainly of the requirements of the application that uses the reference dictionary.

NOTE 3 If a property P1 is declared as applicable in class C2 which is a subclass of C1, P1 may become applicable in class C1 without any backward compatibility problem because applicability is inherited.

# Annex A
# (normative)

# Survey of type classification codes of non-quantitative data element types (main class A)

**Table A.1 — Survey of type classification codes of non-quantitative data element types (main class A)**

| Type classification code | Description |
|---|---|
| A11 | Geographical unit (greater than a place) |
| A12 | Geographical location (place or smaller) |
| A13 | Geographical route and network |
| A21 | Organization |
| A22 | Functionary |
| A31 | Date and time period |
| A32 | Time of day |
| A41 | Private person |
| A51 | Product |
| A52 | Product class |
| A53 | Product batch and package (type) |
| A54 | Transport mode, means and unit |
| A55 | Manufacturing process and technology |
| A56 | Product function and application |
| A57 | Material |
| A58 | Product geometry, shape, and size |
| A59 | Product quality, performance, and test |
| A61 | Document and message |
| A62 | Information element and information group |
| A63 | Data medium and transmission unit |
| A71 | Measuring unit |
| A79 | Type of measurement |

**Table A.1 (Continued)**

| Type classification code | Description |
|---|---|
| A81 | Account |
| A82 | Project, project activity |
| A83 | Procedure |
| A91 | Abstract identification such as language, color, etc. |
| A93 | Clause |

# Annex B
# (normative)

# Short names of entities

Table B.1 provides the short names of entities specified in this part of ISO 13584. Requirements on the use of short names are found in the implementation methods included in ISO 10303

**Table B.1 — Short names of entities**

| Long name | Short name |
|---|---|
| A_PRIORI_SEMANTIC_RELATIONSHIP | APS1 |
| ADMINISTRATIVE_DATA | ADMDT |
| AXIS1_PLACEMENT_TYPE | AXPLTY |
| AXIS2_PLACEMENT_2D_TYPE | AP2T |
| AXIS2_PLACEMENT_3D_TYPE | AP3T |
| BASIC_SEMANTIC_UNIT | BSSMUN |
| BOOLEAN_TYPE | BLNTYP |
| CARDINALITY_CONSTRAINT | CRDCNS |
| CATEGORIZATION_CLASS | CTGCLS |
| CLASS | CLASS |
| CLASS_AND_PROPERTY_ELEMENTS | CAPE |
| CLASS_BSU | CLSBS |
| CLASS_BSU_RELATIONSHIP | CLBSRL |
| CLASS_CONSTRAINT | CLSCNS |
| CLASS_REFERENCE_TYPE | CLRFTY |
| CLASS_RELATED_BSU | CLRLBS |
| CLASS_VALUE_ASSIGNMENT | CLVLAS |
| COMPLEX_TYPE | CMPTYP |
| CONDITION_DET | CNDDT |
| CONFIGURATION_CONTROL_CONSTRAINT | CNCNCN |
| CONSTRAINT | CNSTRN |
| CONTENT_ITEM | CNTITM |
| CONTEXT_RESTRICTION_CONSTRAINT | CNRSCN |
| DATA_TYPE | DTTYP |
| DATA_TYPE_BSU | DTTYBS |
| DATA_TYPE_ELEMENT | DTTYEL |

Table B.1 (continued)

| Long name | Short name |
|---|---|
| DATE_DATA_TYPE | DTDTTY |
| DATE_TIME_DATA_TYPE | DTDT |
| DATES | DATES |
| DEPENDENT_P_DET | DPPDT |
| DIC_UNIT | DCUNT |
| DIC_VALUE | DCVL |
| DICTIONARY_ELEMENT | DCTELM |
| DICTIONARY_IDENTIFICATION | DCTIDN |
| DOCUMENT | DCMNT |
| DOMAIN_CONSTRAINT | DMNCNS |
| ENTITY_INSTANCE_TYPE | ENINTY |
| ENTITY_SUBTYPE_CONSTRAINT | ENSBCN |
| ENUMERATION_CONSTRAINT | ENMCNS |
| EXTERNAL_GRAPHICS | EXTGRP |
| FILTER | FILTER |
| GLOBAL_LANGUAGE_ASSIGNMENT | GLLNAS |
| GRAPHIC_FILES | GRPFLS |
| GRAPHICS | GRPHCS |
| IDENTIFIED_DOCUMENT | IDNDCM |
| INT_CURRENCY_TYPE | INCRTY |
| INT_MEASURE_TYPE | INMSTY |
| INT_TYPE | INTTYP |
| INTEGRITY_CONSTRAINT | INTCNS |
| ITEM_CLASS | ITMCLS |
| ITEM_CLASS_CASE_OF | ICCO |
| ITEM_NAMES | ITMNMS |
| LABEL_WITH_LANGUAGE | LBWTLN |
| LANGUAGE_CODE | LNGCD |
| LEVEL_TYPE | LVLTYP |
| MATHEMATICAL_STRING | MTHSTR |
| NAMED_TYPE | NMDTYP |
| NON_DEPENDENT_P_DET | NDPD |
| NON_QUANTITATIVE_CODE_TYPE | NQCT |

Table B.1 (continued)

| Long name | Short name |
|---|---|
| NON_QUANTITATIVE_INT_TYPE | NQIT |
| NON_SI_UNIT | NNSUN |
| NON_TRANSLATABLE_STRING_TYPE | NTST |
| NUMBER_TYPE | NMBTYP |
| PLACEMENT_TYPE | PLCTYP |
| PRESENT_TRANSLATIONS | PRSTRN |
| PROPERTY_BSU | PRPBS |
| PROPERTY_CONSTRAINT | PRPCNS |
| PROPERTY_DET | PRPDT |
| RANGE_CONSTRAINT | RNGCNS |
| RATIONAL_MEASURE_TYPE | RTMSTY |
| RATIONAL_TYPE | RTNTYP |
| REAL_CURRENCY_TYPE | RLCRTY |
| REAL_MEASURE_TYPE | RLMSTY |
| REAL_TYPE | RLTYP |
| SIMPLE_TYPE | SMPTYP |
| STRING_PATTERN_CONSTRAINT | STPTCN |
| STRING_SIZE_CONSTRAINT | STSZCN |
| STRING_TYPE | STRTYP |
| SUBCLASS_CONSTRAINT | SBCCNS |
| SUPPLIER_BSU | SPPBS |
| SUPPLIER_BSU_RELATIONSHIP | SPBSRL |
| SUPPLIER_ELEMENT | SPPELM |
| SUPPLIER_RELATED_BSU | SPRLBS |
| TIME_DATA_TYPE | TMDTTY |
| TRANSLATABLE_STRING_TYPE | TRSTTY |
| TRANSLATED_LABEL | TRNLBL |
| TRANSLATED_TEXT | TRNTXT |
| TRANSLATION_DATA | TRNDT |
| URI_TYPE | URTYP |
| VALUE_DOMAIN | VLDMN |

# Annex C
# (normative)

# Computer interpretable listings

This annex references listings for the set of EXPRESS schemas documented in this part of ISO 13584. These schemas, without comments or explanatory text, are intended to be used as resources for the other parts of ISO 13584. It also contains the long form schema of these shemas. This long form schema, called ISO13584_P42_2_LONG_FORM_SCHEMA, may be used for describing and exchanging simple dictionaries. All these schemata may be downloaded at:

http://standards.iso.org/iso/13584/-42/EXPRESS

If there is difficulty accessing this site contact ISO Central Secretariat or contact the ISO TC 184/SC4 Secretariat directly at: sc4sec@tc184-sc4.org.

NOTE ISO 13584-32, known as OntoML, is an XML schema that is based on the mechanisms defined in this part of ISO 13584. It provides capabilities for exchanging more complex dictionaries that require collection-valued properties and functional models and views.

The following notice applies to the computer-interpretable files in this annex.





Table C.1 below, describes the URI of the schemas whose computer interpretable listings are provided by this annex.

**Table C.1 — EXPRESS schemas documented in this part of ISO 13584**

| Description | URI |
|---|---|
| ISO13584_IEC61360_dictionary_schema | urn:iso:std:iso:13584:-42:ed-2:tech:express:dictionary |
| ISO13584_IEC61360_language_resource_schema | urn:iso:std:iso:13584:-42:ed-2:tech:express:language |
| ISO13584_IEC61360_class_constraint_schema | urn:iso:std:iso:13584:-42:ed-2:tech:express:constraint |
| ISO13584_IEC61360_item_class_case_of_schema | urn:iso:std:iso:13584:-42:ed-2:tech:express:caseof |
| ISO13584_P42_2_LONG_FORM_SCHEMA | urn:iso:std:iso:13584:-42:ed-2:tech:express:dictionary-long-form |

NOTE Additional information may be provided to support implementation. It would be available at the URL described in Annex K.

# Annex D
# (normative)

# Value format specification

This part of ISO 13584 and IEC 61360-2 provide a particular syntax to specify the allowed formats for the string and numeric values that may be associated with a property.

EXAMPLE 1 The format NR1 3 allows to specify that only integer values consisting of exactly three digits are allowed.

NOTE 1 No value format is defined for any other **data_type**, including **boolean_type**.

NOTE 2 In this part of ISO 13584, to define the format of property values is not mandatory.

The syntax of the allowed formats is defined in this Annex using a subset of the Extended Backus-Naur Form (EBNF) defined in ISO/IEC 14977.

EXAMPLE 2 The syntax of the format NR1 3 are the letters 'NR1' '' '3'.

The meaning of each syntax, that is the characters that may be used to represent a value, cannot be defined using the EBNF. Thus the meaning of each part of the format concerning the characters allowed to represent the value is specified separately for each part of the format.

EXAMPLE 3 The syntax of the format NR1 3 has the following meaning: *NR1* means that only an integer value may be represented. Space means that a fixed number of characters is specified by the format. *3* means that exactly three digits are required.

## D.1 Notation

Table D.1 summarizes the subset of the ISO/IEC 14977 EBNF syntactic metalanguage used by this part of ISO 13584 / IEC 61360-2 to specify value format of properties.

Using these notations, the syntax of the subset of the EBNF metalanguage used by this part of ISO 13584 / IEC 61360-2 to specify value format of properties is summarized by the following grammar (the meta-identifier character, letter and digit are not detailed):

```
syntax = syntaxrule ,{ syntaxrule } ;
syntaxrule = metaidentifier , '=' , definitionslist , ';' ;
definitionslist = singledefinition , { '|' , singledefinition } ;
singledefinition = term , { ',', term } ;
term = primary, [ '-', primary } ;
primary = optionalsequence | repeatedsequence | groupedsequence |
          metaidentifier | terminal | empty ;
optionalsequence = '[' definitionslist ']' ;
repeatedsequence = '{' definitionslist '}' ;
groupedsequence = '(' definitionslist ')' ;
metaidentifier = letter , { letter ] ;
terminal = "'", (character - "'"), {character - "'"}, "'"
           | '"', (character - '"'),{character - '"'}, '"' ;
empty = ;
```

The equal sign '=' indicates a syntax rule. The meta-identifier on the left may be re-written by the combination of the elements on the right. Any spaces appearing between the elements are meaningless unless they appear within a `terminal`. A syntax rule is terminated by a semicolon ';'.

**Table D.1 — ISO/IEC 14977 EBNF syntactic metalanguage**

| Representation | ISO/IEC 10646-1 Character names | Metalanguage symbol and role |
|---|---|---|
| ' ' | apostrophe | First quote symbol: represents language terminals.<br>Terminal shall not contain apostrophe.<br>Example: 'Hello' |
| " " | quotation mark | Second quote symbol: represents language terminals.<br>Terminal shall not contain quotation mark.<br>Example: "John's car" |
| ( ) | left parenthesis, right parenthesis | Start / end group symbols.<br>The content is considered as a single symbol. |
| [ ] | left square bracket, right square bracket | Start / end option symbols.<br>The content may or not be present. |
| { } | left curly bracket, right curly bracket | Start/ end repeat symbols.<br>The content may be present 0 to n times. |
| - | hyphen-minus | Except symbol. |
| , | comma | concatenate symbol. |
| = | equals sign | Defining symbol.<br>Syntax rule: defines the symbol of the left by the formula on the right. |
| \| | vertical line | Alternative separator symbol. |
| ; | semicolon | Terminator symbol.<br>End of a syntax rule. |



The use of a meta-identifier within a definition-list denotes a non-terminal symbol which appears on the left side of another syntax rule. A meta-identifier is composed of letters or digits, the first character being a letter. If a term contains both a `primary` preceding a minus sign, and a `primary` that follows the minus sign, only the sequence of symbols that are represented by the first `primary` and that are not represented by the second `primary` are represented by the term.

EXAMPLE 1 Notation:

"'", character – "'", "'"

means any character but the apostrophe character, inserted between two apostrophe characters.

The `terminal` denotes a symbol which cannot be expanded further by a syntax rule, and which will appear in the final result. Two ways are allowed to represent a `terminal`: either a set of characters without apostrophe, inserted between two apostrophes, or a set of characters without quotation marks, inserted between two quotation marks.

EXAMPLE 2 Assume that we want to describe, by such a grammar, the price of a product in €. Such a price is a positive number with no more than 2 digits in the cents part. We introduce three meta-identifiers associated with three syntax rules:

```
digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
cents = [ '.' , digit [ , digit ] ];
euros = digit { , digit } cents;
```

With these syntax rules: 012, 4323.3, 3.56 are examples of licit representations of Euros. 12., .10 are examples of non licit representation of Euros.

## D.2 Data value format types

The grammar defined in this annex defines eight different types of value formats: four quantitative and five non-quantitative value formats.

In the next clause, we define the meta-identifiers that are used to specify these formats. In Clause D.4 we define the syntax rule for the four meta-identifiers that represent the four quantitative value formats, together with their meaning at the value level. In Clause D.5 we define the meta-identifiers for the five non-quantitative value formats, together with their meaning at the value level.

## D.3 Meta-identifier used to define the formats

The meta-identifiers used in the grammar that define the various value formats are the following:

```
dot = '.';
decimalMark = '.';
exponentIndicator = 'E';
numeratorIndicator = 'N';
denominatorIndicator = 'D';
leadingDigit = '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
lengthOfExponent = leadingDigit, {trailingDigit};
lengthOfIntegerPart = (leadingDigit, {trailingDigit});
lengthOfNumerator = leadingDigit, {trailingDigit};
lengthOfDenominator = leadingDigit, {trailingDigit};
lengthOfFractionalPart = (leadingDigit, {trailingDigit}) | '0' ;
lengthOfIntegralPart = (leadingDigit, {trailingDigit})| '0' ;
lengthOfNumber = leadingDigit, {trailingDigit};
trailingDigit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
signedExponent = 'S';
signedNumber = space,'S';
space = ' ';
variableLengthIndicator = '..';
```

`decimalMark`: separator between integral and fractional part of numbers of format `NR2` or `NR3`.
`leadingDigit`: first cipher of a number comprising one or more ciphers.
`trailingDigit`: one of the ciphers that combines to form numbers, except the first one.

NOTE If a number comprises only one digit, no `trailingDigit` is present.

## D.4 Quantitative value formats

The four quantitative value format syntax rules and their meanings for value representation are defined in the following four subclauses. They are allowed for use for properties having the following data types:

— **number_type** or any of its subtype;

— **level_type** whose **value_type** are either **real_measure_type** or **int_measure_type**;

— **list_type**, **set_type**, **bag_type**, **array_type** or **set_with_subset_constraint_type** whose **value_type** are **number_type** or any of its subtype.

NOTE 1 **list_type**, **set_type**, **bag_type**, **array_type** or **set_with_subset_constraint_type** are defined in ISO 13584-25.

NOTE 2 For **non_quantitative_int_type** the value format applies to the code.

NOTE 3 The value of this attribute should be compatible with the data type of the property: it should not change this data type, else it should be ignored.

EXAMPLE The value format NR2 is not compatible with **int_type**, since integer values shall not have a fractional part.

### D.4.1 NR1-value format

The NR1-value syntax specifies the format of an integer property value.

**Syntax rule:**

```
NR1Value = 'NR1', ((signedNumber, variableLengthIndicator) | (signedNumber, space) | variableLengthIndicator | space), lengthOfNumber;
```

The meaning of NR1-value format components for value representation is as follows:

— `'NR1'`: the value shall be an integer.

NOTE 1 NR1 number values shall not contain any spaces.

— `lengthOfNumber`: number of digits of the value.

NOTE 2 If preceded by a `variableLengthIndicator` the actual number of digits may be less.

— `signedNumber`: if `signedNumber` is present, the related number shall have either a positive, negative, or zero value. In case of positive values a '+' sign may be present. Negative values shall be preceded by a '-' sign. The value zero shall not be preceded by a '-' sign.

— `variableLengthIndicator`: if `variableLengthIndicator` is present, the related number shall contain a number of digits that is less or equal to its length specification, i.e., to `lengthOfNumber`.

### D.4.2 NR2-value format

The NR2-value syntax specifies the format of a real property value that does not need an exponent.

**Syntax rule:**

```
NR2Value = 'NR2', ((signedNumber, variableLengthIndicator) | (signedNumber, space) | variableLengthIndicator | space), lengthOfIntegralPart, decimalMark, lengthOfFractionalPart;
```

The meaning of NR2-value format components for value representation is as follows:

— `'NR2'`: the value shall be a real.

NOTE 1 NR2 number values shall not contain any spaces.

— `lengthOfFractionalPart`: number of digits of the fractional part of the number.

NOTE 2 If preceded by a `variableLengthIndicator` the actual number of digits of the fractional part may be less.

NOTE 3 `lengthOfFractionalPart` implicitly specifies the recommended accuracy of the value. The actual accuracy of the number from which this value was derived may have been greater than the value expressed here.

— `lengthOfIntegralPart`: number of digits of the integral part of the number.

NOTE 4 If preceded by a `variableLengthIndicator` the actual number of digits of the integral part may be less.

— signedNumber: if signedNumber is present, the related number shall have either a positive, negative, or zero value. In case of positive values a '+' sign may be present. Negative values shall be preceded by a '-' sign. The value zero shall not be preceded by a '-' sign.

— variableLengthIndicator: if variableLengthIndicator is present, either integral part or fractional part of the number or both parts shall contain a number of digits that is less or equal to its related length specification, i.e., to lengthOfIntegralPart or lengthOfFractionalPart. At least one cipher shall be present in the number.

### D.4.3 NR3-value format

The NR3-value syntax specifies the format of a real property value that is represented with an exponent.

**Syntax rule:**

```
NR3Value = 'NR3', ((signedNumber, variableLengthIndicator) | (signedNumber, space) | variableLengthIndicator | space), lengthOfIntegralPart, decimalMark, lengthOfFractionalPart, exponentIndicator, [signedExponent], lengthOfExponent;
```

The meaning of NR3-value format components for value representation is as follows:

— 'NR3': the value shall be a real with an exponent of base 10.

NOTE 1 There shall be at least one digit and the decimal mark in the mantissa. The exponent shall contain at least one digit, too.

NOTE 2 NR3 number values shall not contain any spaces.

— exponentIndicator: separator between mantissa and exponent in numbers of format NR3.

— lengthOfExponent: number of digits of the exponent.

NOTE 3 If preceded by a variableLengthIndicator the actual number of digits of the exponent may be less.

NOTE 4 Eventually existing signs or a decimal mark are not counted by lengthOfNumber, lengthOfIntegralPart, lengthOfFractionalPart or lengthOfExponent.

— lengthOfFractionalPart: number of digits of the fractional part of the mantissa.

NOTE 5 If preceded by a variableLengthIndicator the actual number of digits of the fractional part may be less.

NOTE 6 lengthOfFractionalPart implicitly specifies the recommended accuracy of the value. The actual accuracy of the number from which this value was derived may have been greater than the value expressed here.

— lengthOfIntegralPart: number of digits of the integral part of the mantissa.

NOTE 7 If preceded by a variableLengthIndicator the actual number of digits of the integral part may be less.

— signedExponent: if signedExponent is present, the related exponent shall have either a positive, negative, or zero value. In case of positive values a '+' sign may be present. Negative values shall be preceded by a '-' sign. The value zero shall not be preceded by a '-' sign.

— variableLengthIndicator: if variableLengthIndicator is present, the related number or exponent shall contain a number of digits that is less or equal to its related length specification, i.e., to lengthOfIntegralPart, lengthOfFractionalPart, or lengthOfExponent. At least one cipher shall be present in the mantissa and in the exponent.

### D.4.4 NR4-value format

The NR4-value syntax specifies the format of a rational property value that is represented with an integer part, and possibly a fraction part with a denominator and a numerator.

**Syntax rule:**

```
NR4Value = 'NR4', ((signedNumber, variableLengthIndicator) | (signedNumber, space) | variableLengthIndicator | space), lengthOfIntegerPart, numeratorIndicator, lengthOfNumerator, denominatorIndicator, lengthOfDenominator;
```

The meaning of NR4-value format components for value representation is as follows:

— `'NR4'`: the value shall be a rational number represented either as an integer, or as a fraction consisting of a numerator and a denominator, or as an integer and a fraction.

EXAMPLE 12 ½ and 12 ¾ are values that may be represented in the NR4 format.

NOTE 1 There shall be at least one digit either in the integer part, or both in the numerator and in the denominator part. If one part of the fraction contains a digit, the other part shall also contain some digits. All three parts may also contain digits.

NOTE 2 NR4 number values shall not contain any spaces.

— `numeratorIndicator`: separator between the integer part description and the fraction part description in formats NR4.

— `lengthOfNumerator`: number of digits of the numerator.

NOTE 3 If preceded by a `variableLengthIndicator` the actual number of digits of the numerator may be less.

NOTE 4 If the value of the rational number is completely represented by its integer part, neither the numerator of the fraction nor its denominator shall be represented.

— `denominatorIndicator`: separator between the numerator part description and the denominator part description in formats NR4.

— `lengthOfDenominator`: number of digits of the denominator.

NOTE 5 If preceded by a `variableLengthIndicator` the actual number of digits of the denominator may be less.

NOTE 6 If the value of the rational number is completely represented by its integer part, neither the numerator of the fraction nor its denominator shall be represented.

— `lengthOfIntegerPart`: number of digits of the integer part of the rational number.

NOTE 7 If preceded by a `variableLengthIndicator` the actual number of digits of the integer part may be less.

— `variableLengthIndicator`: if `variableLengthIndicator` is present, the three parts of the rational number shall contain a number of digits that is less or equal to its related length specification, i.e., to `lengthOfIntegralPart`, `lengthOfNumerator`, or `lengthOfDenominator`. At least one cipher shall be present either in the integral part, or in the two parts of the fraction.

## D.5 Non-quantitative value formats

The five non-quantitative value format syntax rules and their meanings are defined in the following five sub-clauses. They are allowed for use for properties having the following data types:

— **string_type** or any of its subtype;

— **list_type**, **set_type**, **bag_type**, **array_type** or **set_with_subset_constraint_type** whose **value_type** are **string_type** or any of its subtype.

NOTE 1 **list_type**, **set_type**, **bag_type**, **array_type** or **set_with_subset_constraint_type** are defined in ISO 13584-25.

NOTE 2 For **translatable_string_type** the value format applies to any language-specific representation of the string.

NOTE 3 For **non_quantitative_code_type**, the value format applies to the code.

Non-quantitative values are represented by strings which comprise characters. The length of a string may be either specified by directly specifying the upper limit of the number of contained characters or by specifying that the total number of characters may be any integral multiple of the length specified.

**Syntax rule:**

```
factor = leadingDigit, {trailingDigit};

numberOfCharacters = (leadingDigit, {trailingDigit})|( '(nx', factor,')');
```

The meaning of the factor components is as follows

— `factor`: when `factor` is present, then `numberOfCharacters` shall be any integral multiple of the value given in `factor`. `factor` shall not contain the value zero.

— `numberOfCharacters`: determines the maximum amount of characters contained in the string.

### D.5.1 Alphabetic Value Format

An “Alphabetic Value Format (A)” defines the value format of a string that contains alphabetic letters. Thus, the content shall be taken from the characters of row 00, cell 20, cell 40 to 7E, or cell C0 to FF, of the Basic Multilingual Plane (BMP) (Plane 00 of Group 00) of ISO/IEC 10646-1.

NOTE 1 Due to potential interpretation problems of value content within components of one system or of multiple systems, it is recommended that, where possible, the characters used should be restricted to the G0 set of ISO/IEC 10646-1 and/or row 00 columns 002 to 007 of ISO/IEC 10646-1.

NOTE 2 For alternative languages, as supported by translated data, the relevant characters or ideographs of the related language specific character set are available as defined by the Unicode Standard. In most cases, there will be no 1:1 relation between the characters of the source language to the characters of the target language.

EXAMPLE CJK (Chinese-Japanese-Korean) ideographs.

**Syntax rule:**

```
AValue = 'A', (space | variableLengthIndicator), numberOfCharacters;
```

The meaning of A-value format components for value representation is as follows:

— `'A'`: the value shall be a string, or several substrings, of alphabetic letters.

— `variableLengthIndicator`: if `variableLengthIndicator` is present, the string may contain fewer characters than indicated by `numberOfCharacters`. The string shall contain at least one character.

### D.5.2 Mixed Characters Value Format

A “Mixed Value Format (M)” format defines the value format of a string that may contain any character specified in Clause D.7.

NOTE For alternative languages as supported by translated data, the relevant characters or ideographs of the related language specific character set are available as defined by the Unicode Standard.

EXAMPLE CJK (Chinese-Japanese-Korean) characters.

**Syntax rule:**

```
MValue = 'M', (space | variableLengthIndicator), numberOfCharacters;
```

The meaning of M-value format components for value representation is as follows:

— `'M'`: the value shall be a string, or several substrings.

— `variableLengthIndicator`: if `variableLengthIndicator` is present, the string may contain fewer characters than indicated by `numberOfCharacters`. The string shall contain at least one character.

### D.5.3 Number Value Format

A “Number Value Format (N)”defines the value format of a string that contains numeric digits only. Thus, the content shall be taken from the characters of row 00, cell 2B, cell 2D, cell 30 to 39, or cell 45 of the Basic Multilingual Plane (BMP) (Plane 00 of Group 00) of ISO/IEC 10646-1.

NOTE For alternative languages as supported by translated data, the relevant characters or ideographs of the related language specific character set are available as defined by the Unicode Standard.

EXAMPLE Table D.2 shows the transposition of the European digits “0” to “9” into Arabic digits.

**Table D.2 — Transposing European style digits into Arabic digits**

| European digits | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| Arabic digits | ٩ | ٨ | ٧ | ٦ | ٥ | ٤ | ٣ | ٢ | ١ | ٠ |

**Syntax rule:**

```
NValue = 'N', (space | variableLengthIndicator), numberOfCharacters;
```

The meaning of N-value format components for value representation is as follows:

— `'N'`: the value shall be a string, or several substrings, of numeric digits.

— `variableLengthIndicator`: if `variableLengthIndicator` is present, the string may contain fewer characters than indicated by `numberOfCharacters`. The string shall contain at least one character.

### D.5.4 Mixed Alphabetic or Numeric Characters Value Format

A “Mixed Alphabetic or Numeric Characters Value Format (X)” defines the value format of a string that contains alphanumeric characters, i.e., any combination of characters from A-value format or N-value format.

NOTE For alternative languages as supported by translated data, the relevant characters or ideographs of the related language specific character set are available as defined by the Unicode Standard.

**Syntax rule:**

```
XValue = 'X', (space | variableLengthIndicator), numberOfCharacters;
```

The meaning of X-value format components for value representation is as follows:

— `'X'`: the value shall be a string, or several substrings, of alphanumeric, i.e., any combination of alphabetic and numeric characters;

— `variableLengthIndicator`: if `variableLengthIndicator` is present, the string may contain fewer characters than indicated by `numberOfCharacters`. The string shall contain at least one character.

### D.5.5 Binary Value Format

A “Binary Value Format (B)”defines the value format of a string that contains binary characters, i.e., “0” or “1”. Thus the content shall be taken from the characters of row 00, cell 30 or 31, of the Basic Multilingual Plane (BMP) (Plane 00 of Group 00) of ISO/IEC 10646-1.

NOTE For alternative languages as supported by translated data, the relevant characters or ideographs of the related language specific character set are available as defined by the Unicode Standard.

**Syntax rule:**

```
BValue = 'B', (space | variableLengthIndicator), numberOfCharacters;
```

The meaning of B-value format components for value representation is as follows:

— `'B'`: the value shall be a string, or several substrings, consisting of binary values, i.e., the characters “0” or ”1” or sequences thereof.

— `variableLengthIndicator`: if `variableLengthIndicator` is present, the string may contain fewer characters than indicated by `numberOfCharacters`. The string shall contain at least one character.

## D.6 Value examples

Table D.3 below contains some examples for values that may be contained in numbers and strings characterized by the above value format scheme.

**Table D.3 — Number value examples**

| Value format | Examples of possible values |
|---|---|
| NR1 3 | 123; 001; 000; |
| NR1..3 | 123; 87; 5 |
| NR1S 3 | +123; +000; |
| NR1S..3 | -123; +1; 0; -12; |
| NR2 3.3 | 123.300; 000.400 ; 000.420; |
| NR2..3.3 | 321.233; 1.234; 23.56; 9.783; .72; 324. |
| NR2S 3.3 | -123.123; +123.300; |
| NR2S..3.3 | -123.123; +12.3; 0.1; +.4; -3. ; 0. ; -0; |
| NR3 3.3E4 | 123.123E0004; 003.000E1000; |
| NR3 3.3ES4 | 123.123E+0004; 123.123E0004; 123.000E-0001 |
| NR3..3.3E4 | 123.123E0004; .123E0001; 5.E1234 |
| NR3S 3.3ES4 | +123.123E+0004; 123.000E-0001; |
| NR3S..3.3ES4 | -123.123E+0004; +1.00E-01; .0E0; +3.E-1; -.2E-1000; |
| NR4 3N2D2 | 001 02/03; 012 00/01; 123 03/04; 000 01/04 |
| NR4..3N2D2 | 1 ½ ; 12; 123 ¾; ¼ ; |
| NR4S 3N2D2 | +001 02/03; -012 00/01; 123 03/04; -000 01/04 |
| NR4S..3N2D2 | -1 ½ ; 12; +123 ¾; ¼ ; |
| A 19 | My name is Reinhard, abcdefghijklmnopqrs |
| A..3 | Abc; de; G |
| X..5 | A23RN1; B1; ca |
| M..10 | A23RN1; B1; ca. 256 µm; |
| N (nx5) | 12345; 1234512345; 222223333344444; |
| N..(nx5) | 1234; 12345; 34512345; 1234512345; 23333344444; 222223333344444; -3; 5E2; |
| B 1 | 0; 1; |
| B 3 | 011; 101; |

## D.7 Characters from ISO/IEC 10646-1

The following characters shall be used for the purpose of Mixed Value Format (M) (see D.5.2):

— all characters from row 00 of the Basic Multilingual Plane (BMP) (Plane 00 of Group 00) of ISO/IEC 10646-1;

— characters from other rows of the Basic Multilingual Plane of ISO/IEC 10646-1 as listed in Table D.4;

— for alternative languages as supported by translated data, the full character set is available as defined by the Unicode Standard.

NOTE 1 Due to potential interpretation problems of value content within components of one system or of multiple systems it is recommended that, where possible, the characters used should be restricted to the G0 set of ISO/IEC 10646-1 and/or row 00 columns 002 to 007 of ISO/IEC 10646-1.

NOTE 2 The Unicode Standard is published by The Unicode Consortium, P.O. Box 391476, Mountain View, CA 94039-1476, U.S.A., www.unicode.org.

**Table D.4 — Characters from other rows of the Basic Multilingual Plane of ISO/IEC 10646-1**

| Character | Name | Row | Cell |
|---|---|---|---|
| ˇ | CARON | 02 | C7 |
| ≡ | IDENTICAL TO | 22 | 61 |
| ∧ | LOGICAL AND | 22 | 27 |
| ∨ | LOGICAL OR | 22 | 28 |
| ∩ | INTERSECTION | 22 | 29 |
| ∪ | UNION | 22 | 2A |
| ⊂ | SUBSET OF (IS CONTAINED) | 22 | 82 |
| ⊃ | SUBSET OF (CONTAINS) | 22 | 83 |
| ⇐ | LEFTWARDS DOUBLE ARROW (IS IMPLIED BY) | 21 | D0 |
| ⇒ | RIGHTWARDS DOUBLE ARROW (IMPLIES) | 21 | D2 |
| ∴ | THEREFORE | 22 | 34 |
| ∵ | BECAUSE | 22 | 35 |
| ∈ | ELEMENT OF | 22 | 08 |
| ∋ | CONTAINS AS MEMBER (HAS AS AN ELEMENT) | 22 | 0B |
| ⊆ | SUBSET OR EQUAL TO (CONTAINED AS SUB-CLASS) | 22 | 86 |
| ⊇ | SUPERSET OR EQUAL TO (CONTAINS AS SUB-CLASS) | 22 | 87 |
| ∫ | INTEGRAL | 22 | 2B |
| ∮ | CONTOUR INTEGRAL | 22 | 2E |
| ∞ | INFINITY | 22 | 1E |

**Table D.4 (continued)**

| Character | Name | Row | Cell |
|---|---|---|---|
| ∇ | NABLA | 22 | 07 |
| ∂ | PARTIAL DIFFERENTIAL | 22 | 02 |
| ~ | TILDE OPERATOR (DIFFERENCE BETWEEN) | 22 | 3C |
| ≈ | ALMOST EQUAL TO | 22 | 48 |
| ≃ | ASYMPTOTICALLY EQUAL TO | 22 | 43 |
| ≅ | APPROXIMATELY EQUAL TO (SIMILAR TO) | 22 | 45 |
| ≤ | LESS THAN OR EQUAL TO | 22 | 64 |
| ≠ | NOT EQUAL TO | 22 | 60 |
| ≥ | GREATER THAN OR EQUAL TO | 22 | 65 |
| ⇔ | LEFT RIGHT DOUBLE ARROW (IF AND ONLY IF) | 21 | D4 |
| ¬ | NOT SIGN | 00 | AC |
| ∀ | FOR ALL | 22 | 00 |
| ∃ | THERE EXISTS | 22 | 03 |
| א | HEBREW LETTER ALEF | 05 | D0 |
| □ | WHITE SQUARE (D'ALEMBERTIAN OPERATOR) | 25 | A1 |
| ∥ | PARALLEL TO | 22 | 25 |
| Γ | GREEK CAPITAL LETTER GAMMA | 03 | 93 |
| Δ | GREEK CAPITAL LETTER DELTA | 03 | 94 |
| ⊥ | UPTACK (ORTHOGONAL TO) | 22 | A5 |
| ∠ | ANGLE | 22 | 20 |

**Table D.4 (continued)**

| Character | Name | Row | Cell |
|---|---|---|---|
| ⌙ | RIGHT ANGLE WITH ARC | 22 | BE |
| Θ | GREEK CAPITAL LETTER THETA | 03 | 98 |
| 〈 | LEFT POINTING ANGLE BRACKET (BRA) | 23 | 29 |
| 〉 | RIGHT POINTING ANGLE BRACKET (KET) | 23 | 2A |
| Λ | GREEK CAPITAL LETTER LAMBDA | 03 | 9B |
| ′ | PRIME | 20 | 32 |
| ″ | DOUBLE PRIME | 20 | 33 |
| Ξ | GREEK CAPITAL LETTER XI | 03 | 9E |
| ∓ | MINUS –OR– PLUS SIGN | 22 | 13 |
| Π | GREEK CAPITAL LETTER PI | 03 | A0 |
| ² | SUPERSCRIPT TWO | 00 | B2 |
| Σ | GREEK CAPITAL LETTER SIGMA | 03 | A3 |
| × | MULTIPLICATION SIGN | 00 | D7 |
| ³ | SUPERSCRIPT THREE | 00 | B3 |
| Υ | GREEK CAPITAL LETTER UPSILON | 03 | A5 |
| Φ | GREEK CAPITAL LETTER PHI | 03 | A6 |
| · | MIDDLE DOT | 00 | B7 |
| Ψ | GREEK CAPITAL LETTER PSI | 03 | A8 |
| Ω | GREEK CAPITAL LETTER OMEGA | 03 | A9 |
| ∅ | EMPTY SET | 22 | 05 |

**Table D.4 (continued)**

| Character | Name | Row | Cell |
|---|---|---|---|
| ⃗ | RIGHTWARDS HARPOON WITH BARB UPWARDS (VECTOR OVERBAR) | 21 | C0 |
| √ | SQUARE ROOT (RADICAL) | 22 | 1A |
| ƒ | LATIN SMALL LETTER F WITH HOOK (FUNCTION OF) | 01 | 92 |
| ∝ | PROPORTIONAL TO | 22 | 1D |
| ± | PLUS – MINUS SIGN | 00 | B1 |
| ° | DEGREE SIGN | 00 | B0 |
| α | GREEK SMALL LETTER ALPHA | 03 | B1 |
| β | GREEK SMALL LETTER BETA | 03 | B2 |
| γ | GREEK SMALL LETTER GAMMA | 03 | B3 |
| δ | GREEK SMALL LETTER DELTA | 03 | B4 |
| ε | GREEK SMALL LETTER EPSILON | 03 | B5 |
| ζ | GREEK SMALL LETTER ZETA | 03 | B6 |
| η | GREEK SMALL LETTER ETA | 03 | B7 |
| θ | GREEK SMALL LETTER THETA | 03 | B8 |
| ι | GREEK SMALL LETTER IOTA | 03 | B9 |
| κ | GREEK SMALL LETTER KAPPA | 03 | BA |
| λ | GREEK SMALL LETTER LAMBDA | 03 | BB |
| μ | GREEK SMALL LETTER MU | 03 | BC |
| ν | GREEK SMALL LETTER NU | 03 | BD |
| ξ | GREEK SMALL LETTER XI | 03 | BE |

Table D.4 (continued)

| Character | Name | Row | Cell |
|---|---|---|---|
| ‰ | PER MILLE SIGN | 20 | 30 |
| π | GREEK SMALL LETTER PI | 03 | C0 |
| ρ | GREEK SMALL LETTER RHO | 03 | C1 |
| σ | GREEK SMALL LETTER SIGMA | 03 | C3 |
| ÷ | DIVISION SIGN | 03 | F7 |
| τ | GREEK SMALL LETTER TAU | 03 | C4 |
| υ | GREEK SMALL LETTER UPSILON | 03 | C5 |
| φ | GREEK SMALL LETTER PHI | 03 | C6 |
| χ | GREEK SMALL LETTER CHI | 03 | C7 |
| ψ | GREEK SMALL LETTER PSI | 03 | C8 |
| ω | GREEK SMALL LETTER OMEGA | 03 | C9 |
| † | DAGGER | 20 | 20 |
| ← | LEFTWARDS ARROW | 21 | 90 |
| ↑ | UPWARDS ARROW | 21 | 91 |
| → | RIGHTWARDS ARROW | 21 | 92 |
| ↓ | DOWNWARDS ARROW | 21 | 93 |
| ‾ | OVERLINE | 20 | 3E |

# Annex E
# (normative)

# Information object registration

## E.1 Document identification

To provide for unambiguous identification of an information object in an open system, the object identifier

```
{ iso standard 13584 part (42) version(3) }
```

is assigned to this part of ISO 13584. The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

## E.2 Schema identification

### E.2.1 ISO13584_IEC61360_dictionary_schema

To provide for unambiguous identification of the **ISO13584_IEC61360_dictionary_schema** in an open information system, the object identifier

```
{ iso standard 13584 part (42) version(3) schema(1)
    ISO13584-IEC61360-dictionary-schema(1) }
```

is assigned to the **ISO13584_IEC61360_dictionary_schema** schema (see Clause F.3). The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

### E.2.2 ISO13584_IEC61360_language_resource_schema

The **ISO13584_IEC61360_language_resource_schema** is assigned the object identifier

```
{ iso standard 13584 part (42) version(3) schema (1)
  ISO13584-IEC61360-language-resource-schema(2) }
```

is assigned to the **ISO13584_IEC61360_language_resource_schema** schema (see Clause F.4). The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

### E.2.3 ISO13584_IEC61360_class_constraint_schema

The **ISO13584_IEC61360_class_constraint_schema** is assigned the object identifier

```
{ iso standard 13584 part (42) version(1) schema (1)
  ISO13584-IEC61360-class-constraint-schema(3) }
```

is assigned to the **ISO13584_IEC61360_class_constraint_schema** (see Clause F.5). The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

### E.2.4 ISO13584_IEC61360_item_class_case_of_schema

The **ISO13584_IEC61360_item_class_case_of_schema** is assigned the object identifier

```
{ iso standard 13584 part (42) version(2) schema (1)
  ISO13584-IEC61360-item-class-case-of-schema (4) }
```

is assigned to the **ISO13584_IEC61360_item_class_case_of_schema** schema (see Clause F.6). The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

# Annex F
# (informative)

# Subset of the common IEC/ISO dictionary schema documented in this part of ISO 13584

This annex provides a common subset of the common ISO/IEC dictionary documented both in ISO 13584-25 and IEC 61360-5 . The normative version of this subset is published in IEC 61360-2. This informative annex duplicates the normative content of IEC 61360-2 and provides, by means of notes, some additional explanations referring to this part of ISO 13584.

## F.1 General

### F.1.1 Scope and object of the common ISO13584/IEC61360 dictionary model

The scope the common ISO/IEC dictionary schema based is defined by the intersection of the scopes of the two base standards:

— IEC 61360-1 , generated by IEC TC3 SC3D, and

— this part of ISO 13584, generated by ISO/TC184/SC4/WG2.

The presented EXPRESS model represents a common formal model for the two documents and facilitates a harmonization of both.

Relevant parts of their scope clauses are cited below.

From IEC 61360-1 : "This part of IEC 61360 provides a firm basis for the clear and unambiguous definition of characteristic properties (data element types) of all elements of electrotechnical systems from basic components to sub-assemblies and full systems. Although originally conceived in the context of providing a basis for the exchange of information on electric/electronic components, the principles and methods of this standard may be used in areas outside the original conception such as assemblies of components and electrotechnical systems and subsystems."

From the scope of this part of ISO 13584: "This part of ISO 13584 specifies…

— the attributes to be provided by information suppliers to describe the characterization classes and properties of parts;

— (…)

— the specifications of those attributes in the EXPRESS information model that provides for the exchange of such dictionary data".

The object of this EXPRESS schema is to provide a formal model for data according to the scopes as given above, and thus to provide a means for the computer-sensible representation and exchange of such data.

The intention is to provide a common information model for the work of both committees, thus allowing for the implementation of dictionary systems dealing with data delivered according to either of the standards elaborated by both committees.

### F.1.2 Interoperability of ISO 13584 and IEC 61360

The subset of the common ISO13584/IEC61360 dictionary model documented in this part of ISO 13584 may be used, both in the context of ISO 13584 and in the context of IEC 61360, for exchanging simple dictionary data.

For those dictionaries that require extended capabilities, like external documents associated with dictionary data, properties of which values have an aggregate structure or description of functional models, extensions to the schemata documented in this part of ISO 13584 have been jointly defined by ISO/TC184/SC4/WG2 and by IEC TC3 SC3D. The schemata documented in this part of ISO 13584 and these extensions have been gathered to constitute the new common ISO13584/IEC61360 dictionary model, usable both in the context of ISO 13584 and in the context of IEC 61360. The complete common ISO13584/IEC61360 dictionary model is documented both in ISO 13584-25 and in IEC 61360-5 . An XML-based definition is documented in ISO 13584-32.

The scope of the ISO 13584 standard series is broader than the scope of the IEC 61360 standard series as the latter only addresses product ontologies, when the former addresses both product ontologies and libraries. Indeed, ISO 13584-25 provides for a number of options that may be supported by an implementation. These options have been grouped into conformance classes. Conformance to a particular conformance class requires that all entities, types, and associated constraints defined as part of that class shall be supported. Support for a particular conformance class requires support of all the options specified in this class.

Conformance classes 1 to 4 of ISO 13584-25 correspond exactly to conformance classes 1 to 4 defined in IEC 61360-5 for exchanging dictionaries that are more complex than the ones that may be exchanged using only the subset documented both in this part of ISO 13584 and in IEC 61360-2.

Both committees agreed NOT to change and/or modify the presented EXPRESS models independent of each other in order to guarantee the harmonization and the reusability of the work from both committees. Requests for amendments should therefore be sent to both committees. These requests should be adopted by both committees before the EXPRESS information model is modified.

## F.2 Overview of the subset of the common ISO13584/IEC61360 dictionary model documented in this part of ISO 13584

This section explains the main resource constructs provided by the subset of the common ISO13584/IEC61360 dictionary model documented in this part of ISO 13584.

— **dictionary_element** is any element defined in the dictionary;

— **supplier_element** captures the data of suppliers of dictionary elements (classes, properties, datatypes);

— **class** models the dictionary element of classes (categorization class and characterization class);

— **property_DET** is the dictionary element of a property;

— **data_type** specifies the type of a property.

These parts of the dictionary schema are presented in more detail in the following Clause F.3 "**ISO13584_IEC61360_dictionary_schema**".

In the presentation of the subset of the common ISO13584/IEC61360 dictionary model documented in this part of ISO 13584, some overview diagrams are provided as planning models (Figure F.1 to Figure F.13).These planning models use the EXPRESS-G graphical notation for the EXPRESS language.

For clarification of the diagrams, some of the relationships that are defined in the EXPRESS model are omitted. Figure F.1 below outlines as a planning model the main structure of the common ISO13584/IEC61360 dictionary model.

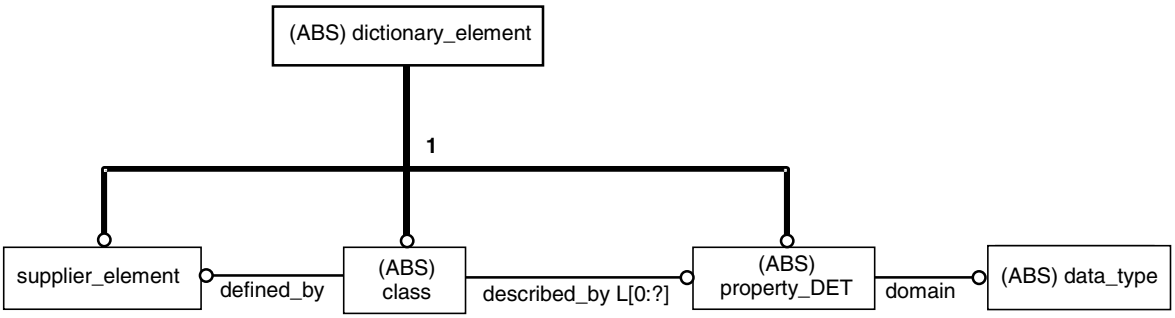



**Figure F.1 — Overview of the dictionary schema**

Most of these figures contain overview models (or planning models) but show only that level of detail that is appropriate at a certain place.

## F.3 ISO13584_IEC61360_dictionary_schema

This clause, that constitutes the main part of the common information model of this part of ISO 13584 and IEC 61360-2, contains the full EXPRESS listing of the subset of the common ISO13584/IEC61360 dictionary model documented in this part of ISO 13584, annotated with comments and explanatory text. The order of text in this clause is determined primarily by the order imposed by the EXPRESS language, secondarily by importance.

### F.3.1 Introduction of the schema of the schema

This subclause introduces the EXPRESS definition of the schema trough its declaration, its external references and the declaration of its constants

#### F.3.1.1 Declaration of the schema

This subclause declares in EXPRESS the **ISO13584_IEC61360_dictionary_schema**.

EXPRESS specification:

```
*)
SCHEMA ISO13584_IEC61360_dictionary_schema;
(*
```

#### F.3.1.2 References to other schemata

This subclause contains references to other EXPRESS schemata that are used in the dictionary schema. Their source is indicated in the respective comment.

EXPRESS specification:

```
*)
REFERENCE FROM support_resource_schema(identifier, label, text);

REFERENCE FROM person_organization_schema(organization, address);

REFERENCE FROM measure_schema;

REFERENCE FROM ISO13584_IEC61360_language_resource_schema;
```

```
REFERENCE FROM ISO13584_IEC61360_class_constraint_schema;

REFERENCE FROM ISO13584_IEC61360_item_class_case_of_schema;

REFERENCE FROM ISO13584_external_file_schema
      (external_item,
      external_file_protocol,
      external_content,
      not_translatable_external_content,
      not_translated_external_content,
      translated_external_content,
      language_specific_content,
      http_file,
      http_class_directory,
      http_protocol);
(*
```

NOTE The schemata referenced above can be found in the following documents:

| | |
|---|---|
| **support_resource_schema** | ISO 10303-41 |
| **person_organization_schema** | ISO 10303-41 |
| **measure_schema** | ISO 10303-41 |
| **ISO13584_IEC61360_language_resource_schema** (which is duplicated for convenience in this annex) | IEC 61360-2 |
| **ISO13584_IEC61360_class_constraint_schema** (which is duplicated for convenience in this annex) | IEC 61360-2 |
| **ISO13584_IEC61360_item_class_case_of_schema** (which is duplicated for convenience in this annex) | IEC 61360-2 |
| **ISO13584_external_file_schema** | ISO 13584-24:2003 |

### F.3.2 Constant definitions

This subclause contains constant definitions used later in type definitions (see F.3.9 "Basic Type and Entity Definitions").

EXPRESS specification:

```
*)
CONSTANT
      dictionary_code_len: INTEGER := 131;
      property_code_len: INTEGER := 35;
      class_code_len: INTEGER := 35;
      data_type_code_len:INTEGER := 35;
      supplier_code_len: INTEGER := 149;
      version_len: INTEGER := 10;
      revision_len: INTEGER := 3;
      value_code_len: INTEGER := 35;
      pref_name_len: INTEGER := 255;
      short_name_len: INTEGER := 30;
      syn_name_len: INTEGER := pref_name_len;
      DET_classification_len: INTEGER := 3;
      source_doc_len: INTEGER := 255;
      value_format_len: INTEGER := 80;
      sep_cv: STRING := '#';
      sep_id: STRING := '#';
END_CONSTANT;
(*
```

### F.3.3 Identification of a dictionary

A **dictionary_identification** entity allows to identify unambiguously a particular version of a particular dictionary of a particular information supplier, standard or not. It contains a **code** defined by the dictionary supplier that identifies the dictionary, a **version** number and **revision** number that characterize a particular state of this dictionary.

NOTE 1 The case where dictionary version and revision should be incremented is defined in Clause 9.

EXPRESS specification:

```
*)
ENTITY dictionary_identification;
      code: dictionary_code_type;
      version: version_type;
      revision: revision_type;
      defined_by: supplier_bsu;
DERIVE
      absolute_id: identifier :=
           defined_by.absolute_id + sep_id + code + sep_cv + version;
UNIQUE
      UR1: absolute_id;
END_ENTITY; -- dictionary_identification
(*
```

Attribute definitions:

**code**: the code that characterizes the dictionary.

**version**: the version number that characterizes the version of the dictionary.

**revision**: the revision number that characterizes the revision of the dictionary.

**defined_by**: the supplier who defines the dictionary.

**absolute_id**: the unique identification of the dictionary.

Formal propositions:

**UR1**: the dictionary identifier defined by the **absolute_id** attribute is unique.

Informal propositions:

**IP1**: when a dictionary is defined by a standard document that contains only one dictionary, the dictionary **code** shall be the standard number of the document that describes this dictionary if this document only defines one dictionary. It shall be the name defined for the pertinent dictionary in the document that describes it if this document defines several dictionaries. Unless otherwise specified, version shall be set to 1 and revision numbers shall be set to 0 for dictionaries defined by standard documents.

NOTE 2 Representation of the standard numbers of standard documents is specified in clauses 5.1 and 5.2 of ISO 13584-26:2000 .

### F.3.4 Basic Semantic Units: defining and using the dictionary

#### F.3.4.1 Requirements for exchange

In the exchange of dictionary and part library data, it is customary to partition the data. For example, a dictionary could be updated with some classes that specify their superclass by a reference to a pre-existing class, or when the content of a library is exchanged, dictionary elements are only referenced and not included every time. It must be possible to refer unambiguously and consistently to the dictionary data.

Thus, it is a clear requirement first, to be able to exchange pieces of data, and second, to have relationships between these pieces. This is depicted in Figure F.2.

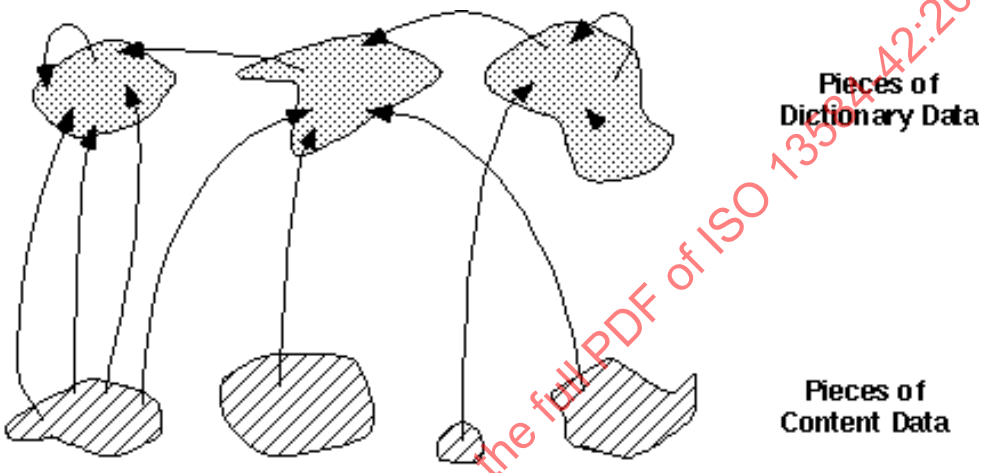



**Figure F.2 — Pieces of data with relationships**

Every one of these pieces corresponds to a physical file (according to ISO 10303-21). EXPRESS (ISO 10303-11:1994) attributes can only contain references to data within the same physical file. Thus it is impossible to use EXPRESS attributes directly to implement inter-piece references.

#### F.3.4.2 Three levels architecture of the dictionary data

In this clause the concept of **basic_semantic_unit** (BSU) is introduced as a means to implement these inter-piece references. A BSU provides a universally unique identification for dictionary descriptions. This is depicted in Figure F.3.

Assume some piece of content wants to refer a certain dictionary description.

EXAMPLE 1 To convey the value of a property of a component.

It does this by referring to a basic semantic unit through the attribute **dictionary_definition**.

A dictionary description (**dictionary_element**) refers to a basic semantic unit through the attribute **identified_by**. From the correspondence of the absolute identifiers of the basic semantic units this indirect relation is established.

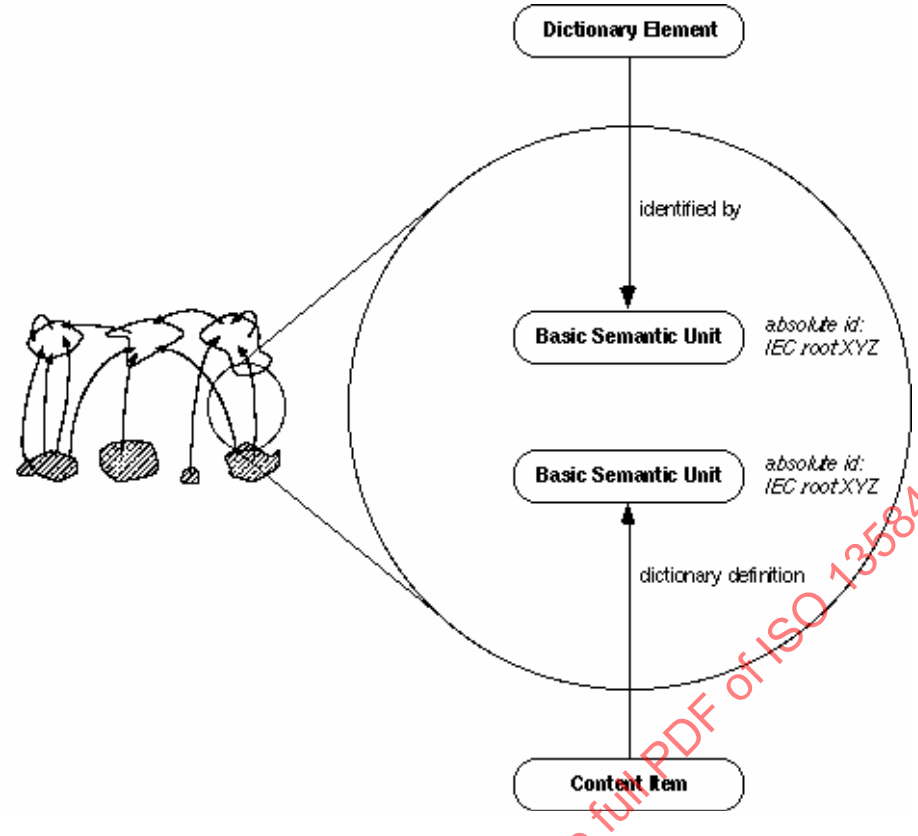



**Figure F.3 — Implementation of "inter-piece" relationships using basic semantic units**

Note that:

— both dictionary element and content item can be present in the same physical file, but need not be;

— the dictionary element does not need to be present for the exchange of some content item referring to it. In this case it is assumed to be present in the dictionary of the target system already. Conversely, dictionary data can be exchanged without any content data;

— the basic semantic unit can be one single instance in the case where both dictionary element and content item instances are in the same physical file;

— the same mechanism applies also to references between various dictionary elements

EXAMPLE 2 Between a class of components and the associated **property_DET**s.

A BSU provides a reference to a dictionary description in any place where this is needed.

EXAMPLE 3 Dictionary delivery, update delivery, library delivery, component data exchange.

The data associated with a property could be exchanged as a pair (**property_BSU**, <value>).

Figure F.3 outlines the implementation of this general mechanism.

#### F.3.4.2.1 Basic_semantic_unit

A **basic_semantic_unit** is a unique identification of a **dictionary_element**. BSU is the abbreviation of basic semantic unit.

EXPRESS specification:

```
*)
ENTITY basic_semantic_unit
ABSTRACT SUPERTYPE OF(ONEOF(
      supplier_BSU,
      class_BSU,
      property_BSU,
      data_type_BSU,
      supplier_related_BSU,
      class_related_BSU));
      code: code_type;
      version: version_type;
DERIVE
      dic_identifier: identifier := code + sep_cv + version;
INVERSE
      definition: SET [0:1] OF dictionary_element
           FOR identified_by;
      referenced_by: SET [0:1] OF content_item
           FOR dictionary_definition;
END_ENTITY; -- basic_semantic_unit
(*
```

Attribute definitions:

**code**: the code assigned to identify a certain dictionary element.

**version**: the version number of a certain dictionary element.

**dic_identifier**: the full identification, consisting of concatenation of code and version.

**definition**: a reference to the dictionary element identified by this BSU. If not present in some exchange context, it is assumed to be present in the dictionary of the target system already.

**referenced_by**: items making use of the dictionary element associated with this BSU.

#### F.3.4.2.2 Dictionary_element

A **dictionary_element** is a full definition of the data required to be captured in the semantic dictionary for some concepts. For every concept, a separate subtype is to be used. The **dictionary_element** is associated with a **basic_semantic_unit** (BSU) that serves to uniquely identify this definition in the dictionary.

By including the version attribute in the **basic_semantic_unit** entity, it forms part of the identification of a dictionary element (in contrast to the **revision** and **time_stamps** attributes).

EXPRESS specification:

```
*)
ENTITY dictionary_element
ABSTRACT SUPERTYPE OF(ONEOF(
      supplier_element,
      class_and_property_elements,
      data_type_element));
      identified_by: basic_semantic_unit;
      time_stamps: OPTIONAL dates;
```

```
      revision: revision_type;
      administration: OPTIONAL administrative_data;
      is_deprecated: OPTIONAL BOOLEAN;
      is_deprecated_interpretation: OPTIONAL note_type;
WHERE
      WR1: NOT EXISTS (SELF.is_deprecated)
                OR EXISTS (SELF.is_deprecated_interpretation);
END_ENTITY; -- dictionary_element
(*
```

Attribute definitions:

**identified_by**: the BSU identifying this dictionary element.

**time_stamps**: the optional dates of creation and update of this dictionary element.

**revision**: the revision number of this dictionary element.

NOTE 1 The type of the **identified_by** attribute will be redefined later to **property_BSU** and **class_BSU** and will then be used to encode together with the code attribute of the BSUs the "Code" attribute for properties (see 7.2) and classes (see 8.2) respectively. It will also be used to encode the "Version Number" attribute for properties and classes respectively.

NOTE 2 The **time_stamps** attribute will be used as a starting point to encode in the **dates** entity the property and class attributes "Date of Original Definition", "Date of Current Version" and "Date of Current Revision" (see 7.2, 8.2 and F.3.9.2).

NOTE 3 The **revision** attribute will be used to encode the property and class attribute "Revision Number" (see 7.2 and 8.2).

**administration**: optional information on the life cycle of the **dictionary_element.**

NOTE 4 The **administration** attribute will be used to represent the information related to the configuration management and translation history.

**is_deprecated**: an optional Boolean. When true, it specifies that the **dictionary_element** shall no longer be used.

**is_deprecated_interpretation**: specifies the deprecation rationale and how instance values of the deprecated element, and of its corresponding **BSU**, should be interpreted.

Formal propositions:

**WR1**: when **is_deprecated** exists, **is_deprecated_interpretation** shall exist.

Informal propositions:

**IP1**: instance values of **is_deprecated_interpretation** element shall be defined at the time where deprecation decision was taken.

Figure F.4 presents a planning model of the relationship between basic semantic unit and the dictionary element.

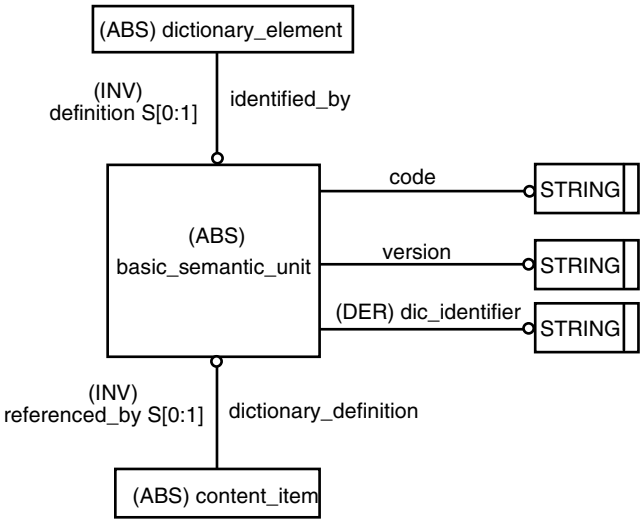



**Figure F.4 — Relationship between basic semantic unit and dictionary element**

#### F.3.4.2.3 Content_item

A **content_item** is a piece of data referring to its description in the dictionary. It shall be subtyped.

EXPRESS specification:

```
*)
ENTITY content_item
ABSTRACT SUPERTYPE;
     dictionary_definition: basic_semantic_unit;
END_ENTITY; -- content_item
(*
```

Attribute definitions:

**dictionary_definition**: the basic semantic unit to be used for referring to the definition in the dictionary.

### F.3.4.3 Overview of basic semantic units and dictionary elements

For every kind of dictionary data, a pair of **basic_semantic_unit** and **dictionary_element** subtypes must be defined. Figure F.5 outlines, as a planning model, the basic semantic units (BSU) and dictionary elements defined later. Note that the relationship between BSU and dictionary elements is redefined for each type of data, so that only corresponding pairs can be related. This is not graphically depicted here, however.

Every kind of dictionary data is treated in one of the following subclauses:

— for suppliers see F.3.5 "Supplier data";

— for classes see F.3.6 "Class data";

— for properties / data element types see F.3.7 "Data element type / properties data";

— for data types see F.3.8 "Domain data: the type system".

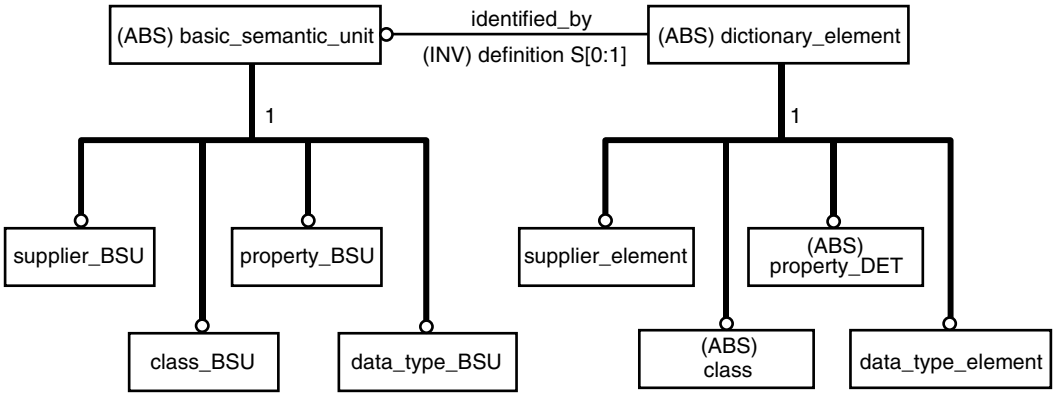



**Figure F.5 — Current BSUs and dictionary elements**

### F.3.4.4 Identification of dictionary elements: three levels structure

The absolute identification of basic semantic units is based on the following three levels structure:

— supplier (of dictionary data);

— supplier-defined dictionary element (any supplier-defined dictionary element defined in the model; in this document supplier-defined dictionary element are **property_DET** and **data_type_element,** but there are provisions to extend this mechanism to other items).

— version of the supplier-defined dictionary element.

An absolute identification can be achieved by concatenation of the applicable code for each level.

NOTE The structure on this absolute identification is different from the structure defined in edition 1 of IEC 61360-2 (duplicated for convenience in ISO 13584-42:1998). In the previous edition, the absolute identification of any **dictionary_element** associated with a **name_scope** (including **property_DET** and **data_type_element**) consisted of: supplier code + class code (corresponding to the **name_scope** class) + dictionary element code + dictionary element version. In this edition, the class code has been removed. Thus, the dictionary element code must be unique, for the same type of dictionary element, over all the classes defined by the same supplier. For existing reference dictionaries, registration authorities, maintenance authorities or standardization groups in charge of standard dictionaries should ensure this unicity, possibly by defining new codes prefixed by **name_scope** class codes.

This identification scheme is appropriate within a multi-supplier context. If in a certain application area, only data of one single (data-) supplier are relevant, the corresponding parts of the identification, that are then constant, can be eliminated. For the purpose of exchange, however, all the levels must be present, to avoid clashes of identifiers.

This identification scheme is described formally in the **absolute_id** attribute of the xxx_BSU entities defined in clauses F.3.5 through F.3.8.

### F.3.4.5 Extension possibilities for other types of data

The BSU - dictionary element mechanism is very general and not limited to the four kinds of data used here (see Figure F.5). This clause specifies some facilities that allow for extensions for other kinds. Depending on whether the scope of the identifier is given by a class or a supplier, the corresponding **xxx_related_BSU** entity has to be subtyped. It is necessary to redefine the **identified_by** attribute of the entity **dictionary_element** (as is done in the F.3.5 through F.3.8 for the current kinds of data).

#### F.3.4.5.1 Supplier_related_BSU

The **supplier_related_BSU** provides for the dictionary elements to be associated with suppliers.

EXAMPLE For ISO 13584: program libraries.

EXPRESS specification:

```
*)
ENTITY supplier_related_BSU
ABSTRACT SUPERTYPE
SUBTYPE OF(basic_semantic_unit);
END_ENTITY; -- supplier_related_BSU
(*
```

#### F.3.4.5.2 Class_related_BSU

The **class_related_BSU** provides for the dictionary elements to be associated with classes.

EXAMPLE For ISO 13584 tables, documents, etc .

EXPRESS specification:

```
*)
ENTITY class_related_BSU
ABSTRACT SUPERTYPE
SUBTYPE OF(basic_semantic_unit);
END_ENTITY; -- class_related_BSU
(*
```

#### F.3.4.5.3 Supplier_BSU_relationship

The **supplier_BSU_relationship** is a provision for association of BSUs with suppliers.

EXPRESS specification:

```
*)
ENTITY supplier_BSU_relationship
ABSTRACT SUPERTYPE;
     relating_supplier: supplier_element;
     related_tokens: SET [1:?] OF supplier_related_BSU;
END_ENTITY; -- supplier_BSU_relationship
(*
```

Attribute definitions:

**relating_supplier**: the **supplier_element** that identifies the data supplier.

**related_tokens**: the set of dictionary elements associated to the supplier identified by the **relating_supplier** attribute.

#### F.3.4.5.4 Class_BSU_relationship

The **class_BSU_relationship** entity is a provision for association of BSUs with classes.

EXPRESS specification:

```
*)
ENTITY class_BSU_relationship
```

```
ABSTRACT SUPERTYPE;
      relating_class: class;
      related_tokens: SET [1:?] OF class_related_BSU;
END_ENTITY; -- class_BSU_relationship
(*
```

Attribute definitions:

**relating_class**: the **class** that identifies the dictionary element.

**related_tokens**: the set of dictionary elements associated to the class identified by the **relating_class** attribute.

### F.3.5 Supplier Data

This clause contains definitions for the representation of data about a supplier itself. In a multi-supplier environment it is necessary to be able to identify the source of a certain dictionary element. Figure F.6 presents a planning model of the data associated with suppliers, followed by the EXPRESS definition.

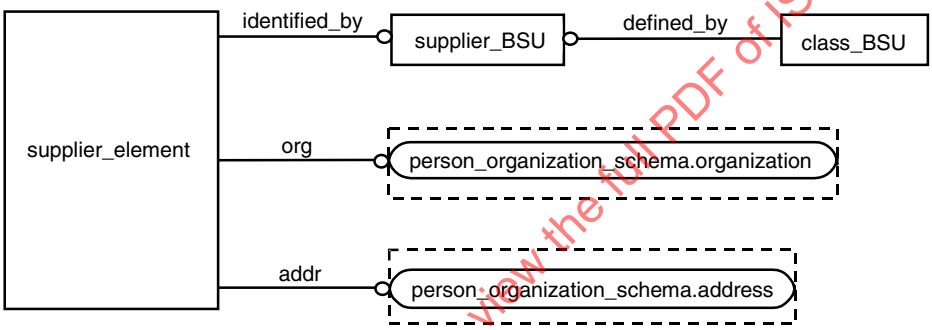



**Figure F.6 — Overview of supplier data and relationships**

#### F.3.5.1 Supplier_BSU

The **supplier_BSU** entity provides for unique identification of information suppliers.

EXPRESS specification:

```
*)
ENTITY supplier_BSU
SUBTYPE OF(basic_semantic_unit);
      SELF\basic_semantic_unit.code: supplier_code_type;
DERIVE
      SELF\basic_semantic_unit.version: version_type := '1';
      absolute_id: identifier := SELF\basic_semantic_unit.code;
UNIQUE
      UR1: absolute_id;
END_ENTITY; -- supplier_BSU
(*
```

Attribute definitions:

**code**: the supplier's code assigned according to ISO 13584-26 .

**version**: the version number of a supplier code shall be equal to 1.

**absolute_id**: the absolute identification of the supplier.

Formal propositions:

**UR1**: the supplier identifier defined by the **absolute_id** attribute is unique.

#### F.3.5.2 Supplier_element

The **supplier_element** entity gives the dictionary description of suppliers.

EXPRESS specification:

```
*)
ENTITY supplier_element
SUBTYPE OF(dictionary_element);
      SELF\dictionary_element.identified_by: supplier_BSU;
      org: organization;
      addr: address;
INVERSE
      associated_items: SET [0:?] OF supplier_BSU_relationship
           FOR relating_supplier;
END_ENTITY; -- supplier_element
(*
```

Attribute definitions:

**identified_by**: the **supplier_BSU** used to identify this **supplier_element**.

**org**: the organizational data of this supplier.

**addr**: the address of this supplier.

**associated_items**: allows access to other kinds of data via the BSU mechanism.

EXAMPLE Program library in ISO 13584-24:2003.

### F.3.6 Class Data

This clause contains definitions for the representation of dictionary data of classes.

#### F.3.6.1 General

Figure F.7 outlines, as a planning model, the data associated with classes and their relationship to other dictionary elements.

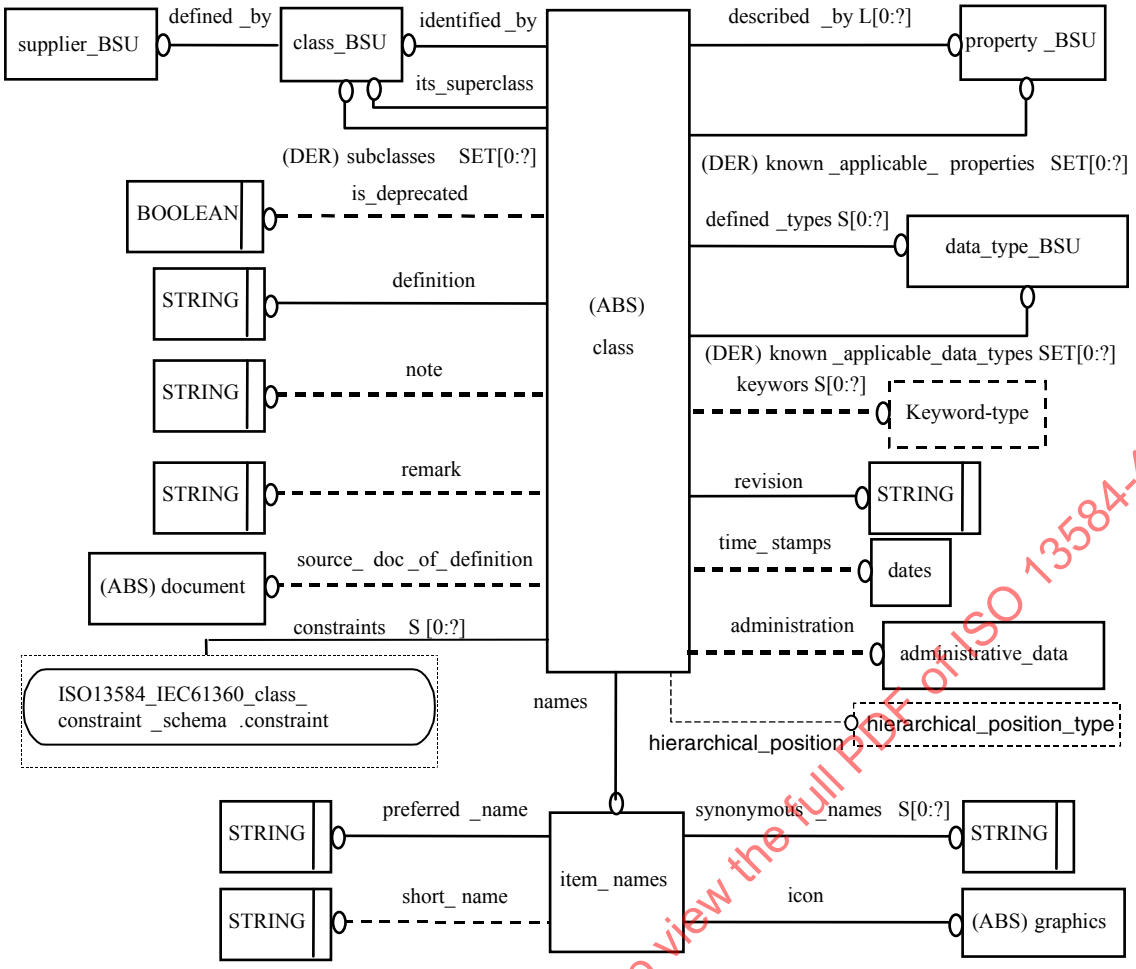



**Figure F.7 — Overview of class data and relationships**

As indicated in the Figure F.7 with the **its_superclass** attribute, classes form an inheritance tree. It is important to note that throughout this document, the terms "inheritance" and "to inherit" stand for this relationship between classes (defined in the dictionary), although EXPRESS has an inheritance concept, too. These must be clearly distinguished to avoid misunderstandings.

The dictionary data for classes (as shown in Figure F.7) is spread over three inheritance levels:

— **class_and_property_elements** defines data common to both **class**es and **property_DET**s;

— **class** allows for other kinds of classes to be specified later;

EXAMPLE Other subtypes of **class**es, in particular **functional_view_class**, **functional_model_class** and **fm_class_view_of** are specified in ISO13584-24:2003. They do not characterize products, but they provide for exchanging particular representations of product, e. g., geometrical representations.

— **item_class** and **categorization_class** are the entities that hold data of different classes of application domain objects.

NOTE 1 Two subtypes of **item_class**, named **component_class** and **material_class**, were defined within the dictionary model of the first edition of this part of ISO 13584 and IEC 61360-2. These subtypes are deprecated, and they are removed from this edition of this part of ISO 13584.

NOTE 2 The following changes ensure that the class definitions of a dictionary conforming with the first edition of this part of ISO 13584 conforms to this edition: (1) replace **component_class** and **material_class** by **item_class** throughout the reference dictionary; (2) add to each new **item_class** class the **instance_sharable** attribute, the value of which being

true; (3) add for each new **item_class** class the optional **hierarchical_position** attribute without setting any value; (4) add for each new **item_class** class the **keywords** attribute, the value of which being an empty collection.

NOTE 3 Another subtype of **item_class**, named **feature_class,** was provided in ISO 13584-24:2003. This subtype is also deprecated and its usage is not allowed in new implementations of this part of ISO 13584 and IEC 61360-2.

NOTE 4 The following changes ensure that the class definitions of a dictionary conforming with ISO 13584-25 conforms to this edition of ISO 13584-42: (1) replace **feature_class** by **item_class** throughout the reference dictionary; (2) add to each new **item_class** class the **instance_sharable** attribute, the value of which being false; (3) add for each new **item_class** class the optional **hierarchical_position** attribute without setting any value; (4) add for each new **item_class** class the **keywords** attribute, the value of which being an empty collection.

#### F.3.6.1.1 Class_BSU

The **class_BSU** entity provides for the identification of classes.

EXPRESS specification:

```
*)
ENTITY class_BSU
SUBTYPE OF(basic_semantic_unit);
      SELF\basic_semantic_unit.code: class_code_type;
      defined_by: supplier_BSU;
DERIVE
      absolute_id: identifier
           := defined_by.absolute_id + sep_id + dic_identifier;
      known_visible_properties: SET [0:?]OF property_BSU
           := compute_known_visible_properties(SELF);
      known_visible_data_types: SET [0:?]OF data_type_BSU
           := compute_known_visible_data_types(SELF);
INVERSE
      subclasses: SET [0:?] OF class FOR its_superclass;
      added_visible_properties: SET [0:?] OF property_BSU
           FOR name_scope;
      added_visible_data_types: SET [0:?] OF data_type_BSU
           FOR name_scope;
UNIQUE
      UR1: absolute_id;
END_ENTITY; -- class_BSU
(*
```

Attribute definitions:

**code**: the code assigned to this class by its supplier.

**defined_by**: the supplier defining this class and its dictionary element.

**absolute_id**: the unique identification of this class.

**known_visible_properties**: the set of **property_BSU**s that refer to the class as their **name_scope** attribute or to any known super-class of this class and that are therefore visible for the class (and any of its subclass).

NOTE 1 When some class **dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the properties defined as visible by this super-class do not belong to the **known_visible_properties** attribute. Only on the receiving system all the **dictionary_definition**s of the BSUs are required to be available. Therefore, on the receiving system, the **known_visible_properties** attribute contains all properties visible for the class.

**known_visible_data_types**: the set of **data_type_BSU**s that refer to the class as their **name_scope** attribute or to any known super-class of this class and that are therefore visible for the class (and any of its subclass).

NOTE 2 When some class **dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the **data_type**s defined as visible by this super-class do not belong to the **known_visible_data_types** attribute. Only on the receiving system all the **dictionary_definition**s of the BSUs are required to be available. Therefore, on the receiving system, the **known_visible_data_types** attribute contains all **data_type**s visible for the class.

**subclasses**: the set of classes specifying this class as their superclass.

**added_visible_properties**: the set of **property_BSU**s that refer to the class as their **name_scope** and that are therefore visible for the class (and any of its subclass).

NOTE 3 Only the **property_BSU**s that belongs to the same exchange context are referenced by this inverse attribute. On the receiving system they may already exit other **property_BSU**s that refer to this class (a PLIB exchange context is never assumed to be complete).

NOTE 4 The **added_visible_properties** attribute will be used to encode the class attribute "Visible Properties" (see 8.2.9).

**added_visible_data_types**: the set of **data_type_BSU**s that refer to the class as their **name_scope** and that are therefore visible for the class (and any of its subclass).

NOTE 5 Only the **data_type_BSU**s that belongs to the same exchange context are referenced by this inverse attribute. On the receiving system they may already exits other **data_type_BSU**s that refer to this class (a PLIB exchange context is never assumed to be complete).

NOTE 6 The **added_visible_data_types** attribute will be used to encode the class attribute "Visible Types" (see 8.2.6).

Formal propositions:

**UR1**: the concatenation of supplier code and class code is unique.

#### F.3.6.1.2 Class_and_property_elements

The **class_and_property_elements** entity captures the attributes that are common to both **class**es and **property_DET**s.

EXPRESS specification:

```
*)
ENTITY class_and_property_elements
ABSTRACT SUPERTYPE OF(ONEOF(
      property_DET,
      class))
SUBTYPE OF(dictionary_element);
      names: item_names;
      definition: definition_type;
      source_doc_of_definition: OPTIONAL document;
      note: OPTIONAL note_type;
      remark: OPTIONAL remark_type;
END_ENTITY; -- class_and_property_elements
(*
```

Attribute definitions:

**names**: the names describing this dictionary element.

**definition**: the text describing this dictionary element.

**source_doc_of_definition**: the source document of this textual description.

**note**: further information on any part of the dictionary element, which is essential to the understanding.

**remark**: explanatory text further clarifying the meaning of this dictionary element.

NOTE 1 The **names** attribute will be used as a starting point to encode in the **item_names** entity the property and class attributes "Preferred Name", "Short Name" and "Synonymous Name" (see 7.2 and 8.2).

NOTE 2 The **definition** attribute will be used to encode the property attribute "Definition" (see 7.2.10) and the class attribute "Definition" (see 8.2.12).

NOTE 3 The **source_of_doc_definition** attribute will be used to encode the property attribute "Source Document of Definition" (see 7.2.11) and the class attribute "Source Document of Definition" (see 8.2.13).

NOTE 4 The **note** attribute will be used to encode the property and class attribute "Note" (see 7.2.12 and 8.2.14).

NOTE 5 The **remark** attribute will be used to encode the property and class attribute "Remark;" (see 7.2.13 and 8.2.15).

#### F.3.6.1.3 Class

The **class** entity is an abstract resource for all kinds of classes.

EXPRESS specification:

```
*)
ENTITY class
ABSTRACT SUPERTYPE OF( ONEOF (item_class, categorization_class))
SUBTYPE OF(class_and_property_elements);
      SELF\dictionary_element.identified_by: class_BSU;
      its_superclass: OPTIONAL class_BSU;
      described_by: LIST [0:?] OF UNIQUE property_BSU;
      defined_types: SET [0:?] OF data_type_BSU;
      constraints: SET [0:?] OF constraint_or_constraint_id;
      hierarchical_position: OPTIONAL hierarchical_position_type;
      keywords: SET [0:?] OF keyword_type;
      sub_class_properties: SET [0:?] OF property_BSU;
      class_constant_values: SET [0:?] OF class_value_assignment;
DERIVE
      subclasses: SET [0:?] OF class := identified_by.subclasses;
      known_applicable_properties: SET [0:?] OF property_BSU
           := compute_known_applicable_properties(
                SELF\dictionary_element.identified_by);
      known_applicable_data_types: SET [0:?] OF data_type_BSU
           := compute_known_applicable_data_types(
                SELF\dictionary_element.identified_by);
      known_property_constraints: SET [0:?] OF property_constraint
           := compute_known_property_constraints(
                [SELF\dictionary_element.identified_by]);
```

```
INVERSE
      associated_items: SET [0:?] of class_BSU_relationship
           FOR relating_class;
WHERE
      WR1: acyclic_superclass_relationship(SELF.identified_by, []);
      WR2: NOT all_class_descriptions_reachable(
           SELF\dictionary_element.identified_by)
           OR (list_to_set(SELF.described_by) <=
           SELF\dictionary_element.identified_by
           \class_BSU.known_visible_properties);
      WR3: NOT all_class_descriptions_reachable(
           SELF\dictionary_element.identified_by)
           OR (SELF.defined_types <=
           SELF\dictionary_element.identified_by
           \class_BSU.known_visible_data_types);
      WR5: NOT all_class_descriptions_reachable(
           SELF\dictionary_element.identified_by)
           OR (QUERY (cdp <* described_by
           | (SIZEOF (cdp\basic_semantic_unit.definition)=1)
           AND (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
           +'.DEPENDENT_P_DET') IN TYPEOF
           (cdp\basic_semantic_unit.definition[1]))
           AND NOT
           (cdp\basic_semantic_unit.definition[1].depends_on
           <= known_applicable_properties))=[]);
      WR6: check_datatypes_applicability(SELF);
      WR7: QUERY (cons <* constraints
           | ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
           +'.INTEGRITY_CONSTRAINT' IN TYPEOF (cons))
           AND (SIZEOF (cons\property_constraint.constrained_property
           .definition) =1)
           AND NOT correct_constraint_type(
           cons\integrity_constraint.redefined_domain,
           cons\property_constraint.constrained_property
           .definition[1].domain)) = [];
      WR8: QUERY (cons <* constraints
           | (('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
           +'.CONFIGURATION_CONTROL_CONSTRAINT') IN TYPEOF (cons))
           AND NOT correct_precondition (cons, SELF)) = [];
      WR9: NOT all_class_descriptions_reachable(
           SELF\dictionary_element.identified_by)
           OR (QUERY (cons <* constraints
           | (('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
           +'.PROPERTY_CONSTRAINT') IN TYPEOF (cons))
           AND NOT
           ((cons\property_constraint.constrained_property
           IN SELF\dictionary_element.identified_by
           \class_BSU.known_visible_properties)
           OR (cons\property_constraint.constrained_property
           IN known_applicable_properties)))=[]);
      WR10: (SIZEOF( QUERY (lab <* keywords
           | ('ISO13584_IEC61360_DICTIONARY_SCHEMA'
           +'.LABEL_WITH_LANGUAGE') IN TYPEOF (lab)))
```

```
            = SIZEOF( keywords))
            OR (SIZEOF (QUERY (lab <* keywords
            | ('ISO13584_IEC61360_DICTIONARY_SCHEMA'
            +'.LABEL_WITH_LANGUAGE') IN TYPEOF (lab)))
            = SIZEOF( keywords));
      WR11: (('ISO13584_IEC61360_ITEM_CLASS_CASE_OF_SCHEMA'
            + '.A_PRIORI_SEMANTIC_RELATIONSHIP')
            IN TYPEOF (SELF)) OR
            ( QUERY(p <* sub_class_properties
            | NOT(p IN SELF.described_by)) = []);
      WR12: NOT all_class_descriptions_reachable(SELF.identified_by) OR
            (QUERY(va <* class_constant_values |
            NOT is_class_valued_property(
            va.super_class_defined_property, SELF.identified_by)) = []);
      WR13: QUERY(val <* SELF.class_constant_values
            | QUERY (v <* class_value_assigned (
            val.super_class_defined_property, SELF.identified_by)
            | val.assigned_value <> v) <>[]) = [];
END_ENTITY; -- class
(*
```

Attribute definitions:

**identified_by**: the **class_BSU** identifying this class.

**its_superclass**: reference to the class the current one is a subclass of.

**described_by**: the list of references to the additional properties available for use in the description of the products within the class, and any of its subclasses.

NOTE 1 A property may also be applicable to a class when this property is imported from another class through an **a_priori_semantic_relationship** as defined in Clause F.6 of this part of ISO 13584. Therefore the properties referenced by the **described_by** attribute do not define all the applicable properties for a class.

NOTE 2 The list order is the presentation order of the properties suggested by the supplier.

NOTE 3 A property that is a context dependent property (**context_dependent_P_DET**) may become applicable to a class only if all the context parameters (**condition_DET**) on which its value depends are also applicable to this class. This is stated in where rule 5.

**defined_types**: the set of references to the types that can be used for various **property_DET**s throughout the inheritance tree descending from this class.

NOTE 4 A **data_type** may also be applicable to a class when this **data_type** is imported from another class through an **a_priori_semantic_relationship** as defined in Clause F.6 of this part of ISO 13584. Therefore the data types referenced by the **defined_types** attribute do not define all the applicable data types for a class.

**constraints**: the set of constraints that restrict the target domains of values of some properties of the class to some subsets of their inherited domains of values.

NOTE 5 Each constraint in the **constraints** attribute must be fulfilled by class instances. Thus the **constraints** attribute is a conjunction of constraints.

**hierarchical_position**: the coded representation of the class position in a class inclusion hierarchy to which it belongs; a **hierarchical_position** of a class changes when the class structure of an ontology is changed. Thus it cannot be used as a stable identifier for classes.

NOTE 6 This kind of coded name is used in particular in product categorization hierarchies for representing the class inclusion structure through some coding conventions.

EXAMPLE 1 In UNSPSC, *Manufacturing Components and Supplies* has the hierarchical position 31000000, *Hardware* has the hierarchical position 31160000 and *Bolt* the hierarchical position 31161600. By convention, this representation of the hierarchical position allows to represent that *Manufacturing Components and Supplies* is at the first level of the hierarchy, that *Hardware* is at the second level of the hierarchy and is included in *Manufacturing Components and Supplies* and that *Bolt* is at the third level of the hierarchy and is included in *Hardware.*

**keywords**: a set of keywords, possibly in several languages, allowing to search the class.

**sub_class_properties**: declares properties as class-valued, i.e. in subclasses one single value will be assigned per class. See F.3.7.4 "Class-valued properties".

**class_constant_values**: assignments in the current class for class-valued properties declared in superclasses. See F.3.7.4 "Class-valued properties".

**subclasses**: the set of classes specifying this class as their superclass.

**known_applicable_properties**: the **property_BSU**s that are referenced by the class or any of its known super-class(es) by their **described_by** attribute and that are therefore applicable to this class (and to any of its subclasses).

NOTE 7 When some class **dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the properties defined as applicable by this super-class do not belong to the **known_applicable_properties** attribute. Only on the receiving system all the **dictionary_definition**s of the **BSU**s are required to be available. Therefore, on the receiving system, the **known_applicable_properties** attribute contains all the properties that are applicable to a class by virtue of being referenced by a **described_by** attribute.

**known_applicable_data_types**: the **data_type_BSU**s that are referenced by the class or any of its known super-class(es) by their **defined_types** attribute and that are therefore applicable to this class (and to any of its subclasses).

NOTE 8 When some class **dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the **data_type**s defined as applicable by this super-class do not belong to the **known_applicable_data_types** attribute. Only on the receiving system all the **dictionary_definition**s of the **BSU**s are required to be available. Therefore, on the receiving system, the **known_applicable_data_types** attribute contains all the **data_type**s that are applicable to a class by virtue of being referenced by a **defined_types** attribute.

**known_property_constraints**: the **constraint**s over a property that are referenced by the class or any of its known is-a superclass by their **constraint**s attribute, or, in case of a class that is a subtype of **a_priori_semantic_relationship**, by its **referenced_constraints** attribute.

**associated_items**: allows to access other kinds of data using the BSU mechanism.

Formal propositions:

**WR1**: the inheritance structure defined by the class hierarchy does not contain cycles.

**WR2**: only those properties that are visible for a class may become applicable to this class by virtue of being referenced by the **described_by** attribute.

**WR3**: only those data types that are visible for a class may become applicable to this class by virtue of being referenced by the **defined_types** attribute.

**WR4**: only those properties that are not applicable for a class by inheritance may become applicable to this class by virtue of being referenced by the **described_by** attribute.

**WR5**: only context dependent properties (**dependent_P_DET**) whose all context parameters (**condition_DET**) are applicable in to class may become applicable for this class by virtue of being referenced by its **described_by** attribute.

**WR6**: only those data types that are not applicable for a class by inheritance may become applicable to this class by virtue of being referenced by the **defined_types** attribute.

NOTE 9 The **its_superclass** attribute will be used to encode the class attribute "Superclass" (see 8.2.2).

NOTE 10 The **described_by** attribute provides the encoding for the "Applicable Properties" of a class (see 8.2.10).

NOTE 11 The **defined_types** attribute is used to encode the "Applicable Types" attribute of a class (see 8.2.7).

**WR7**: the set of constraints that are property constraints shall define restrictions that are compatible with the domain of values of the properties to which they apply.

**WR8:** all the properties referenced in the precondition of a **configuration_control_constraint** shall be applicable to the class.

**WR9:** all the properties referenced in the **constraint** attribute shall be either visible or applicable to the class.

**WR10:** either all **keyword**s are represented as **label_with_language**s or all are represented as **label**s.

**WR11**: if the **class** is not an **a_priori_semantic_relationship**, the **sub_class_properties** shall belong to the **described_by** list.

NOTE 12 Through an **a_priori_semantic_relationship**, **sub_class_properties** may also be imported.

**WR12**: the properties referenced in **class_constant_values** are declared as class-valued in some superclass of the current class, or in the current class itself.

NOTE 13 The **sub_class_properties** attribute of the **class** entity is used to encode the "Class valued properties" attribute for classes (see 8.2.8).

NOTE 14 The **class_constant_values** attribute of the **class** entity is used to encode the "Class Constant Values" for classes (see 8.2.11).

**WR13**: if a property referenced in **class_constant_values** was already assigned a value in a superclass, the value assigned in the current class should be the same.

Informal propositions:

**IP1**: if all the is-a superclasses of the class are available, then the **known_property_constraints** are all the constraints that apply to the properties that are connected to the class either as visible or as applicable properties.

### F.3.6.2 Item_class

The entity **item_class** enables the modeling of any type of entity of the application domain that may be captured by a characterization class defined by a class structure and a set of properties. In particular, both instances of products and instances of particular aspects of products represented as features, are mapped onto **item_class**.

The **item_class** entity includes an **instance_sharable** attribute that specifies the conceptual status of an item. If this attribute is true, then each instance represents an independent item, otherwise it is a feature, i.e., a dependent item that has to be component of another item. This does not prescribe any specific implementation at the data representation level.

EXAMPLE The *head of a screw* is a feature described by a number of properties but that may only exist when referenced by a screw. It is represented as an **item_class** with the **instance_sharable** attribute equal to *false*.

NOTE 1 Two subtypes of **item_class**, named **component_class** and **material_class**, were defined within the dictionary model of the first edition of this part of ISO 13584 and IEC 61360-2. These subtypes are deprecated, and they are removed from this edition of this part of ISO 13584.

NOTE 3 Another subtype of **item_class**, named **feature_class,** was provided in ISO 13584-24:2003. This subtype is also deprecated and its usage is not recommended in new implementations of this part of ISO 13584 and IEC 61360-2.

EXPRESS specification:

```
*)
ENTITY item_class
SUBTYPE OF(class);
      simplified_drawing: OPTIONAL graphics;
      coded_name: OPTIONAL value_code_type;
      instance_sharable: OPTIONAL BOOLEAN;
END_ENTITY; -- item_class
(*
```

Attribute definitions:

**simplified_drawing**: optional drawings (**graphics**) that can be associated to the described class.

NOTE 5 The **simplified_drawing** attribute of the **item_class** entity is used to encode the "Simplified Drawing" attribute for classes (see 8.2.16).

**coded_name**: may be used as a class constant value to characterize the class in the value domain of a **sub_class_properies** of its superclass.

NOTE 6 This attribute in not used in ISO 13584. It is only used in IEC 61360.

**instance_sharable**: when false, it specifies that instances of the **item_class** are features; when not provided or true it specifies that instances of the **item_class** are stand-alone items.

NOTE 7 In the common ISO13584/IEC61360 dictionary model, it is implementation dependent to decide whether several real world instances of features modeled by the same set of property-values pairs are represented by several EXPRESS pieces of data or by the same piece of data in the data exchange file. Thus, an instance of an **item_class** whose **instance_sharable** equals *false* and that is referenced by several instances of **item_class**es at the data model level is interpreted as several real world instances of the same feature.

### F.3.6.3 Categorization_class

The **categorization_class** entity enables the modeling of a grouping of a set of objects that constitutes an element of a categorization.

EXAMPLE 1 Manufacturing components and supplies, industrial optics, are example of product categorization class defined in UNSPSC.

Neither properties nor datatypes, nor constraints are associated, as visible or applicable, with such a class. Moreover, **categorization_class**es may not be related to each other by the is-a inheritance relationship, but they may only be related to each other through the is-case-of class relationship. A specific attribute, called **categorization_class_superclasses**, allows to record the **categorization_class**es that are superclasses of a **categorization_class** in a case-of hierarchy.

NOTE Using the case-of resource constructs, **item_class**es may also be connected to **categorization_class**es.

EXAMPLE 2 The following example shows how characterization classes and categorization classes may be connected to achieve some particular goals. A ball bearing supplier wants to design its own ontology and to make it easy to retrieve and easy to use. To achieve these goals, he/she wants to use standard properties and to be connected to standard classifications. The supplier provides only ball bearings, but some bearings are sealed, some others are not. Particular properties may be associated with sealed bearings and with not sealed bearings, but these categories do not exist as classes in standard bearing ontologies. Thus, the bearing supplier processes as follows. (1) He/she designs a proprietary ontology consisting of three characterization classes: *my_bearing, my_sealed_bearings, my_non_sealed_bearing*. The two latter are connected to the former by the is-a inheritance relationship, and all the properties assigned to the former are inherited by the latter. (2) To use some of the properties defined in the future ISO/TS 23768-1 Bearing — Parts library —

Reference dictionary, the bearing supplier specifies that his/her class *my_bearings* is case-of the standard bearing class *ball bearing* defined in ISO/TS 23768 . Through this case-of relationship, he/she may import in his/her class *my_bearings* the standard-defined properties: *bore diameter, outside diameter, ISO tolerance class.* Moreover, he/she creates those needed properties that are not defined in the standard. (3) To facilitate the retrieval of the server that display the supplier's catalogue, he/she represents a small fragment of the UNSPSC classification, and a case-of relationship between the UNSPSC class *ball_bearings* and its own class *my_bearing*. The result is presented in Figure 8 below:

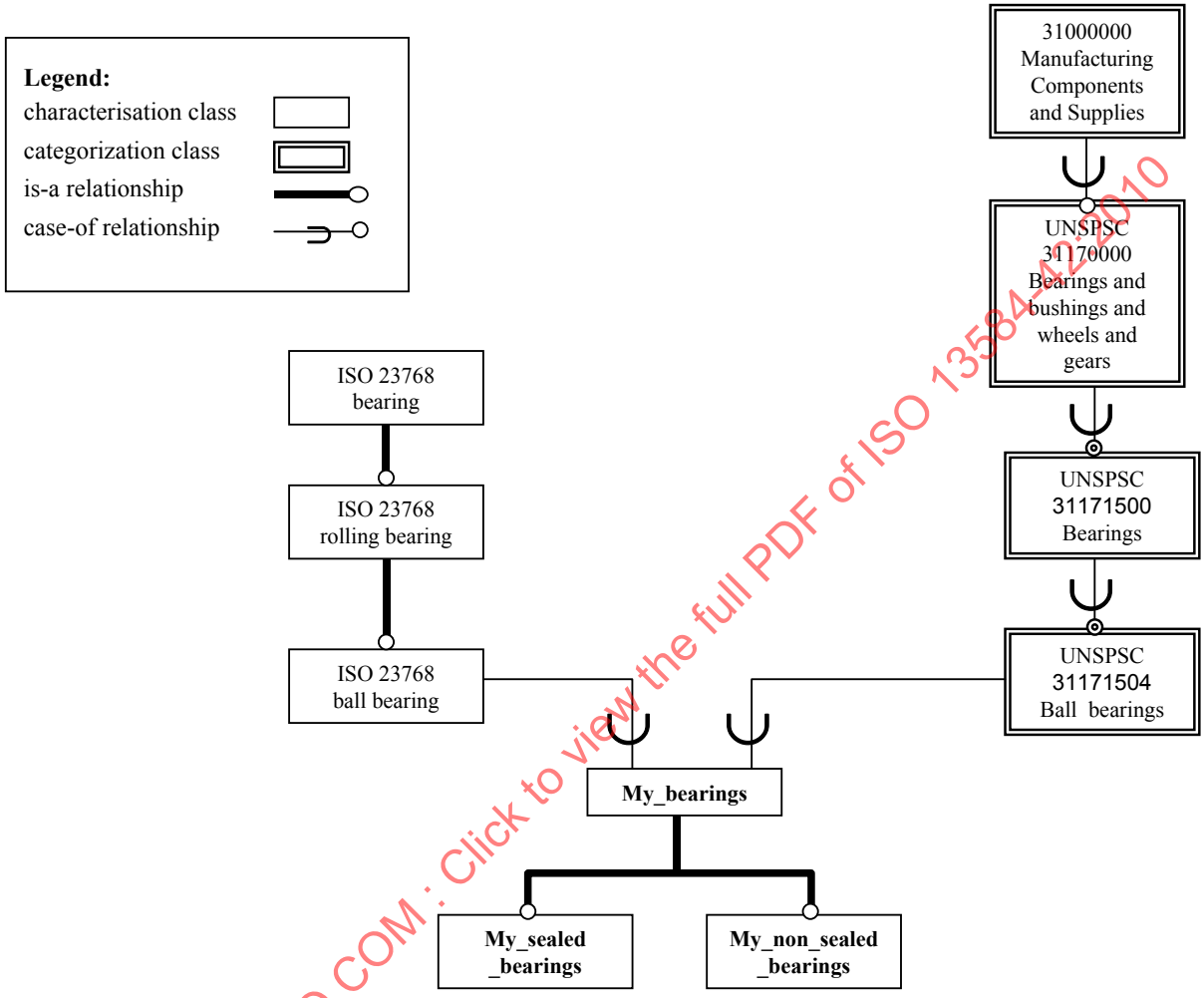



**Figure F.8 — Example of a supplier onbtology**



EXPRESS specification:

```
*)
ENTITY categorization_class
SUBTYPE OF(class);
      categorization_class_superclasses: SET [0:?] of class_BSU;
WHERE
      WR1: QUERY (cl <* SELF. categorization_class_superclasses
           | NOT (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
           +'.CATEGORIZATION_CLASS') IN TYPEOF(cl.definition[1])))
           = [];
      WR2: NOT EXISTS(SELF\class.its_superclass);
      WR3: SIZEOF(SELF\class.described_by) = 0;
      WR4: SIZEOF(SELF\class.defined_types) = 0;
      WR5: SIZEOF(SELF\class.constraints) = 0;
      WR6: SIZEOF(compute_known_visible_properties
```

```
            (SELF\dictionary_element.identified_by)) = 0;
        WR7: SIZEOF(SELF\class.sub_class_properties) = 0;
        WR8: SIZEOF(SELF\class.class_constant_values) = 0;
        WR9: SIZEOF(SELF\class.identified_by.known_visible_properties)
             = 0;
        WR10: SIZEOF(SELF\class.identified_by.known_visible_data_types)
             = 0;

END_ENTITY; -- categorization_class
(*
```

Attribute definitions:

**categorization_class_superclasses**: the **categorization_class**es that are one step above the categorization class in a case-of class hierarchy.

Formal propositions:

**WR1**: only **categorization_class**es may appear as superclasses of a **categorization_class**.

**WR2**: a **categorization_class** shall not have is-a superclass.

**WR3**: no property shall be associated with a **categorization_class**.

**WR4**: no datatype shall be associated with a **categorization_class**.

**WR5**: no constraint shall be associated with a **categorization_class**.

**WR6**: a **categorization_class** shall not be the property definition class of any property.

**WR7**: no subclass property shall be associated with a **categorization_class**.

**WR8**: no class constant value shall be associated with a **categorization_class**.

**WR9**: no visible property shall be associated with a **categorization_class**.

**WR10**: no visible datatype shall be associated with a **categorization_class**.

### F.3.7 Data Element Type / properties data

This clause contains definitions for the dictionary data for properties.

#### F.3.7.1 Property_BSU

The entity **property_BSU** provides for identification of a property.

EXPRESS specification:

```
*)
ENTITY property_BSU
SUBTYPE OF(basic_semantic_unit);
      SELF\basic_semantic_unit.code: property_code_type;
      name_scope: class_BSU;
DERIVE
      absolute_id: identifier :=
           name_scope.defined_by.absolute_id
           + sep_id + dic_identifier;
```

```
INVERSE
      describes_classes: SET OF class FOR described_by;
UNIQUE
      UR1: absolute_id;
WHERE
      WR1: QUERY(c <* describes_classes |
           NOT(is_subclass(c, name_scope.definition[1])))= [];
END_ENTITY; -- property_BSU
(*
```

Attribute definitions:

**code**: to allow for unique identification of the property over all ontologies defined by the same **name_scope.defined_by** supplier.

**name_scope**: the reference to the class at which or below which the property element is available for reference by the **described_by** attribute.

**absolute_id**: the unique identification of this property.

**describes_classes**: the classes declaring this property as available for use in the description of a product.

Formal propositions:

**WR1**: any class referenced by the **describes_classes** attribute of a **property_BSU** either is the class referenced by its **name_scope** attribute, or is a subclass of this class.

**UR1**: the property identifier **absolute_id** is unique.

NOTE The **name_scope** attribute of the **property_BSU** entity will be used to encode the "Definition Class" attribute for properties (see 7.2.2).

### F.3.7.2 Property_DET

The **property_DET** entity captures the dictionary description of properties.

EXPRESS specification:

```
*)
ENTITY property_DET
ABSTRACT SUPERTYPE OF(ONEOF(
      condition_DET, dependent_P_DET, non_dependent_P_DET))
SUBTYPE OF(class_and_property_elements);
      SELF\dictionary_element.identified_by: property_BSU;
      preferred_symbol: OPTIONAL mathematical_string;
      synonymous_symbols: SET [0:?] OF mathematical_string;
      figure: OPTIONAL graphics;
      det_classification: OPTIONAL DET_classification_type;
      domain: data_type;
      formula: OPTIONAL mathematical_string;
DERIVE
      describes_classes: SET [0:?] OF class
           := identified_by.describes_classes;
END_ENTITY; -- property_DET
(*
```

Attribute definitions:

**identified_by**: the **property_BSU** identifying this property.

**preferred_symbol**: a shorter description of this property.

**synonymous_symbols**: synonymous for the shorter description of the property.

**figure**: an optional **graphics** that describes the property.

**det_classification**: the ISO 80000/IEC 80000 (formerly ISO 31) class for this property.

**domain**: the reference to the **data_type** associated to the property.

**formula**: a mathematical expression for explaining the property.

**describes_classes**: the classes declaring this property as available for use in the description of a product.

NOTE 1 The **preferred_symbol** attribute is used to encode the "Preferred Letter Symbol" attribute for properties (see 7.2.6).

NOTE 2 The **synonymous_symbols** attribute is used to encode the "Synonymous Letter Symbol" attribute for properties (see 7.2.7).

NOTE 3 The **det_classification** attribute is used to encode the "Property Type Classification" attribute of a property (see 7.2.9).

NOTE 4 The **domain** attribute is used as a starting point for the encoding of the property attribute "Data Type" (see 7.2.3). The entity **data_type** will be subtyped for various possible data types.

NOTE 5 The **formula** attribute is used to encode the "Formula" attribute for properties (see 7.2.16).

Figure F.9 presents a planning model of the data associated with **property_DET**s.

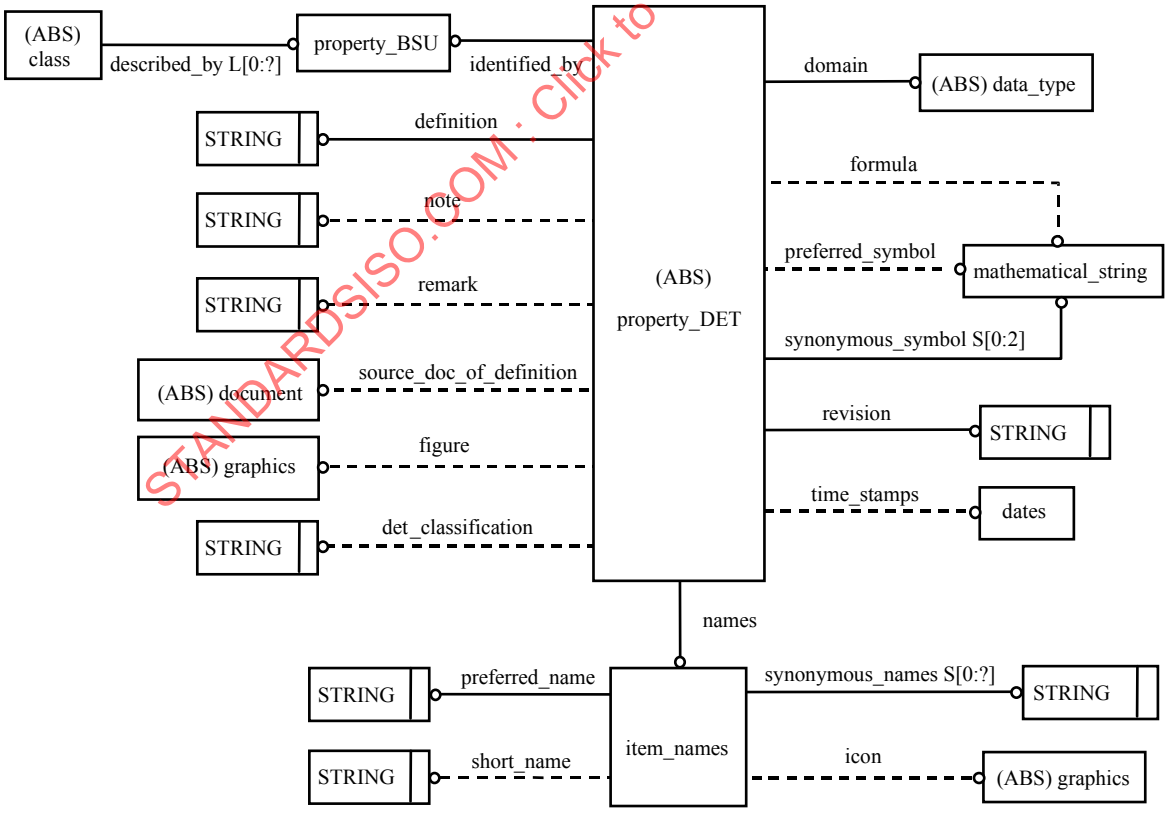



**Figure F.9 — Overview of property data element type data and relationships**

### F.3.7.3 Condition, dependent and non-dependent Data Element Types

Figure F.10 depicts the various kinds of Data Element Types in the format of a planning model.

Note that Figure F.10 is simplified: the "**depends_on**" relation essentially is implemented with a BSU reference, but a constraint is specified that the referred-to **property_DET** must be a **condition_DET** (see entity **dependent_P_DET**).

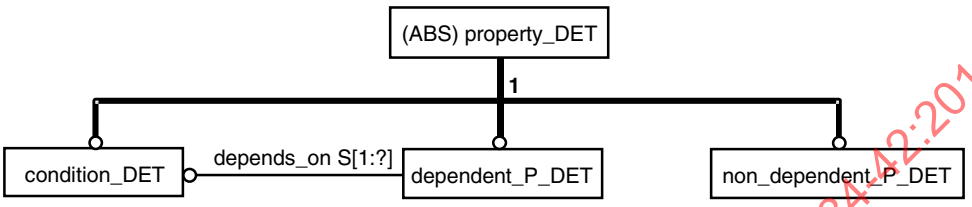



**Figure F.10 — Kinds of data element types**

#### F.3.7.3.1 Condition_DET

A **condition_DET** is a property on which other properties may depend upon.

EXPRESS specification:

```
*)
ENTITY condition_DET
SUBTYPE OF(property_DET);
END_ENTITY; -- condition_DET
(*
```

#### F.3.7.3.2 Dependent_P_DET

A **dependent_P_DET** is a property whose value depends explicitly on the value(s) of some condition(s).

EXAMPLE The resistance of a thermistor depends upon the ambiant temperature. Thermistor resistance should be presented as a **dependent_P_DET** and thermistor ambiant temperature as a **condition_DET**.

EXPRESS specification:

```
*)
ENTITY dependent_P_DET
SUBTYPE OF(property_DET);
     depends_on: SET [1:?] OF property_BSU;
WHERE
     WR1: QUERY(p <* depends_on | NOT(definition_available_implies(
          p, ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CONDITION_DET'
          IN TYPEOF(p.definition[1])))))) = [];
END_ENTITY; -- dependent_P_DET
(*
```

Attribute definitions:

**depends_on**: the set of basic semantic units identifying the properties on which this property depends on.

Formal propositions:

**WR1**: only **condition_DET**s shall be used in the **depends_on** set.

#### F.3.7.3.3 Non_dependent_P_DET

A **non_dependent_P_DET** is a property that does not depend explicitly on certain conditions.

EXPRESS specification:

```
*)
ENTITY non_dependent_P_DET
SUBTYPE OF(property_DET);
END_ENTITY; -- non_dependent_P_DET
(*
```

NOTE 1 The three subtypes **condition_DET**, **dependent_P_DET** and **non_dependent_P_DET** of the entity **property_DET** are used to encode the different kinds of properties (see Clause 7). **Condition_DET** is used for context parameters, **dependent_P_DET** is used for context dependent characteristics and the **non_dependent_P_DET** entity is used for product characteristics.

NOTE 2 The **depends_on** attribute of the **dependent_P_DET** entity is used to encode the "Condition" attribute for properties (see 7.2.15).

### F.3.7.4 Class_value_assignment

Class-valued properties are those properties whose value cannot be assigned individually for an instance of a class but can only be assigned for all instances belonging to a class. Such properties are declared by being included in the **sub_class_properties** list of an **item_class** entity. Then, such a property may be assigned a value for all instances of any **item_class** that is a subclass of the class where the class-valued property is declared, or in this class itself. A value of a class-valued property is assigned to an **item_class** by a **class_value_assignment** referenced by the **class_constant_values** attribute of this class.

NOTE Class-valued properties may be of any data type.

EXPRESS specification:

```
*)
ENTITY class_value_assignment;
      super_class_defined_property: property_BSU;
      assigned_value: primitive_value;
WHERE
      WR1: definition_available_implies(super_class_defined_property,
           compatible_data_type_and_value(super_class_defined_property.
           definition[1]\property_DET.domain, assigned_value));
END_ENTITY; -- class_value_assignment
(*
```

Attribute definitions:

**super_class_defined_property**: the reference to the property (defined in the class or in any of its superclasses as belonging to the **sub_class_properties** set) to which the **assigned_value** value is assigned.

**assigned_value**: the value assigned to the property, valid for the whole class referring this **class_value_assignment** instance in its **class_constant_values** set, and all its subclasses.

Formal proposition:

**WR1**: the value assigned to the **super_class_defined_property** shall be type compatible with the value domain of the **super_class_defined_property**.

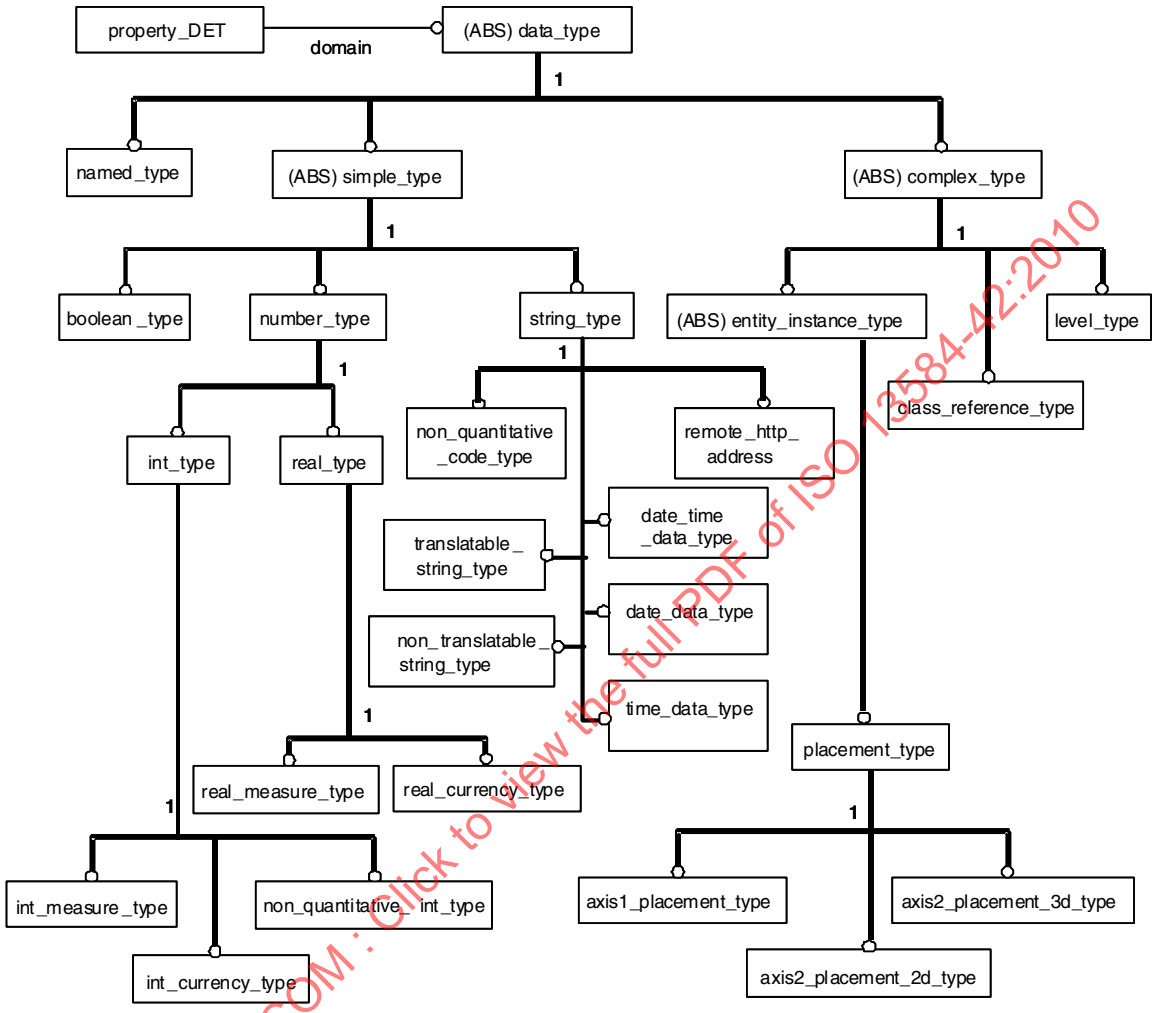



**Figure F.11 — Entity hierarchy for the type system**

## F.3.8 Domain data: the type system

This clause contains definitions for the representation of the data types of a **property_DET**. Figure F.11 outlines, as a planning model, the entity hierarchy for data types.

### F.3.8.1 General

In contrast to the other dictionary elements (Suppliers, Classes, Properties), an identification with the basic semantic unit concept is not mandatory for **data_type**, since it will be attached directly to the **property_DET** in many cases, and thus does not need an identification. However, the entities **data_type_BSU** and **data_type_element** allow for a unique identification where this is suitable. It provides for re-using the same type definition in another **property_DET** definition, even outside the current physical file.

#### F.3.8.1.1 Data_type_BSU

The **data_type_BSU** entity provides for identification of **data_type_element**s.

EXPRESS specification:

```
*)
ENTITY data_type_BSU
SUBTYPE OF(basic_semantic_unit);
      SELF\basic_semantic_unit.code: data_type_code_type;
      name_scope: class_BSU;
DERIVE
      absolute_id: identifier :=
           name_scope.defined_by.absolute_id     (* Supplier*)
           + sep_id + dic_identifier;       (* Data_type *)
INVERSE
      defining_class: SET OF class FOR defined_types;
UNIQUE
      absolute_id;
WHERE
      WR1: is_subclass(defining_class[1], name_scope.definition[1]);
END_ENTITY; -- data_type_BSU
(*
```

Attribute definitions:

**code**: to allow for unique identification of the data type over all ontologies defined by the same **name_scope.defined_by** supplier.

**name_scope**: the reference to the class at which or below which the data type element is available for reference by the **defined_types** attribute.

**absolute_id**: the unique identification of this property.

**defining_class**: the classes declaring this **data_type** as available for use in the description of a product.

Formal propositions:

**WR1**: the class used in the **name_scope** attribute is a superclass of the one where this **data_type** is defined.

NOTE The **name_scope** attribute is used to encode the reference to a class the related data type belongs to. This itself, beside the **data_type_element** entity (see below), is part of the encoding of the class attribute "Visible Types" (see 8.2.6).

#### F.3.8.1.2 Data_type_element

The **data_type_element** entity describes the dictionary element for types. Note that it is not necessary in every case to have BSU and **dictionary_element** for a certain **data_type**, because a **property_DET** can refer to the **data_type** directly. Usage of the BSU relation is only necessary when a supplier wants to refer to the same type in a different physical file.

EXPRESS specification:

```
*)
ENTITY data_type_element
SUBTYPE OF(dictionary_element);
      SELF\dictionary_element.identified_by: data_type_BSU;
      names: item_names;
      type_definition: data_type;
END_ENTITY; -- data_type_element
(*
```

Attribute definitions:

**identified_by**: the BSU that identifies the described **data_type_element**.

**names**: the names that allow the description of the defined **data_type_element**.

**type_definition**: the description of the type carried by the **data_type_element**.

NOTE The re-declared attribute **identified_by** is used to encode the reference to the BSU, this **data_type_element** is related to. This itself, beside the **data_type_BSU** entity (see above), is used to encode the class attribute "Visible Types" (see 8.2.6).

### F.3.8.2 The type system

#### F.3.8.2.1 Data_type

The **data_type** entity serves as a common supertype for the entities used to indicate the type of the associated DET.

EXPRESS specification:

```
*)
ENTITY data_type
ABSTRACT SUPERTYPE OF(ONEOF(
      simple_type,
      complex_type,
      named_type));
      constraints: SET [0:?] OF domain_constraint;
WHERE
      WR1: QUERY (cons <* constraints
           |NOT correct_constraint_type(cons, SELF)) = [];
END_ENTITY; -- data_type
(*
```

Attribute definitions:

**constraints**: the set of domain constraints that restrict the domain of values of the data type.

NOTE Each domain constraint in the **constraints** attribute must be fulfilled. Thus the **constraints** attribute is a conjunction of constraints.

Formal proposition:

**WR1**: the set of domain constraints shall define restrictions that are compatible with the domain of values of the data type.

#### F.3.8.2.2 Simple_type

The **simple_type** entity serves as a common supertype for the entities used to indicate a simple type of the associated DET.

EXPRESS specification:

```
*)
ENTITY simple_type
ABSTRACT SUPERTYPE OF(ONEOF(
      number_type,
      boolean_type,
      string_type))
SUBTYPE OF(data_type);
      value_format: OPTIONAL value_format_type;
END_ENTITY; -- simple_type
(*
```

Attribute definitions:

**value_format**: the optional encoding of the format of values for properties.

NOTE 1 The **value_format** attribute of the **simple_type** entity is used to encode the "Value Format" attribute for properties (see 7.2.17).

NOTE 2 If any **string_pattern_constraint** applies to the value of a simple type, then it takes precedence on the **value_format**.

#### F.3.8.2.3 Number_type

The **number_type** entity provides for values of DETs that are of type NUMBER.

EXPRESS specification:

```
*)
ENTITY number_type
ABSTRACT SUPERTYPE OF(ONEOF(
      int_type,
      real_type,
      rational_type))
SUBTYPE OF(simple_type);
END_ENTITY; -- number_type
(*
```

#### F.3.8.2.4 Int_type

The **int_type** entity provides for values of DETs that are of type INTEGER.

EXPRESS specification:

```
*)
ENTITY int_type
SUPERTYPE OF(ONEOF(
      int_measure_type,
      int_currency_type,
      non_quantitative_int_type))
SUBTYPE OF(number_type);
END_ENTITY; -- int_type
(*
```

#### F.3.8.2.5 Int_measure_type

The **int_measure_type** entity provides for values of DETs that are measures of type INTEGER. It specifies a **unit** or a unit identifier (**unit_id**), in which values exchanged as single integer are expressed. It may also specify alternative units, or alternative unit identifiers, that are also allowed for use when each value is explicitly associated with its unit.

NOTE 1 Either a **unit** or a **unit_id** is mandatory. In case where both are provided, the **unit** takes precedence.

NOTE 2 When both **alternative_units** and **alternative_unit_ids** are provided, both have the same size and the **alternative_units** attribute takes precedence.

NOTE 3 The **dic_unit_identifier** used in **unit_id** and in **alternative_unit_ids** attributes are unit identifiers that may resolved to a **dic_unit** from an ISO/TS 29002-20 server.

NOTE 4 Each **dic_unit** defined in the **alternative_units** attribute, and each **dic_unit** identified in the **alternative_unit_ids** attribute are required to be associated with a **string_representation** whose **text_representation** may be used for characterizing the alternative unit used at the instance level.

EXPRESS specification:

```
*)
ENTITY int_measure_type
SUBTYPE OF(int_type);
      unit: OPTIONAL dic_unit;
      alternative_units: OPTIONAL LIST [1:?] OF dic_unit;
      unit_id: OPTIONAL dic_unit_identifier;
      alternative_unit_ids: OPTIONAL LIST [1:?] OF dic_unit_identifier;
WHERE
      WR1: EXISTS(unit) OR EXISTS(unit_id);
      WR2: NOT EXISTS(alternative_units) OR
           NOT EXISTS(alternative_unit_ids) OR
           (SIZEOF(alternative_units) = SIZEOF(alternative_unit_ids));
      WR3: NOT EXISTS(alternative_units)
           OR (QUERY (un <* SELF.alternative_units
           |NOT EXISTS (un.string_representation))
           = [];
END_ENTITY; -- int_measure_type
(*
```

Attribute definitions:

**unit**: the default unit of reference associated with the value of the **int_measure_type**.

**alternative_units**: the list of other units that may be used to express the value of the **int_measure_type**.

NOTE 5 The list order is used to ensure that **alternative_units** and **alternative_unit_ids**, if both existsdefines the same unit in the same order.

**unit_id**: the identifier of the default unit of reference associated to the described measure.

NOTE 6 The attribute **unit** and the attribute **unit_id** are both used to encode the "Unit" attribute for properties (see 7.2). When both are provided, **unit** takes precedence

NOTE 7 If the value of a property whose domain is an **int_measure_type** is exchanged as a single integer number, this means that this value is expressed in the **unit** or **unit_id** unit of measure.

**alternative_unit_ids**: the list of identifiers of other units that may be used to express the value of the **int_measure_type**.

NOTE 8 When the value of a property whose domain is an **int_measure_type** is evaluated in a unit either defined by means of the **alternative_units** attribute or identified by means of the **alternative_unit_ids** attribute, its value cannot be represented as a single integer. It needs to be represented as a pair (value, unit).

Formal propositions:

**WR1**: one of the two attributes **unit** and **unit_**id shall exist.

**WR2**: if both attributes **alternative_units** and **alternative_unit_ids** exist, they shall have the same length.

**WR3**: each **dic_unit** in the **alternative_units** shall have a **string_representation**.

Informal propositions:

**IP1**: the **dic_unit_identifier**s used in **unit_id** and in **alternative_unit_ids** attributes shall be resolved to a **dic_unit** from an existing ISO/TS 29002-20 server.

**IP2**: when both **unit** and **unit_id** attributes are provided, they shall define the same unit.

**IP3**: when both **alternative_units** and **alternative_unit_ids** attributes are provided, they shall define the same list of units in the same order.

**IP4**: when the **alternative_unit_ids** attribute is provided, all the units the attribute identifies shall resolve to a **dic_unit** that has a **string_representation**.

#### F.3.8.2.6 Int_currency_type

The **int_currency_type** entity provides for values of DETs that are integer currencies.

EXPRESS specification:

```
*)
ENTITY int_currency_type
SUBTYPE OF(int_type);
      currency: OPTIONAL currency_code;
END_ENTITY; -- int_currency_type
(*
```

Attribute definitions:

**currency**: the associated code of the described currency according to ISO 4217 . If not present, the currency code has to be exchanged together with the data (values).

#### F.3.8.2.7 Non_quantitative_int_type

The **non_quantitative_int_type** entity is an enumeration type where elements of the enumeration are represented with an INTEGER value (see also entity **non_quantitative_code_type** and Figure F.12).

EXPRESS specification:

```
*)
ENTITY non_quantitative_int_type
SUBTYPE OF(int_type);
      domain: value_domain;
```

```
WHERE
      WR1: QUERY(v <* domain.its_values |
           'ISO13584_IEC61360_DICTIONARY_SCHEMA.VALUE_CODE_TYPE' IN
           TYPEOF(v.value_code)) = [];
END_ENTITY; -- non_quantitative_int_type
(*
```

Attribute definitions:

**domain**: the set of enumerated values described in the **value_domain** entity.

Formal propositions:

**WR1**: the values associated with the **domain.its_values** list shall not contain a **value_code_type**.

#### F.3.8.2.8 Real_type

The **real_type** entity provides for values of DETs that are of type REAL.

EXPRESS specification:

```
*)
ENTITY real_type
SUPERTYPE OF(ONEOF(
      real_measure_type,
      real_currency_type))
SUBTYPE OF(number_type);
END_ENTITY; -- real_type
(*
```

#### F.3.8.2.9 Real_measure_type

The **real_measure_type** entity provides for values of DETs that are measures of type REAL. It specifies a **unit** or a unit identifier, in which values exchanged as single real are expressed. It may also specify alternative units, or alternative unit identifiers, that are also allowed for use when each value is explicitly associated with its unit.

NOTE 1 Either a **unit** or a **unit_id** is mandatory. In case where both are provided, the **unit** takes precedence.

NOTE 2 When both **alternative_units** and **alternative_unit_ids** are provided, both have the same size and the **alternative_units** attribute takes precedence.

NOTE 3 The **dic_unit_identifier** used in **unit_id** and in **alternative_unit_ids** attributes are unit identifiers that may resolve to a **dic_unit** from an ISO/TS 29002-20 server.

NOTE 4 Each **dic_unit** defined in the **alternative_units** attribute, and each **dic_unit** identified in the **alternative_unit_ids** attribute are required to be associated with a **string_representation**, whose **text_representation** may be used for characterizing the alternative unit used at the instance level.

EXPRESS specification:

```
*)
ENTITY real_measure_type
SUBTYPE OF(real_type);
      unit: OPTIONAL dic_unit;
      alternative_units: OPTIONAL LIST [1:?] OF dic_unit;
```

```
        unit_id : OPTIONAL dic_unit_identifier;
        alternative_unit_ids: OPTIONAL LIST [1:?] OF dic_unit_identifier;
    WHERE
        WR1: EXISTS(unit) OR EXISTS(unit_id);
        WR2: NOT EXISTS(alternative_units)
             OR NOT EXISTS(alternative_unit_ids)
             OR (SIZEOF(alternative_units) =
             SIZEOF(alternative_unit_ids));
        WR3: NOT EXISTS(alternative_units)
             OR (QUERY (un <* SELF.alternative_units
             |NOT EXISTS (un.string_representation))
             = []);
    END_ENTITY; -- real_measure_type
    (*
```

Attribute definitions:

**unit**: the default unit of reference associated with the value of the **real_measure_type**.

**alternative_units**: the list of other units that may be used to express the value of the **real_measure_type**.

NOTE 5 The list order is used to ensure that **alternative_units** and **alternative_unit_ids**, if both exist, define the same unit in the same order.

**unit_id**: the identifier of the default unit of reference associated to the described measure.

NOTE 6 The attribute **unit** and the attribute **unit_id** are both used to encode the "Unit" attribute for properties (see 6.2). When both are provided, **unit** takes precedence.

NOTE 7 If the value of a property whose domain is a **real_measure_type** is exchanged as a single real number, this means that this value is expressed in the **unit** or **unit_id** unit of measure.

**alternative_unit_ids**: the list of identifiers of other units that may be used to express the value of the **real_measure_type**.

NOTE 8 When the value of a property whose domain is a **real_measure_type** is evaluated in a unit either defined by means of the **alternative_units** attribute or identified by means of the **alternative_unit_ids** attribute, its value cannot be represented as a single real. It needs to be represented as a pair (value, unit).

Formal propositions:

**WR1**: one of the two attributes **unit** and **unit_id** shall exist.

**WR2**: if both attributes **alternative_units** and **alternative_unit_ids** exist, they shall have the same length.

**WR3**: each **dic_unit** in the **alternative_units** shall have a **string_representation**.

Informal propositions:

**IP1**: the **dic_unit_identifier**s used in **unit_id** and in **alternative_unit_ids** attributes shall resolve to a **dic_unit** from an existing ISO/TS 29002-20 server.

**IP2**: when both **unit** and **unit_id** attributes are provided, they shall define the same unit.

**IP3**: when both **alternative_units** and **alternative_unit_ids** attributes are provided, they shall define the same list of units in the same order.

**IP4**: when the **alternative_unit_ids** attribute is provided, all the units the attribute identifies shall resolve to a **dic_unit** that has a **string_representation**.

#### F.3.8.2.10 Real_currency_type

The **real_currency_type** entity defines real currencies.

EXPRESS specification:

```
*)
ENTITY real_currency_type
SUBTYPE OF(real_type);
     currency: OPTIONAL currency_code;
END_ENTITY; -- real_currency_type
(*
```

Attribute definitions:

**currency**: the associated code of the described currency according to ISO 4217. If not present, the currency code has to be exchanged together with the data (values).

#### F.3.8.2.11 Rational_type

The **rational_type** entity provides for values of DETs that are of type rational.

NOTE In ISO 13584-32, rational values are represented by three integer XML elements: the whole part, the numerator and the denominator. In ISO 13584-24:2003 rational values are represented by an array of three integers: the whole part, the numerator and the denominator.

EXPRESS specification:

```
*)
ENTITY rational_type
SUPERTYPE OF(
     rational_measure_type)
SUBTYPE OF(number_type);
END_ENTITY; -- rational_type
(*
```

#### F.3.8.2.12 Rational_measure_type

The **rational_measure_type** entity provides for values of DETs that are measures of type RATIONAL.

EXAMPLE Screw diameter: 4 1/8 inches.

The **rational_measure_type** entity specifies a **unit** or a unit identifier, in which values exchanged as rational are expressed. It may also specify alternative units, or alternative unit identifiers, that are also allowed for use when each value is explicitly associated with its unit.

NOTE 1 Either a **unit** or a **unit_id** is mandatory. In case where both are provided, the **unit** takes precedence.

NOTE 2 When both **alternative_units** and **alternative_unit_ids** are provided, both have the same size and the **alternative_units** attribute takes precedence.

NOTE 3 The **dic_unit_identifier**s used in **unit_id** and in **alternative_unit_ids** attributes are unit identifiers that may resolve to a **dic_unit** from an ISO/TS 29002-20 server.

NOTE 4 Each **dic_unit** defined in the **alternative_units** attribute, and each **dic_unit** identified in the **alternative_unit_ids** attribute are required to be associated with a **string_representation**, whose **text_representation** may be used for characterizing the alternative unit used at the instance level.

EXPRESS specification:

```
*)
ENTITY rational_measure_type
SUBTYPE OF(rational_type);
      unit: OPTIONAL dic_unit;
      alternative_units: OPTIONAL LIST [1:?] OF dic_unit;
      unit_id: OPTIONAL dic_unit_identifier;
      alternative_unit_ids: OPTIONAL LIST [1:?] OF dic_unit_identifier;
WHERE
      WR1: EXISTS(unit) OR EXISTS(unit_id);
      WR2: NOT EXISTS(alternative_units)
           OR NOT EXISTS(alternative_unit_ids)
           OR (SIZEOF(alternative_units) =
           SIZEOF(alternative_unit_ids));
      WR3: NOT EXISTS(alternative_units) OR (QUERY (un <*
           SELF.alternative_units |
           NOT EXISTS (un.string_representation)) = []);
END_ENTITY; -- rational_measure_type
(*
```

Attribute definitions:

**unit**: the default unit of reference associated with the value of the **rational_measure_type**.

**alternative_units**: the list of other units that may be used to express the value of the **rational_measure_type**.

NOTE 5 The list order is used to ensure that **alternative_units** and **alternative_unit_ids**, if both exist defines the same unit in the same order.

**unit_id**: the identifier of the default unit of reference associated to the described measure.

NOTE 6 The attribute **unit** and the attribute **unit_id** are both used to encode the "Unit" attribute for properties (see 6.2). When both are provided, **unit** takes precedence.

NOTE 7 If the value of a property whose domain is a **rational_measure_type** is exchanged as a single rational number, this means that this value is expressed in the **unit** or **unit_id** unit of measure.

**alternative_unit_ids**: the list of identifiers of other units that may be used to express the value of the **rational_measure_type**.

NOTE 8 When the value of a property whose domain is a **rational_measure_type** is evaluated in a unit either defined by means of the **alternative_units** attribute or identified by means of the **alternative_unit_ids** attribute, its value cannot be represented as a single rational. It needs to be represented as a pair (value, unit).

Formal propositions:

**WR1**: one of the two attributes **unit** and **unit_**id shall exist.

**WR2**: if both attributes **alternative_units** and **alternative_unit_ids** exist, they shall have the same length.

**WR3**: each **dic_unit** in the **alternative_units** shall have a **string_representation**.

Informal propositions:

**IP1**: the **dic_unit_identifier**s used in **unit_id** and in **alternative_unit_ids** attributes shall resolve to a **dic_unit** from an existing ISO/TS 29002-20 server.

**IP2**: when both **unit** and **unit_id** attributes are provided, they shall define the same unit.

**IP3**: when both **alternative_units** and **alternative_unit_ids** attributes are provided, they shall define the same list of units in the same order.

**IP4**: when the **alternative_unit_ids** attribute is provided, all the units the attribute identifies shall resolve to a **dic_unit** that has a **string_representation**.

#### F.3.8.2.13 boolean_type

The **boolean_type** entity provides for values of DETs that are of type BOOLEAN.

EXPRESS specification:

```
*)
ENTITY boolean_type
SUBTYPE OF(simple_type);
END_ENTITY; -- boolean_type
(*
```

#### F.3.8.2.14 String_type

The **string_type** provides for values of DETs that are of type STRING.

EXPRESS specification:

```
*)
ENTITY string_type
SUPERTYPE OF ( ONEOF (
      translatable_string_type,
      non_translatable_string_type,
      URI_type,
      non_quantitative_code_type,
      date_time_data_type,
      date_data_type,
      time_data_type))
SUBTYPE OF(simple_type);
END_ENTITY; -- string_type
 (*
```

#### F.3.8.2.15 Translatable_string_type

The **translatable_string_type** provides for values of DETs that are of type STRING, but that are supposed to be represented as different strings in different languages.

NOTE 1 Values of such properties cannot be used for product identification.

NOTE 2 Values of such properties may be either a simple **string_value** when a **global_language_assignment** defines a current language, or a **translated_string_value** where each string value is associated with a language.

NOTE 3 Two values of the same property whose **data_type** is **translatable_string_type** may only be compared for equality if the corresponding property as a **source_language** defined as part of its **administrative_data** and if these values are available in this **source_language**. It is not assumed that in languages different from this **source_language** the same meaning is represented by the same string.

EXPRESS specification:

```
*)
ENTITY translatable_string_type
SUBTYPE OF(string_type);
END_ENTITY; -- translatable_string_type
(*
```

#### F.3.8.2.16 Non_translatable_string_type

The **non_translatable_string_type** provides for values of DETs that are of type STRING, but that are represented in the same way in any language.

NOTE Values of such properties can be used for product identification.

EXPRESS specification:

```
*)
ENTITY non_translatable_string_type
SUBTYPE OF(string_type);
END_ENTITY; -- non_translatable_string_type
(*
```

#### F.3.8.2.17 URI_type

The **URI_type** provides for values of DETs that are of type STRING, but contains a URI.

NOTE A **URI_type** allows in particular to provide URL.

EXPRESS specification:

```
*)
ENTITY URI_type
SUBTYPE OF(string_type);
END_ENTITY; -- URI_type
(*
```

#### F.3.8.2.18 Date_time_data_type

The **date_time_data_type** provides for values of DETs that are of type STRING, but contains a specific instant of time specified according to a particular representation compliant with ISO 8601.

NOTE 1 Only a subset of the lexical representations allowed by ISO 8601 is allowed for values of **date_time_data_type**. This is specified by IP1.

NOTE 2 The above restriction of ISO 8601 representations is the same as the one defined by XML Schema.

EXPRESS specification:

```
*)
ENTITY date_time_data_type
SUBTYPE OF(string_type);
END_ENTITY; -- date_time_data_type
(*
```

Informal propositions:

**IP1**: the value of a property whose data type is **date_time_data_type** shall comply with the following lexical representation, which is a subset of the lexical representations allowed by ISO 8601. This lexical representation is the ISO 8601 extended format CCYY-MM-DDThh:mm:ss where "CC" represents the century order, the order of the first century being "00", "YY" the year, "MM" the month and "DD" the day, preceded by an optional leading "-" sign to indicate a negative number. If the sign is omitted, "+" is assumed. The letter "T" is the date/time separator and "hh", "mm", "ss" represent hour, minute and second respectively. Additional digits can be used to increase the precision of fractional seconds if desired i.e., the format ss.ss... with any number of digits after the decimal point is supported. The fractional second part is optional; other parts of the lexical form are not optional. To accommodate year values greater than 9999 additional digits can be added to the left of this representation. Leading zeros are required if the year value would otherwise have fewer than four digits; otherwise they are forbidden. The year 0000 is prohibited. The CCYY field must have at least four digits, the MM, DD, SS, hh, mm and ss fields exactly two digits each (not counting fractional seconds); leading zeroes must be used if the field would otherwise have too few digits. This representation may be immediately followed by a "Z" to indicate Coordinated Universal Time (UTC) or, to indicate the time zone, i.e. the difference between the local time and Coordinated Universal Time, immediately followed by a sign, + or -, followed by the difference from UTC represented as hh:mm (note: the minutes part is required). See ISO 8601 for details about legal values in the various fields. If the time zone is included, both hours and minutes must be present.

EXAMPLE To indicate 1:20 p.m. on May the 31st, 1999 for Eastern Standard Time which is 5 hours behind Coordinated Universal Time (UTC), one would write: `1999-05-31T13:20:00-05:00`.

#### F.3.8.2.19 Date_data_type

The **date_data_type** provides for values of DETs that are of type STRING, but contains a specific calendar date specified according to a particular representation compliant with ISO 8601.

NOTE 1 Only a subset of the lexical representations allowed by ISO 8601 is allowed for values of **date_data_type**. This is specified by IP1.

NOTE 2 The above restriction of ISO 8601 representations is the same as the one defined by XML Schema.

EXPRESS specification:

```
*)
ENTITY date_data_type
SUBTYPE OF(string_type);
END_ENTITY; -- date_data_type
(*
```

Informal propositions:

**IP1**: the value of a property whose data type is **date_data_type** shall comply with the following lexical representation, which is a subset of the lexical representations allowed by ISO 8601. The lexical representation for **date_data_type** is the reduced (right truncated) lexical representation for **date_time_data_type**: CCYY-MM-DF. No left truncation is allowed. An optional following time zone qualifier is allowed as for **date_time_data_type**. To accommodate year values outside the range from 0001 to 9999, additional digits can be added to the left of this representation and a preceding "-" sign is allowed.

EXAMPLE 1999-05-31 is the **date_data_type** representation of: May the 31st, 1999.

#### F.3.8.2.20 Time_data_type

The **time_data_type** provides for values of DETs that are of type STRING, but contains a specific time specified according to a particular representation compliant with ISO 8601. A value of **time_data_type** represents an instant of time that recurs every day.

NOTE 1 Only a subset of the lexical representations allowed by ISO 8601 is allowed for values of **time_data_type**. This is specified by IP1.

NOTE 2 The above restriction of ISO 8601 representations is the same as the one defined by XML Schema.

NOTE 3 Since the lexical representation allows an optional time zone indicator, **time_data_type** values are partially ordered because it may not be able to determine the order of two values one of which has a time zone and the other does not.

EXPRESS specification:

```
*)
ENTITY time_data_type
SUBTYPE OF(string_type);
END_ENTITY; -- time_data_type
(*
```

Informal propositions:

**IP1**: the value of a property whose data type is **time_data_type** shall comply with the following lexical representation, which is a subset of the lexical representations allowed by ISO 8601. The lexical representation for **time_data_type** is the left truncated lexical representation for **date_time_data_type** hh:mm:ss.sss with optional following time zone indicator.

EXAMPLE 13:20:00-05:00 is the **time_data_type** representation of 1.20 p.m. for Eastern Standard Time which is 5 hours behind Coordinated Universal Time (UTC),

#### F.3.8.2.21 Non_quantitative_code_type

The **non_quantitative_code_type** entity is an enumeration type where elements of the enumeration are represented with a STRING value (see also ENTITY **non_quantitative_int_type** and Figure F.12).

EXPRESS specification:

```
*)
ENTITY non_quantitative_code_type
SUBTYPE OF(string_type);
      domain: value_domain;
WHERE
      WR1: QUERY(v <* domain.its_values |
           NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA.VALUE_CODE_TYPE' IN
           TYPEOF(v.value_code))) = [];
END_ENTITY; -- non_quantitative_code_type
(*
```

Attribute definitions:

**domain**: the set of enumerated values described in the **value_domain** entity.

Formal propositions:

**WR1**: the values associated with the **domain.its_values** list shall only contain elements of type **value_code_type**.

#### F.3.8.2.22 Complex_type

The **complex_type** entity provides for the definition of types of which the values are represented as EXPRESS instances.

EXPRESS specification:

```
*)
ENTITY complex_type
ABSTRACT SUPERTYPE OF(ONEOF(
      level_type,
      class_reference_type,
      entity_instance_type))
SUBTYPE OF(data_type);
END_ENTITY; -- complex_type
(*
```

#### F.3.8.2.23 Level_type

The **level_type** is a complex type indicating that the value of a property consists of one up to four real measure or integer measure values, each one qualified by a particular indicator specifying the meaning of the value.

NOTE 1 Instance values of a **level_type** contain values for and only for the indicators specified in the **levels** attribute. When some of these values are not available, they are represented by **null_value**s.

EXAMPLE If the **level_type** specifies that only *minimum* and *typical* values are to be provided as integer, any instance contains integer values (or **null_value**s) only for the *minimum* and *typical* levels of the **level_type** instance value.

EXPRESS specification:

```
*)
ENTITY level_type
SUBTYPE OF(complex_type);
      levels: LIST [1:4] OF UNIQUE level;
      value_type: simple_type;
WHERE
      WR1: ('ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_MEASURE_TYPE'
           IN TYPEOF(value_type))
           OR ('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_MEASURE_TYPE'
           IN TYPEOF(value_type));
      WR2: NOT EXISTS(SELF.levels[2]) OR
            (SELF.levels[1] < SELF.levels[2]);
      WR3: NOT EXISTS(SELF.levels[2]) OR NOT EXISTS(SELF.levels[3]) OR
           (SELF.levels[2] < SELF.levels[3]);
      WR4: NOT EXISTS(SELF.levels[3]) OR NOT EXISTS(SELF.levels[4]) OR
           (SELF.levels[3] < SELF.levels[4]);
END_ENTITY; -- level_type
(*
```

Attribute definitions:

**levels**: the list of unique indicators that specifies which qualified values shall be associated with the property.

**value_type**: the data type of the qualified values of the **level_type**.

Formal propositions:

**WR1**: the **SELF.value_type** shall be of type **int_measure_type** or of type **real_measure_type**.

**WR2**: the order of the first and second **level**, if both exist, shall follow the enumeration order of the **level** type.

**WR3**: the order of the second and third **level**, if both exist, shall follow the enumeration order of the **level** type.

**WR4**: the order of the third and fourth **level**, if both exist, shall follow the enumeration order of the **level** type.

#### F.3.8.2.24 Level

The **level** entity specifies the various indicators that may be used to qualify a value in a **level_type**.

These indicators are as follows:

— minimum: lowest value specified of a quantity, established for a specified set of operating conditions at which a component, device or equipment is operable and performs according to specified requirements;

— nominal: value of a quantity used to designate and identify a component, device, equipment, or system;

— typical: commonly encountered value of a quantity used for specification purposes, established for a specified set of operating conditions of a component, device, equipment, or system;

— maximum: highest value specified of a quantity, established for a specified set of operating conditions at which a component device or equipment is operable and performs according to specified requirements.

NOTE 1 The nominal value is generally a rounded value.

EXAMPLE A 12 V (nominal) car battery has 6 cells with a typical voltage of about 2.2 V each, giving a typical battery voltage of about 13.5 V. On charge, the voltage may reach a maximum of about 14.5 V but it is considered fully discharged when the voltage falls below a minimum of 12.5

NOTE 2 The values that shall be provided for a level type-valued property are specified in the dictionary.

NOTE 3 It is advised that the use of the level type is restricted to those DETs that are applicable in domains where the reporting of multiple values on a single characteristic is recognized as common practice and requested, as is true for the electronic component industry.

EXPRESS specification:

```
*)
TYPE level = ENUMERATION OF(
     min,      (* the minimal value of the physical quantity *)
     nom,      (* the nominal value of the physical quantity *)
     typ,      (* the typical value of the physical quantity *)
     max);     (* the maximal value of the physical quantity *)
END_TYPE; -- level
(*
```

#### F.3.8.2.25 Class_reference_type

The **class_reference_type** entity provides for values of DETs that are represented as instances of a **class**. It is used, in particular, for the description of assemblies or to describe the material a (part of a) component consists of.



EXPRESS specification:

```
*)
ENTITY class_reference_type
SUBTYPE OF(complex_type);
      domain: class_BSU;
END_ENTITY; -- class_reference_type
(*
```

Attribute definitions:

**domain**: the **class_BSU** referring to the **class** representing the described type.

#### F.3.8.2.26 Entity_instance_type

The **entity_instance_type** entity provides for values of DETs that are represented as instances of some EXPRESS entity data types. A **type_name** attribute enables the specification what are the allowed data types. These data types are strings contained in a set. This attribute, together with the EXPRESS TYPEOF function applied to the value, permits strong type checking and polymorphism. This entity will be subtyped below for some data types that are allowed for use in the dictionary schema.

NOTE When an EXPRESS entity is the value of a given DET the data type of which is an **entity_instance_type**, it is possible to check the correct typing by applying the EXPRESS TYPEOF function on this DET value and compare the results of this TYPEOF application with the strings contained in the **type_name** set attribute.

EXPRESS specification:

```
*)
ENTITY entity_instance_type
SUBTYPE OF(complex_type);
      type_name: SET OF STRING;
END_ENTITY; -- entity_instance_type
(*
```

Attribute definitions:

**type_name**: the set of strings that describe, in the format of the EXPRESS TYPEOF function, the EXPRESS entity data type names that shall belong to the result of the EXPRESS TYPEOF function when it is applied to a value that references the present entity as its data type.

#### F.3.8.2.27 Placement_type

The **placement_type** entity provides for values of DETs that are instances of **placement** entity data type.

NOTE 1 **Placement** entities are imported from ISO 10303-42 . According to ISO 10303-42 , an instance of **placement** may only exist if it is related to an instance of **geometric_representation_context** in some instance of **representation**. Therefore, if some class properties have instances of **placement** as their values, this class shall contain a **geometric_representation_context** (that defines the context of these placements) and a **representation** (that gathers these **placement**s with their context). Both **geometric_representation_context** and **representation** being not imported in this part of ISO 13584, the **placement** entities cannot be used when only ISO 13584-42 schemas are used. These entities are introduced as resources for the other parts of ISO 13584.

NOTE 2 **Placement** entities are in particular used in ISO 13584-32 (OntoML) and in ISO 13584-25.

EXPRESS specification:

```
*)
ENTITY placement_type
SUPERTYPE OF(ONEOF(
      axis1_placement_type,
      axis2_placement_2d_type,
      axis2_placement_3d_type))
SUBTYPE OF(entity_instance_type);
WHERE
      WR1: 'GEOMETRY_SCHEMA.PLACEMENT'
           IN SELF\entity_instance_type.type_name;
END_ENTITY; -- placement_type
(*
```

Formal propositions:

**WR1**: the string 'GEOMETRY_SCHEMA.PLACEMENT' shall be contained in the set defined by the **SELF\entity_instance_type.type_name** attribute.

#### F.3.8.2.28 Axis1_placement_type

The **axis1_placement_type** entity provides for values of DETs that are instances of **axis1_placement** entity data type. (See ISO 10303-42 for details).

EXPRESS specification:

```
*)
ENTITY axis1_placement_type
SUBTYPE OF(placement_type);
WHERE
      WR1: 'GEOMETRY_SCHEMA.AXIS1_PLACEMENT' IN
           SELF\entity_instance_type.type_name;
END_ENTITY; -- axis1_placement_type
(*
```

Formal propositions:

**WR1**: the string 'GEOMETRY_SCHEMA.AXIS1_PLACEMENT' shall be contained in the set defined for the **SELF\entity_instance_type.type_name** attribute.

#### F.3.8.2.29 Axis2_placement_2d_type

The **axis2_placement_2d_type** entity provides for values of DETs that are instances of **axis2_placement_2d** entity data type (See ISO 10303-42 for details).

EXPRESS specification:

```
*)
ENTITY axis2_placement_2d_type
SUBTYPE OF(placement_type);
WHERE
      WR1: 'GEOMETRY_SCHEMA.AXIS2_PLACEMENT_2D'
           IN SELF\entity_instance_type.type_name;
END_ENTITY; -- axis2_placement_2d_type
(*
```

Formal propositions:

**WR1**: the string 'GEOMETRY_SCHEMA.AXIS2_PLACEMENT_2D' shall be contained in the set defined for the **SELF\entity_instance_type.type_name** attribute.

#### F.3.8.2.30 Axis2_placement_3d_type

The **axis2_placement_3d_type** entity provides for values of DETs that are instances of **axis2_placement_3d** entity data type. (See ISO 10303-42 for details).

EXPRESS specification:

```
*)
ENTITY axis2_placement_3d_type
SUBTYPE OF(placement_type);
WHERE
     WR1: 'GEOMETRY_SCHEMA.AXIS2_PLACEMENT_3D'
          IN SELF\entity_instance_type.type_name;
END_ENTITY; -- axis2_placement_3d_type
(*
```

Formal propositions:

**WR1**: the string 'GEOMETRY_SCHEMA.AXIS2_PLACEMENT_3D' shall be contained in the set defined for the **SELF\entity_instance_type.type_name** attribute.

#### F.3.8.2.31 Named_type

The **named_type** entity provides for referring to other types via the BSU mechanism.

EXPRESS specification:

```
*)
ENTITY named_type
SUBTYPE OF(data_type );
     referred_type: data_type_BSU;
END_ENTITY; -- named_type
(*
```

Attribute definitions:

**referred_type**: the BSU identifying the **data_type** the present entity refers to.

### F.3.8.3 Values

This clause contains definitions for non-quantitative data element types (see **non_quantitative_int_type** and **non_quantitative_code_type** entities).

Figure F.12 outlines, as a planning model, the main data associated with non-quantitative data element types.

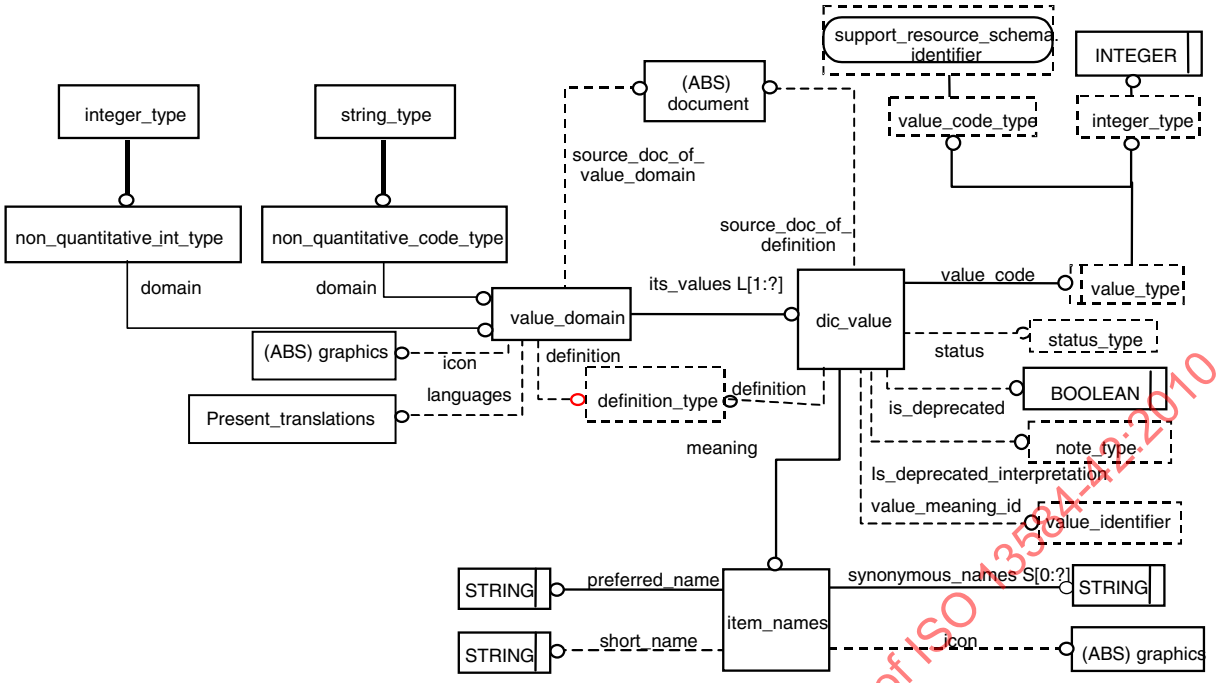



**Figure F.12 — Overview of non-quantitative data element types**



### F.3.8.3.1 Value_domain

The **value_domain** entity describes the set of allowed values for a non-quantitative data element type.

EXPRESS specification:

```
*)
ENTITY value_domain;
      its_values: LIST [1:?] OF dic_value;
      source_doc_of_value_domain: OPTIONAL document;
      languages: OPTIONAL present_translations;
      terms: LIST [0:?] OF item_names;
      definition: OPTIONAL definition_type;
      icon: OPTIONAL graphics;
WHERE
      WR1: NOT EXISTS(languages) OR (QUERY(v <* its_values |
           languages :<>: v.meaning.languages) = []);
      WR2: codes_are_unique(its_values);
      WR3: EXISTS(languages) OR (QUERY(v <* its_values |
           EXISTS(v.meaning.languages)) = []);
      WR4: EXISTS(languages) OR (QUERY(v <* its_values |
           EXISTS(v.definition.languages)) = []);
END_ENTITY; -- value_domain
(*
```

Attribute definitions:

**its_values**: the enumeration list of values contained in the described domain.

**source_doc_of_value_domain**: the document describing the domain associated to the described **value_domain** entity.

**languages**: the optional languages in which the translations are provided.

**terms**: the list of **item_names** to allow for IEC 61360 the link to the terms dictionary.

**definition**: the optional text describing the **value_domain**.

**icon**: an optional **icon** which graphically represents the description associated with the **value_domain**.

Formal propositions:

**WR1**: if the value meanings are provided in more than one language, then the set of languages used must be the same for the whole set of values.

**WR2**: value codes must be unique within this data type.

**WR3**: if no **languages** is provided, the value meanings shall not be assigned any language.

**WR4**: if no **languages** is provided, the value definition shall not be assigned any language.

#### F.3.8.3.2 Value_type

Each value of a non-quantitative data element is associated with a code that characterizes the value. A **value_type** may be either an INTEGER or a **value_code_type**.

EXPRESS specification:

```
*)
TYPE integer_type = INTEGER;
END_TYPE; -- integer_type

TYPE value_type = SELECT(value_code_type, integer_type);
END_TYPE; -- value_type
(*
```

#### F.3.8.3.3 Dic_value

The **dic_value** entity is one of the values of a **value_domain** entity.

EXPRESS specification:

```
*)
ENTITY dic_value;
      value_code: value_type;
      meaning: item_names;
      source_doc_of_definition: OPTIONAL document;
      definition: OPTIONAL definition_type;
      status: OPTIONAL status_type;
      is_deprecated: OPTIONAL BOOLEAN;
      is_deprecated_interpretation: OPTIONAL note_type;
      value_meaning_id: OPTIONAL dic_value_identifier;
WHERE
      WR1: NOT EXISTS (SELF. is_deprecated)
                OR EXISTS (SELF. is_deprecated_interpretation);
END_ENTITY; -- dic_value
(*
```

Attribute definitions:

**value_code**: the code associated to the described value. It can be either a **value_code_type** or an **integer_type**.

**meaning**: the meaning associated to this value. It is provided by names.

**source_doc_of_definition**: the optional source document in which the value is defined.

**definition**: the optional text describing the **dic_value**.

**status**: a **status_type** that defines the life cycle state of the **dic_value**.

NOTE 1 Allowed values of a **status_type** are not standardized. They are defined for each particular dictionary by its information supplier.

EXAMPLE 1 A set of allowed values for the status of items proposed for standardization to an ISO standard maintenance agency are defined in the ISO directives.

EXAMPLE 2 A set of allowed values for the status of items in an IEC database standard is defined in the IEC directives.

NOTE 2 For those **dic_value**s that are not yet released for insertion, representation of draft **dic_value**s might be useful.

EXAMPLE 3 For experimentation purposes before validation.

NOTE 3 If the **status** attribute is not provided for a **dic_value**, and if the use of this **dic_value** is not deprecated as denoted by a possible **is_deprecated** attribute, then the **dic_value** has the same standardization status as the whole dictionary. In particular, if the dictionary is standardized, this **dic_value** is part of the current edition of the standard.

**is_deprecated**: a Boolean that specifies, when true, that the **dic_value** shall no longer be used.

NOTE 4 When **is_deprecated** has no value, the **dic_value** is not deprecated.

NOTE 5 Deprecated **dic_value**s are left in the **value_domain**s for upward compatibility reasons.

**is_deprecated_interpretation**: specify the deprecation rationale and how instance values of the deprecated element should be interpreted.

**value_meaning_id**: a **dic_value_identifier** that is a global identifier of the **dic_value**, independently of the **value_domain** in which it is included.

NOTE 6 This identifier allows to reuse the same **dic_value** definition in various domains.

Formal proposition:

**WR1**: when **is_deprecated** exists, **is_deprecated_interpretation** shall exist.

Informal proposition:

**IP1**: instance values of **is_deprecated_interpretation** element shall be defined at the time where deprecation decision was taken.

#### F.3.8.3.4 Administrative_data

An **administrative_data** entity records information about the life cycle of a dictionary element.

EXPRESS specification:

```
*)
ENTITY administrative_data;
      status: OPTIONAL status_type;
      translation: LIST[0:?] of translation_data;
      source_language: language_code;
INVERSE
      administrated_element: dictionary_element FOR administration;
WHERE
      WR1: one_language_per_translation(SELF);
      WR2: SIZEOF(QUERY (trans <* SELF.translation |
           trans.language = source_language)) = 0;
END_ENTITY; -- administrative_data
(*
```

Attribute definitions:

**status:** a **status_type** that defines the life cycle state of the dictionary element.

NOTE 1 Allowed values of a **status_type** are not standardized. They are defined for each particular dictionary by its information supplier.

EXAMPLE 1 A set of allowed values for the status of items proposed for standardization to an ISO standard maintenance agency are defined in the ISO directives.

EXAMPLE 2 A set of allowed values for the status of items in an IEC database standard is defined in the IEC directives.

NOTE 2 For those **dictionary_element**s that are not yet released for insertion, representation of draft **dictionary_element**s might be useful.

EXAMPLE For experimentation purposes before validation.

NOTE 3 If the **status** attribute is not provided, and if this **dictionary_element** use is not deprecated as denoted by a possible **is_deprecated** attribute, then the **dictionary_element** has the same standardization status as the whole dictionary. In particular, if the dictionary is standardized, this **dictionary_element** is part of the current edition of the standard.

**translation**: description of responsible translators in the various languages.

**source_language**: the language in which the **dictionary_element** was initially defined and that provides the reference meaning in case of translation discrepancy.

NOTE 4 A dictionary may contain **dictionary_element**s whose **source_language**s are different, for instance because they where imported from different dictionaries. It is the responsibility of the dictionary data supplier to ensure that the information about these various elements is provided in the same language or languages.

**administrated_element**: the **dictionary_element** of which life cycle data are recorded in the **administrative_data**.

Formal propositions:

**WR1**: the languages of translation associated to an **administrative_data** are unique.

**WR2**: the **source_language** is not present in the languages of translation associated to an **administrative_data**.

#### F.3.8.3.5 Translation_data

The **translation_data** entity records information about the possible translations of a dictionary element.

EXPRESS specification:

```
*)
ENTITY translation_data;
      language: language_code;
      responsible_translator: supplier_BSU;
      translation_revision: revision_type;
      date_of_current_translation_revision: OPTIONAL date_type;
INVERSE
      belongs_to: administrative_data FOR translation;
END_ENTITY; -- translation_data
(*
```

Attribute definitions:

**language**: the language in which the dictionary element is translated.

NOTE In case of discrepancy between the initial definition of a dictionary element and some of its translation, the actual meaning of the dictionary element is the one of the source definition language.

**responsible_translator**: the organization responsible for the translation in the language element.

**translation_revision**: the revision number of the corresponding translation.

NOTE Change of **version** or change of **revision** of a dictionary element does not always require any change in their translations. If there is no change in a translation due to a change of **version** or change of **revision** of a dictionary element, the corresponding **translation_revision** shall not be changed. However any change of a translation will imply change of the corresponding **translation_revision**.

**date_of_current_translation**: the date of the last revision of the corresponding translation.

**belongs_to**: the **administrative_data** that references the **translation_data** record.

### F.3.8.4 Extension to ISO 10303-41 unit definitions

This clause defines the resources for description of units in a dictionary. It extends the resources defined in ISO 10303-41.

#### F.3.8.4.1 Non_si_unit

The **non_si_unit** entity extends the unit model of ISO 10303-41 to allow for the representation of non-SI-units that are neither context dependent, nor conversion-based (See ISO 10303-41 for details).

EXPRESS specification:

```
*)
ENTITY non_si_unit
SUBTYPE OF(named_unit);
      name: label;
END_ENTITY; -- non_si_unit
(*
```

Attribute definitions:

**name**: the **label** used to name the described unit.

#### F.3.8.4.2 Assert_ONEOF rule

The **assert_ONEOF** rule asserts that ONEOF holds between the following subtypes of **named_unit**: **si_unit**, **context_dependent_unit**, **conversion_based_unit**, **non_si_unit**.

EXPRESS specification:

```
*)
RULE assert_ONEOF FOR(named_unit);
WHERE
      QUERY(u <* named_unit |
            ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NON_SI_UNIT'
            IN TYPEOF(u)) AND
            ('MEASURE_SCHEMA.SI_UNIT' IN TYPEOF(u))
            OR ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NON_SI_UNIT'
            IN TYPEOF(u)) AND
            ('MEASURE_SCHEMA.CONTEXT_DEPENDENT_UNIT' IN TYPEOF(u))
            OR ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NON_SI_UNIT'
            IN TYPEOF(u)) AND
            ('MEASURE_SCHEMA.CONVERSION_BASED_UNIT' IN TYPEOF(u))
            ) = [];
END_RULE; -- assert_ONEOF
(*
```

#### F.3.8.4.3 Dic_unit

The basic representation of units is in structured form according to ISO 10303-41. But since one of the purposes of storing units in the dictionary is for the presentation to the user, a structured representation alone is not sufficient. It must be supplemented by a string representation.

The present definitions allow various possibilities:

— the function **string_for_unit** (see F.3.10 "Function Definitions") can be used. For a given structured representation of a unit, it returns a string representation corresponding to the one used in Annex B of IEC 61360-1:2009 ;

— a string representation can be supplied in plain text form (entity **mathematical_string**, attribute **text_representation**);

— a MathML representation can be supplied to allow for an enhanced presentation of the unit including sub- and superscripts etc. (entity **mathematical_string**, attribute **MathML_representation**).

The **dic_unit** entity describes a unit to be stored in a dictionary.

EXPRESS specification:

```
*)
ENTITY dic_unit;
      structured_representation: unit;
      string_representation: OPTIONAL mathematical_string;
END_ENTITY; -- dic_unit
(*
```

Attribute definitions:

**structured_representation**: structured representation, from ISO 10303-41, including extension defined in F.3.8.4 "Extension to ISO 10303-41 definitions".

**string_representation**: the function **string_for_unit** can be used to compute a string representation from the **structured_representation**, for the case where no **string_representation** is present.

NOTE The **structured_representation** attribute of the entity **dic_unit** is used to encode the property attribute "Unit" (see 7.2.14).

## F.3.9 Basic type and entity definitions

This subclause contains the basic type and entity definitions that were used in the main part of the model.

### F.3.9.1 Basic type definitions

This subclause contains the basic type and entity definitions, sorted alphabetically.

#### F.3.9.1.1 Class_code_type

The **class_code_type** identifies the allowed values for a class code.

EXPRESS specification:

```
*)
TYPE class_code_type = code_type;
WHERE
     WR1: LENGTH(SELF) <= class_code_len;
END_TYPE; -- class_code_type
(*
```

Formal propositions:

**WR1**: the length of values corresponding to **class_code_type** shall be less or equal to the length of **class_code_len** (i.e., 35).

#### F.3.9.1.2 Code_type

The **code_type** identifies the allowed values for a code type used to identify.

NOTE If the code is also intended to be exchanged using ISO/TS 29002-5, it is recommended to fulfil the requirements defined by this standard. For a code, only "safe characters" are allowed. Safe characters include: upper case letters, digits, colons, periods, or underscore. Moreover, the minus character '-' is allowed for particular purposes

EXPRESS specification:

```
*)
TYPE code_type = identifier;
WHERE
     WR1: NOT(SELF LIKE '*#*');
     WR2: NOT(SELF LIKE '* *');
     WR3: NOT(SELF = '');
END_TYPE; -- code_type
(*
```

Formal propositions:

**WR1**: the '#' shall not be contained in a **code_type** value. '#' is used to concatenate identifiers, (see: CONSTANT **sep_id**), or code and version (see: CONSTANT **sep_cv**).

**WR2**: spaces are not allowed, to avoid problems with leading and trailing blanks when concatenating codes.

**WR3**: a **code_type** shall not be an empty string.

#### F.3.9.1.3 Currency_code

The **currency_code** identifies the values allowed for a currency code.

These values are defined according to ISO 4217 .

EXAMPLE Values are: "CHF" for Swiss Francs, "CNY" for Yuan Renminbi (Chinese), "JPY" for Yen (Japanese), "SUR" for SU Rouble, "USD" for US Dollars, "EUR" for EURO.

EXPRESS specification:

```
*)
TYPE currency_code = identifier;
WHERE
     WR1: LENGTH(SELF) = 3;
END_TYPE; -- currency_code
(*
```

Formal propositions:

**WR1**: the length of a **currency_code** value shall be equal to 3.

#### F.3.9.1.4 Data_type_code_type

The **data_type_code_type** identifies the values allowed for a data type code.

EXPRESS specification:

```
*)
TYPE data_type_code_type = code_type;
WHERE
     WR1: LENGTH(SELF) = data_type_code_len;
END_TYPE; -- data_type_code_type
(*
```

Formal propositions:

**WR1**: the length of a **data_type_code_type** value shall be equal to the value of a **data_type_code_len** (i.e., 35).

#### F.3.9.1.5 Date_type

The **date_type** identifies the values allowed for a date.

These values are defined according to ISO 8601

EXAMPLE "1994-03-21".

EXPRESS specification:

```
*)
TYPE date_type = STRING(10) FIXED;
WHERE
     WR1: SELF LIKE '####-##-##';
END_TYPE; -- date_type
(*
```

#### F.3.9.1.6 Definition_type

The **definition_type** identifies the values allowed for a definition.

EXPRESS specification:

```
*)
TYPE definition_type = translatable_text;
END_TYPE; -- definition_type
(*
```

#### F.3.9.1.7 DET_classification_type

The **DET_classification_type** identifies the values allowed for a DET classification. These values are used for DET classification according to ISO 80000/IEC 80000 (formerly ISO 31).

EXPRESS specification:

```
*)
TYPE DET_classification_type = identifier;
WHERE
     WR1: LENGTH(SELF) = DET_classification_len;
END_TYPE; -- DET_classification_type
(*
```

Formal propositions:

**WR1**: the length of a **DET_classification_type** value shall be equal to the value of a **DET_classification_len** (i.e., 3).

#### F.3.9.1.8 Note_type

The **note_type** identifies the values allowed for a note.

EXPRESS specification:

```
*)
TYPE note_type = translatable_text;
END_TYPE; -- note_type
(*
```

#### F.3.9.1.9 Pref_name_type

The **pref_name_type** identifies the values allowed for a preferred name.

EXPRESS specification:

```
*)
TYPE pref_name_type = translatable_label;
WHERE
     WR1: check_label_length(SELF, pref_name_len);
END_TYPE; -- pref_name_type
(*
```

Formal propositions:

**WR1**: the length of a **pref_name_type** value shall not exceed the length of **pref_name_len** (i.e., 255).

#### F.3.9.1.10 Property_code_type

The **property_code_type** identifies the values allowed for a property code.

EXPRESS specification:

```
*)
TYPE property_code_type = code_type;
WHERE
     WR1: LENGTH(SELF) <= property_code_len;
END_TYPE; -- property_code_type
(*
```

Formal propositions:

**WR1**: the length of a **property_code_type** value shall not exceed the length of **property_code_len** (i.e., 35).

#### F.3.9.1.11 Remark_type

The **remark_type** identifies the values allowed for a remark.

EXPRESS specification:

```
*)
TYPE remark_type = translatable_text;
END_TYPE; -- remark_type
(*
```

#### F.3.9.1.12 Hierarchical_position_type

The **hierarchical_position_type** identifies the values allowed for a hierarchical position.

EXPRESS specification:

```
*)
TYPE hierarchical_position_type = identifier;
END_TYPE; -- hierarchical_position_type
(*
```

NOTE Representation of a hierarchical position in a **hierarchical_position_type** is based on some coding conventions. This coding convention is not defined by this part of ISO 13584.

#### F.3.9.1.13 Revision_type

The **revision_type** identifies the values allowed for a revision.

NOTE When a new version is issued, its revision value is set to '0'.

EXPRESS specification:

```
*)
TYPE revision_type = code_type;
WHERE
     WR1: LENGTH(SELF) <= revision_len;
     WR2: EXISTS(VALUE(SELF)) AND ('INTEGER' IN TYPEOF(VALUE(SELF)))
          AND (VALUE(SELF) >= 0);
END_TYPE; -- revision_type
(*
```

Formal propositions:

**WR1**: the length of a **revision_type** value shall not exceed the length of **revision_len** (i.e., 3).

**WR2**: the **revision_type** shall contain digits only and the integer it represents shall be a natural integer.

#### F.3.9.1.14 Short_name_type

The **short_name_type** identifies the values allowed for a short name.

EXPRESS specification:

```
*)
TYPE short_name_type = translatable_label;
WHERE
     WR1: check_label_length(SELF, short_name_len);
END_TYPE; -- short_name_type
(*
```

Formal propositions:

**WR1**: the length of a **short_name_type** value shall not exceed the length of **short_name_len** (i.e., 30).

#### F.3.9.1.15 Supplier_code_type

The **supplier_code_type** identifies the values allowed for a supplier code.

NOTE If the supplier code is also intended to be exchanged using ISO/TS 29002-5, the various parts of the supplier code as defined by ISO 6523 (ICD, OI, OPI, OPIS and AI) have to be separated by '-'.

EXPRESS specification:

```
*)
TYPE supplier_code_type = code_type;
WHERE
     WR1: LENGTH(SELF) <= supplier_code_len;
END_TYPE; -- supplier_code_type
(*
```

Formal propositions:

**WR1**: the length of a **supplier_code_type** value shall not exceed the length of **supplier_code_len** (i.e., 149).

#### F.3.9.1.16 Syn_name_type

The **syn_name_type** identifies the values allowed for a synonymous name.

EXPRESS specification:

```
*)
TYPE syn_name_type = SELECT(label_with_language, label);
WHERE
     WR1: check_syn_length(SELF, syn_name_len);
END_TYPE; -- syn_name_type
(*
```

Formal propositions:

**WR1**: the length of a **syn_name_type** value shall not exceed the length of **syn_name_len** (i.e., 255).

#### F.3.9.1.17 Keyword_type

The **keyword_type** identifies the values allowed for a keyword.

EXPRESS specification:

```
*)
TYPE keyword_type = SELECT(label_with_language, label);
END_TYPE; -- keyword_type
(*
```

#### F.3.9.1.18 ISO_29002_IRDI_type

The **ISO_29002_IRDI_type** is a global identifier that identifies an administrated item in a registry. The structure of this identifier complies with identifier syntax defined in ISO/TS 29002-5.

NOTE 1 An **ISO_29002_IRDI_type** may be used for any kind of information considered in ISO/TS 29002-5 and that may be associated with an IRDI identifier. Three special cases are identified below because they are specifically used in the **ISO13584_IEC61360_dictionary_schema**: **constraint_identifier**, **dic_unit_identifier** and **dic_value_identifier.**

EXPRESS specification:

```
*)
TYPE ISO_29002_IRDI_type = identifier;
WHERE
     WR1: LENGTH (SELF) <= 290;
END_TYPE; -- syn_name_type
(*
```

Formal propositions:

**WR1**: as specified in ISO/TS 29002-5, the length of the identifier shall not be greater than 290.

Informal propositions:

**IP1**: the identifier shall fulfil the requirements specified in ISO/TS 29002-05 for an "international registration data identifier" (IRDI).

NOTE 2 According to ISO/TS 29002-05 an IRDI consists either of a string that do not contain the "#' character, to identify an organization, or of three sub-strings that do not contain the '#' character and that are separated by the '#' character to identify any other administrated item.

NOTE 3 In the case where the IRDI is not used for identifying an organization:

— the first sub-string, called the called the Registration Authority Identifier (*RAI*), identifies the organization which is responsible for the administration of the administrated item;

— the second sub-string, called the Data Identifier (*DI*), contains both a categorization of the administrated item, represented by two characters followed by the minus ('-') sign as defined in ISO/TS 29002-5 (for instance: class, property, unit), and the identifier assigned to the administrated item by the RAI;

— the third sub-string corresponds to the Version Identifier (*VI*) of the IRDI.

#### F.3.9.1.19 Constraint_identifier

The **constraint_identifier** is an **ISO_29002_IRDI_type** identifier that provides a global identifier to a constraint represented as a **constraint** entity. The structure of this identifier complies with identifier syntax defined in ISO/TS 29002-5.

NOTE A **constraint_identifier** may be associated with a resolution service compliant with ISO/TS 29002-20 . This service would be able to return a formal definition of the constraint identified by the **constraint_identifier** compliant with the **constraint** EXPRESS model in the syntax defined by ISO 13584-32 (OntoML), and possibly in ISO 10303-21 syntax.

EXPRESS specification:

```
*)
TYPE constraint_identifier = ISO_29002_IRDI_type;
END_TYPE; -- constraint_identifier
(*
```

Informal propositions:

**IP1**: the part of the identifier after the second '#' character, that is the Data Identifier, shall start by '04-' to identify a constraint as specified in ISO/TS 29002-5.

#### F.3.9.1.20 Dic_unit_identifier

The **dic_unit_identifier** is an **ISO_29002_IRDI_type** identifier that identifies a unit of which the **dic_unit** representation shall be downloadable from an ISO/TS 29002-20 server. The structure of this identifier complies with identifier syntax defined in ISO/TS 29002-5.

EXPRESS specification:

```
*)
TYPE dic_unit_identifier = ISO_29002_IRDI_type;
END_TYPE; -- dic_unit_identifier
(*
```

Informal propositions:

**IP1**: a **dic_unit_identifier** shall be associated with a resolution service compliant with ISO/TS 29002, and this service shall be able to return a formal definition of the unit identified by the **dic_unit_identifier** compliant

with the **dic_unit** EXPRESS model in the syntax defined by ISO 13584-32 (OntoML), and possibly in the ISO 10303-21 syntax.

**IP2**: the part of the identifier after the second '#' character, that is the Data Identifier, shall start by '05-' to identify a unit as specified in ISO/TS 29002-5.

NOTE A **dic_unit_identifier** constitutes an International Registration Data Identifier (IRDI) as defined by ISO/IEC 11179-5 .

#### F.3.9.1.21 Dic_value_identifier

The **dic_value_identifier** is an **ISO_29002_IRDI_type** identifier that provides a global identifier to a property value represented as a **dic_value** entity. The structure of this identifier complies with identifier syntax defined in ISO/TS 29002-5.

NOTE 1 Assigning a **dic_value_identifier** allows to reuse the same **dic_value** definition in several **value_domains**.

NOTE 2 A **dic_value_identifier** may be associated with a resolution service compliant with ISO/TS 29002. This service would be able to return a formal definition of the value identified by the **dic_value_identifier** compliant with the **dic_value** EXPRESS model in the syntax defined by ISO 13584-32 (OntoML), and possibly in ISO 10303-21 syntax.

EXPRESS specification:

```
*)
TYPE dic_value_identifier = ISO_29002_IRDI_type;
END_TYPE; -- dic_value_identifier
(*
```

Informal proposition:

**IP1**: the part of the identifier after the second '#' character, that is the Data Identifier, shall start by '07-' to identify a property value as specified in ISO/TS 29002-5.

NOTE 3 A **dic_value_identifier** constitutes an International Registration Data Identifier (IRDI) as defined by ISO/IEC 11179-5 .

#### F.3.9.1.22 Value_code_type

The **value_code_type** identifies the values allowed for a value code.

EXPRESS specification:

```
*)
TYPE value_code_type = identifier;
WHERE
     WR1: LENGTH(SELF) <= value_code_len;
END_TYPE; -- value_code_type
(*
```

Formal propositions:

**WR1**: the length of a **value_code_type** value shall not exceed the length of **value_code_len** (i.e., 35).

#### F.3.9.1.23 Value_format_type

The **value_format_type** identifies the values allowed for a value format. These values are defined according to Annex D.

EXPRESS specification:

```
*)
TYPE value_format_type = identifier;
WHERE
     WR1: LENGTH(SELF) <= value_format_len;
END_TYPE; -- value_format_type
(*
```

Formal propositions:

**WR1**: the length of a **value_format_type** value shall not exceed the length of **value_format_len** (i.e., 80).

#### F.3.9.1.24 Version_type

The **version_type** identifies the values allowed for a version.

EXPRESS specification:

```
*)
TYPE version_type = code_type;
WHERE
     WR1: LENGTH(SELF) <= version_len;
     WR2: EXISTS(VALUE(SELF)) AND ('INTEGER' IN TYPEOF(VALUE(SELF)))
          AND (VALUE(SELF) >= 0);
END_TYPE; -- version_type
(*
```

Formal propositions:

**WR1**: the length of a **version_type** value shall not exceed the length of **version_len** (i.e., 10).

**WR2**: the **version_type** shall contain digits only.

#### F.3.9.1.25 Status_type

The **status_type** identifies the values allowed for a status. Allowed values of a **status_type** are not standardized. They shall be defined for each particular dictionary by the supplier of dictionary data.

EXAMPLE 1 A set of allowed values for the status of items proposed for standardization to an ISO standard maintenance agency are defined in the ISO directives.

EXAMPLE 2 A set of allowed values for the status of items in an IEC database standard is defined in the IEC directives.

NOTE A status may be associated both with a **dictionary_element** or with a **dic_value**.

EXPRESS specification:

```
*)
TYPE status_type = identifier;
END_TYPE; -- status_type
(*
```

#### F.3.9.1.26 Dictionary_code_type

The **dictionary_code_type** is a **code_type** that identifies a dictionary.

EXPRESS specification:

```
*)
TYPE dictionary_code_type = identifier;
WHERE
     WR1: LENGTH(SELF) <= dictionary_code_len;
END_TYPE; -- dictionary_code_type
(*
```

Formal propositions:

**WR1**: the length of a **dictionary_code_type** value shall not exceed the length of **dictionary_code_len** (i.e., 131).

### F.3.9.2 Basic entity definitions

This subclause contains the basic entity definitions, sorted alphabetically.

#### F.3.9.2.1 Dates

The **dates** entity describes the three dates associated respectively to the first stable description, to the current version and to the current revision for a given description.

NOTE For each particular dictionary management rules, it is the responsibility of the dictionary information supplier to choose which point in time corresponds to the first stable description of an item.

EXPRESS specification:

```
*)
ENTITY dates;
     date_of_original_definition: date_type;
     date_of_current_version: date_type;
     date_of_current_revision: OPTIONAL date_type;
END_ENTITY; -- dates
(*
```

Attribute definitions:

**date_of_original_definition**: the date associated to the first stable version of an item.

**date_of_current_version**: the date associated to the present version.

**date_of_current_revision**: the date associated to the last revision.

#### F.3.9.2.2 Document

The **document** entity is an abstract resource that stands for a document. The dictionary schema only provides for exchanging the identification of documents (see below). The **document** entity may also be subtyped with entities implementing a means for exchanging document data.

EXAMPLE By reference to an external file and exact specification of the format of the file.

EXPRESS specification:

```
*)
ENTITY document
ABSTRACT SUPERTYPE;
END_ENTITY; -- document
(*
```

#### F.3.9.2.3 Graphics

The **graphics** entity is an abstract resource to be subtyped with entities implementing a means for exchanging graphical data,

EXAMPLE By reference to an external file and exact specification of the format of the file.

EXPRESS specification:

```
*)
ENTITY graphics
ABSTRACT SUPERTYPE;
END_ENTITY; -- graphics
(*
```

#### F.3.9.2.4 External_graphics

The **external_graphics** entity provides for exchanging graphical data by means of external files referenced by a **graphic_files** entity.

EXPRESS specification:

```
*)
ENTITY external_graphics
SUBTYPE OF (graphics);
      representation: graphic_files;
END_ENTITY; -- external_graphics
(*
```

Attribute definitions:

**representation**: representation of a graphics by means of external files.

#### F.3.9.2.5 Graphic_files

A **graphic_files** is an **external_item** whose content is a picture.

NOTE 1 An **external_item** entity, defined in ISO 13584-24:2003, is an item whose content may be provided as library external file(s). It refers to an **external_file_protocol** that specifies how the library external file(s) shall be processed, and to an **external_content** that specifies the library external file(s) that represents its content.

NOTE 2 Both **external_file_protocol** and **external_content** entities are defined in ISO 13584-24:2003.

NOTE 3 Only **external_content**s that consist of **http_file**s and only the **http_protocol external_file_protocol**s are referenced by the **ISO13584_IEC61360_dictionary_schema** and may be used in the context of this part of ISO 13584.

NOTE 4 The files of a **graphic_files** may depend upon the language; this is specified by the following subtypes of external_content: **not_translatable_external_content**, **not_translated_external_content** and **translated_external_content**.

NOTE 5 **http_file**, **http_protocol**, **not_translatable_external_content**, **not_translated_external_content** and **translated_external_content**, are defined in ISO 13584-24:2003 and referenced by the **ISO13584_IEC61360_dictionary_schema**.

EXPRESS specification:

```
*)
ENTITY graphic_files
SUBTYPE OF (external_item);
END_ENTITY; -- graphic_files
(*
```

### F.3.9.2.6 Identified_document

The **identified_document** entity describes a document identified by its label.

EXPRESS specification:

```
*)
ENTITY identified_document
SUBTYPE OF(document);
      document_identifier: translatable_label;
WHERE
      WR1: check_label_length(SELF.document_identifier,source_doc_len);
END_ENTITY; -- identified_document
(*
```

Attribute definitions:

**document_identifier**: the label of the described document.

Formal propositions:

**WR1**: the length of a **document_identifier** value shall not exceed the length of **source_doc_len** (i.e., 255).

### F.3.9.2.7 Item_names

The **item_names** entity identifies the names that can be associated to a given description. It states the preferred name, the set of synonymous names, the short name and the **languages** in which the different names are provided. It may be associated with an icon.

EXPRESS specification:

```
*)
ENTITY item_names;
      preferred_name: pref_name_type;
      synonymous_names: SET OF syn_name_type;
      short_name: OPTIONAL short_name_type;
      languages: OPTIONAL present_translations;
      icon: OPTIONAL graphics;
```

```
WHERE
      WR1: NOT(EXISTS(languages )) OR (
           ('ISO13584_IEC61360_LANGUAGE_RESOURCE_SCHEMA'
           + '.TRANSLATED_LABEL' IN TYPEOF(preferred_name))
           AND (languages :=:
           preferred_name\translated_label.languages)
           AND (NOT(EXISTS(short_name)) OR
           ('ISO13584_IEC61360_LANGUAGE_RESOURCE_SCHEMA'
           + '.TRANSLATED_LABEL' IN TYPEOF(short_name))
           AND (languages :=: short_name\translated_label.languages))
           AND (QUERY(s <* synonymous_names |
           NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
           '.LABEL_WITH_LANGUAGE' IN TYPEOF(s))) = []));
      WR2: NOT EXISTS(languages) OR (QUERY(s <* synonymous_names |
           EXISTS(s.language) AND NOT(s.language IN
           QUERY(l <* languages.language_codes | TRUE
           ))) = []);
      WR3: EXISTS(languages) OR
           (('SUPPORT_RESOURCE_SCHEMA.LABEL' IN
           TYPEOF(preferred_name))
           AND (NOT(EXISTS(short_name)) OR
           ('SUPPORT_RESOURCE_SCHEMA.LABEL' IN
           TYPEOF(short_name)))
           AND (QUERY(s <* synonymous_names |
           'ISO13584_IEC61360_DICTIONARY_SCHEMA.LABEL_WITH_LANGUAGE' IN
           TYPEOF(s)) = []));
END_ENTITY; -- item_names
(*
```

Attribute definitions:

**preferred_name**: the name that is preferred for use.

**synonymous_names**: the set of synonymous names.

**short_name**: the abbreviation of the preferred name.

**languages**: the optional list of languages in which the different names are provided.

**icon**: an optional **icon** which graphically represents the description associated with the **item_names**.

Formal propositions:

**WR1**: if preferred and short names are provided in more than one language, then all the **languages** attributes of the **translated_label**s must contain the **present_translations** instance as in the languages attribute of this **item_names** instance.

**WR2**: if synonymous names are provided in more than one language, then only languages indicated in the **present_translations** instance in the **languages** attribute of this **item_names** instance can be used.

**WR3**: if no **languages** are provided, **preferred_name**, **short_name** and **synonymous_names** shall not be translated.

NOTE 1 The attributes **preferred_name**, **synonymous_names** and **short_name** are used to encode the "Preferred Name", "Synonymous Name" and "Short Name" attributes respectively for properties and classes (see 7.2 and 8.2).

NOTE 2 The attribute **languages** is used to define the sequence of translations (if requested for attributes).

#### F.3.9.2.8 Label_with_language

The **label_with_language** entity provides resources for associating a label to a language.

EXPRESS specification:

```
*)
ENTITY label_with_language;
      l: label;
      language: language_code;
END_ENTITY; -- label_with_language
(*
```

Attribute definitions:

**l**: the label associated to a language.

**language**: the code of the labeled language.

#### F.3.9.2.9 Mathematical_string

The **mathematical_string** entity provides resources defining a representation for mathematical strings. It also allows a representation in the MathML format.

EXPRESS specification:

```
*)
ENTITY mathematical_string;
      text_representation: text;
      MathML_representation: OPTIONAL text;
END_ENTITY; -- mathematical_string
(*
```

Attribute definitions:

**text_representation**: "linear" form of a mathematical string, using ISO 843, if necessary.

**MathML_representation**: MathML-Text, marked up according to the XML DTD for MathML (document Type Definition). The MathML text must be processed so that it will be treated as one single string during the exchange (see ISO 10303-21).

### F.3.10 Function definitions

This subclause contains functions that are referenced in WHERE clauses to assert data consistency, or that provide resources for application development.

#### F.3.10.1 Acyclic_superclass_relationship function

The **acyclic_superclass_relationship** function checks that there is no cycle in the superclass relationship. By the cardinality of the **its_superclass** attribute in ENTITY class, it is ensured that there is an inheritance

tree, no acyclic graph. Thus, this function merely has to check that no class instance refers in the **its_superclass** attribute to another one that is essentially a subclass.

EXPRESS specification:

```
*)
FUNCTION acyclic_superclass_relationship(
      current: class_BSU; visited: SET OF class): LOGICAL;

IF SIZEOF(current.definition) = 1 THEN
      IF current.definition[1] IN visited THEN
           RETURN(FALSE);
      (* wrong: current declares a subclass as its superclass *)
      ELSE
           IF EXISTS(current.definition[1]\class.its_superclass)
           THEN
                RETURN(acyclic_superclass_relationship(
                     current.definition[1]\class.its_superclass,
                     visited + current.definition[1]));
           ELSE
                RETURN(TRUE);
           END_IF;
      END_IF;
ELSE
      RETURN(UNKNOWN);
END_IF;
END_FUNCTION; -- acyclic_superclass_relationship
(*
```

### F.3.10.2 Check_syn_length function

The **check_syn_length** function checks that the length of **s** does not exceed the length indicated by **s_length**.

EXPRESS specification:

```
*)
FUNCTION check_syn_length(s: syn_name_type; s_length: INTEGER):BOOLEAN;

IF 'ISO13584_IEC61360_DICTIONARY_SCHEMA.LABEL_WITH_LANGUAGE'
      IN TYPEOF(s)
THEN
      RETURN(LENGTH(s.l) <= s_length);
ELSE
      RETURN(LENGTH(s) <= s_length);
END_IF;
END_FUNCTION; -- check_syn_length
(*
```

### F.3.10.3 Codes_are_unique function

The **codes_are_unique** function returns TRUE if the **value_code**s are unique within this list of **values**.

EXPRESS specification:

```
*)
FUNCTION codes_are_unique(values: LIST OF dic_value): BOOLEAN;
LOCAL
      ls: SET OF STRING := [];
      li: SET OF INTEGER := [];
END_LOCAL;

IF('ISO13584_IEC61360_DICTIONARY_SCHEMA.VALUE_CODE_TYPE' IN
      TYPEOF(values[1].value_code))
THEN
      REPEAT i := 1 TO SIZEOF(values);
           ls := ls + values[i].value_code;
      END_REPEAT;

      RETURN(SIZEOF(values) = SIZEOF(ls));
ELSE
      IF('ISO13584_IEC61360_DICTIONARY_SCHEMA.INTEGER_TYPE' IN
           TYPEOF(values[1].value_code))
      THEN
           REPEAT i := 1 TO SIZEOF(values);
                li := li + values[i].value_code;
           END_REPEAT;

           RETURN(SIZEOF(values) = SIZEOF(li));
      ELSE
           RETURN(?);
      END_IF;
END_IF;

END_FUNCTION; -- codes_are_unique
(*
```

### F.3.10.4 Definition_available_implies function

The **definition_available_implies** function checks whether the definition corresponding to the **BSU** parameter exists or not. Then, if this definition exists, the **expression** parameter is returned.

EXPRESS specification:

```
*)
FUNCTION definition_available_implies(
      BSU: basic_semantic_unit;
      expression: LOGICAL): LOGICAL;

RETURN(NOT(SIZEOF(BSU.definition) = 1) OR expression);

END_FUNCTION; -- definition_available_implies
(*
```

### F.3.10.5 Is_subclass function

The function **is_subclass** returns TRUE if **sub** is either **super** or a subclass of **super**. If some class **dictionary_definition** are not available, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION is_subclass(sub, super: class): LOGICAL;
      IF (NOT EXISTS(sub)) OR (NOT EXISTS(super)) THEN
           RETURN(UNKNOWN);
      END_IF;

      IF sub = super
      THEN
           RETURN(TRUE);
      END_IF;

      IF NOT EXISTS(sub.its_superclass)
      THEN
           (* end of chain reached, didn't meet super so far *)
           RETURN(FALSE);
      END_IF;

      IF SIZEOF(sub.its_superclass.definition) = 1
      THEN
      (* definition available *)
           IF (sub.its_superclass.definition[1] = super)
           THEN
                RETURN(TRUE);
           ELSE
                RETURN(is_subclass(sub.its_superclass.definition[1],
                     super));
           END_IF;
      ELSE
           RETURN(UNKNOWN);
      END_IF;

END_FUNCTION; -- is_subclass
(*
```

### F.3.10.6 String_for_derived_unit function

The function **string_for_derived_unit** returns a STRING representation of the **derived_unit** (according to ISO 10303-41) passed as parameter. First, the elements of the derived unit are separated according to the sign of the exponent. If there are elements of both kinds, the '/' notation is used to separate those with positive from those with negative sign. If there are only negative exponents, the u-e notation is used. A dot '.' (decimal code 46 according to ISO/IEC 8859-1 , see ISO 10303-21) is used to separate individual elements.

EXPRESS specification:

```
*)
FUNCTION string_for_derived_unit(u: derived_unit): STRING;

      FUNCTION string_for_derived_unit_element(
           u: derived_unit_element; neg_exp: BOOLEAN
           (* print negative exponents with power -1 *)): STRING;
           (* returns a STRING representation of the
                derived_unit_element (according to ISO 10303-41)
                passed as parameter *)

      LOCAL
           result: STRING;
      END_LOCAL;

      result := string_for_named_unit(u.unit);
      IF (u.exponent <> 0)
      THEN
           IF (u.exponent > 0) OR NOT neg_exp
           THEN
                result := result + '**' + FORMAT(
                     ABS(u.exponent), '2I')[2];
           ELSE
                result := result + '**' + FORMAT(u.exponent, '2I')[2];
           END_IF;
      END_IF;
           RETURN(result);
      END_FUNCTION; -- string_for_derived_unit_element

LOCAL
      pos, neg: SET OF derived_unit_element;
      us: STRING;
END_LOCAL;

(* separate unit elements according to the sign of the exponents: *)
pos := QUERY(ue <* u.elements | ue.exponent > 0);
neg := QUERY(ue <* u.elements | ue.exponent < 0);
us := '';
IF SIZEOF(pos) > 0 THEN
      (* there are unit elements with positive sign *)
      REPEAT i := LOINDEX(pos) TO HIINDEX(pos);
           us := us + string_for_derived_unit_element(pos[i], FALSE);
           IF i <> HIINDEX(pos)
           THEN
                us := us + '.';
           END_IF;
      END_REPEAT;

      IF SIZEOF(neg) > 0
      THEN
           (* there are unit elements with negative sign, use '/'
                notation: *)
           us := us + '/';
```

```
            IF SIZEOF(neg) > 1
            THEN
                 us := us + '(';
            END_IF;

            REPEAT i := LOINDEX(neg) TO HIINDEX(neg);
                 us := us + string_for_derived_unit_element(
                      neg[i], FALSE);
                 IF i <> HIINDEX(neg)
                 THEN
                      us := us + '.';
                 END_IF;
            END_REPEAT;

            IF SIZEOF(neg) > 1
            THEN
                 us := us + ')';
            END_IF;
       END_IF;
  ELSE
       (* only negative signs, use u-e notation *)
       IF SIZEOF(neg) > 0 THEN
            REPEAT i := LOINDEX(neg) TO HIINDEX(neg);
                 us := us + string_for_derived_unit_element(
                      neg[i], TRUE);
                 IF i <> HIINDEX(neg)
                 THEN
                      us := us + '.';
                 END_IF;
            END_REPEAT;
       END_IF;
  END_IF;

  RETURN(us);

  END_FUNCTION; -- string_for_derived_unit
  (*
```

### F.3.10.7 String_for_named_unit function

The **string_for_named_unit** function returns a STRING representation of the **named_unit** (according to ISO 10303-41 and the extension in F.3.8.4.1) passed as parameter.

EXPRESS specification:

```
  *)
  FUNCTION string_for_named_unit(u: named_unit): STRING;

  IF 'MEASURE_SCHEMA.SI_UNIT' IN TYPEOF(u) THEN
       RETURN(string_for_SI_unit(u));
  ELSE
       IF 'MEASURE_SCHEMA.CONTEXT_DEPENDENT_UNIT' IN TYPEOF(u)
```

```
        THEN
             RETURN(u\context_dependent_unit.name);
        ELSE
             IF 'MEASURE_SCHEMA.CONVERSION_BASED_UNIT' IN TYPEOF(u)
             THEN
                  RETURN(u\conversion_based_unit.name);
             ELSE
                  IF 'ISO13584_IEC61360_DICTIONARY_SCHEMA'
                       +'.NON_SI_UNIT' IN TYPEOF(u)
                  THEN
                       RETURN(u\non_si_unit.name);
                  ELSE
                       RETURN('name_unknown');
                  END_IF;
             END_IF;
        END_IF;
   END_IF;

   END_FUNCTION; -- string_for_named_unit
   (*
```

#### F.3.10.8 String_for_SI_unit function

The **string_for_SI_unit** function returns a STRING representation of the **si_unit** (according to ISO 10303-41) passed as parameter.

EXPRESS specification:

```
   *)
   FUNCTION string_for_SI_unit(unit: si_unit): STRING;

   LOCAL
        prefix_string, unit_string: STRING;
   END_LOCAL;

   IF EXISTS(unit.prefix) THEN
        CASE unit.prefix OF
             exa       : prefix_string := 'E';
             peta      : prefix_string := 'P';
             tera      : prefix_string := 'T';
             giga      : prefix_string := 'G';
             mega      : prefix_string := 'M';
             kilo      : prefix_string := 'k';
             hecto     : prefix_string := 'h';
             deca      : prefix_string := 'da';
             deci      : prefix_string := 'd';
             centi     : prefix_string := 'c';
             milli     : prefix_string := 'm';
             micro     : prefix_string := 'u';
             nano      : prefix_string := 'n';
             pico      : prefix_string := 'p';
             femto     : prefix_string := 'f';
             atto      : prefix_string := 'a';
```

```
      END_CASE;
ELSE
      prefix_string := '';
END_IF;

CASE unit.name OF
      metre               : unit_string:= 'm';
      gram                : unit_string := 'g';
      second              : unit_string := 's';
      ampere              : unit_string := 'A';
      kelvin              : unit_string := 'K';
      mole                : unit_string := 'mol';
      candela             : unit_string := 'cd';
      radian              : unit_string := 'rad';
      steradian           : unit_string := 'sr';
      hertz               : unit_string := 'Hz';
      newton              : unit_string := 'N';
      pascal              : unit_string := 'Pa';
      joule               : unit_string := 'J';
      watt                : unit_string := 'W';
      coulomb             : unit_string := 'C';
      volt                : unit_string := 'V';
      farad               : unit_string := 'F';
      ohm                 : unit_string := 'Ohm';
      siemens             : unit_string := 'S';
      weber               : unit_string := 'Wb';
      tesla               : unit_string := 'T';
      henry               : unit_string := 'H';
      degree_Celsius      : unit_string := 'Cel';
      lumen               : unit_string := 'lm';
      lux                 : unit_string := 'lx';
      becquerel           : unit_string := 'Bq';
      gray                : unit_string := 'Gy';
      sievert             : unit_string := 'Sv';
END_CASE;

RETURN(prefix_string + unit_string);

END_FUNCTION; -- string_for_SI_unit
(*
```



### F.3.10.9 String_for_unit function

The **string_for_unit** function returns a STRING representation of the **unit** (according to ISO 10303-41) passed as parameter.

NOTE The **string_for_unit** function is not called from the EXPRESS code. It is a utility function allowing to compute a string representation from the **structured_representation** of a **dic_unit**, for the case where no **string_representation** is present. This string representation corresponds to the one used in Annex B of IEC 61360-1:2009 .

EXPRESS specification:

```
*)
FUNCTION string_for_unit(u: unit): STRING;
      IF 'MEASURE_SCHEMA.DERIVED_UNIT' IN TYPEOF(u)
      THEN
           RETURN(string_for_derived_unit(u));
      ELSE (* 'MEASURE_SCHEMA.NAMED_UNIT' IN TYPEOF(u) holds true *)
           RETURN(string_for_named_unit(u));
      END_IF;
END_FUNCTION; -- string_for_unit
(*
```

### F.3.10.10 All_class_descriptions_reachable function

The **all_class_descriptions_reachable** function checks if the **dictionary_elements** that describe a class, referred by a **class_BSU**, and all its super-classes, can be computed in the inheritance tree defined by the class hierarchy.

EXPRESS specification:

```
*)
FUNCTION all_class_descriptions_reachable(cl: class_BSU): BOOLEAN;

IF NOT EXISTS(cl)
THEN
      RETURN(?);
END_IF;

IF SIZEOF(cl.definition) = 0
THEN
      RETURN(FALSE);
END_IF;

IF NOT(EXISTS(cl.definition[1]\class.its_superclass))
THEN
      RETURN(TRUE);
ELSE
      RETURN(all_class_descriptions_reachable(
           cl.definition[1]\class.its_superclass));
END_IF;

END_FUNCTION; -- all_class_descriptions_reachable
(*
```

### F.3.10.11 Compute_known_visible_properties function

The **compute_known_visible_properties** function computes the set of properties that are visible for a given class. When a definition is not available, it returns only the visible properties that may be computed.

NOTE When some **class dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the properties defined as visible by this super-class cannot be computed by the **compute_known_visible_properties** function. Only on the receiving system all the **dictionary_definition**s of the **BSU**s are required to be available. Therefore, on the receiving

system, the **compute_known_visible_properties** function computes all the properties that are visible to a class by virtue of referencing it (or any of its superclass) by their **name_scope** attribute.

EXPRESS specification:

```
*)
FUNCTION compute_known_visible_properties(cl: class_BSU):
      SET OF property_BSU;
LOCAL
      s: SET OF property_BSU := [];
END_LOCAL;

s := s + USEDIN(cl, 'ISO13584_IEC61360_DICTIONARY_SCHEMA' +
      '.PROPERTY_BSU.NAME_SCOPE');
IF SIZEOF(cl.definition) = 0
THEN
      RETURN(s);
ELSE
      IF EXISTS(cl.definition[1]\class.its_superclass) THEN
           s := s + compute_known_visible_properties(
                cl.definition[1]\class.its_superclass);
      END_IF;

      RETURN(s);
END_IF;

END_FUNCTION; -- compute_known_visible_properties
(*
```

### F.3.10.12 Compute_known_visible_data_types function

The **compute_known_visible_data_types** function computes the set of **data_type**s that are visible for a given class. When a definition is not available, it returns only the visible **data_type**s that may be computed.

NOTE When some **class dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the **data_type**s defined as visible by this super-class cannot be computed by the **compute_known_visible_data_types** function. Only on the receiving system all the **dictionary_definition**s of the **BSU**s are required to be available. Therefore, on the receiving system, the **compute_known_visible_data_types** function computes all the **data_type**s that are visible to a class by virtue of referencing it (or any of its superclass) by their **name_scope** attribute.

EXPRESS specification:

```
*)
FUNCTION compute_known_visible_data_types(cl: class_BSU):
      SET OF data_type_BSU;
LOCAL
      s: SET OF data_type_BSU :=[ ];
END_LOCAL;

s := s + USEDIN(cl, 'ISO13584_IEC61360_DICTIONARY_SCHEMA' +
      '.DATA_TYPE_BSU.NAME_SCOPE');

IF SIZEOF(cl.definition) = 0
```

```
THEN
      RETURN(s);
ELSE
      IF EXISTS(cl.definition[1]\class.its_superclass)
      THEN
           s := s + compute_known_visible_data_types(
                cl.definition[1]\class.its_superclass);
      END_IF;

      RETURN(s);
END_IF;

END_FUNCTION; -- compute_known_visible_data_types
(*
```

### F.3.10.13 Compute_known_applicable_properties function

The **compute_known_applicable_properties** function computes the set of properties that are applicable for a given class. When a definition is not available, it returns only the applicable properties that may be computed.

NOTE When some class **dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the properties defined as applicable by this super-class cannot be computed by the **compute_known_applicable_properties** function. Only on the receiving system all the **dictionary_definition**s of the **BSU**s are required to be available. Therefore, on the receiving system, the **compute_known_applicable_properties** function computes all the properties that are applicable to a class by virtue of being referenced by a **described_by** attribute or of being imported through an **a_priori_semantic_relationship**.

EXPRESS specification:

```
*)
FUNCTION compute_known_applicable_properties(cl: class_BSU):
      SET OF property_BSU;

LOCAL
      s: SET OF property_BSU := [];
END_LOCAL;

IF SIZEOF(cl.definition)=0
THEN
      RETURN(s);
ELSE
      REPEAT i := 1 TO SIZEOF(cl.definition[1]\class.described_by);
           s := s + cl.definition[1]\class.described_by[i];
      END_REPEAT;

      IF (('ISO13584_IEC61360_ITEM_CLASS_CASE_OF_SCHEMA.'
           + 'A_PRIORI_SEMANTIC_RELATIONSHIP')
           IN TYPEOF (cl.definition[1]))
      THEN
           s := s + cl.definition[1]\a_priori_semantic_relationship
            .referenced_properties;
      END_IF;
```

```
      IF EXISTS(cl.definition[1]\class.its_superclass)
      THEN
           s := s + compute_known_applicable_properties(
                cl.definition[1]\class.its_superclass);
      END_IF;

      RETURN(s);
END_IF;
END_FUNCTION; -- compute_known_applicable_properties
(*
```

### F.3.10.14Compute_known_applicable_data_types function

The **compute_known_applicable_data_types** function computes the set of **data_type**s that are applicable for a given class. When a definition is not available, it returns only the applicable **data_type**s that may be computed.

NOTE When some class **dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the **data_type**s defined as applicable by this super-class cannot be computed by the **compute_known_applicable_data_types** function. Only on the receiving system all the **dictionary_definition**s of the **BSU**s are required to be available. Therefore, on the receiving system, the **compute_known_applicable_data_types** function computes all the **data_type**s that are applicable to a class by virtue of being referenced by a **defined_types** attribute or of being imported through an **a_priori_semantic_relationship**.

EXPRESS specification:

```
*)
FUNCTION compute_known_applicable_data_types(cl: class_BSU):
      SET OF data_type_BSU;
LOCAL
      s: SET OF data_type_BSU := [];
END_LOCAL;

IF SIZEOF(cl.definition) = 0
THEN
      RETURN(s);
ELSE
      REPEAT i := 1 TO SIZEOF(cl.definition[1]\class.defined_types);
           s := s + cl.definition[1]\class.defined_types[i];
      END_REPEAT;

      IF (('ISO13584_IEC61360_ITEM_CLASS_CASE_OF_SCHEMA.'
           + 'A_PRIORI_SEMANTIC_RELATIONSHIP')
           IN TYPEOF (cl.definition[1]))
      THEN
           s := s +  cl.definition[1]\a_priori_semantic_relationship
            .referenced_data_types;
      END_IF;

      IF EXISTS(cl.definition[1]\class.its_superclass)
      THEN
           s := s + compute_known_applicable_data_types(
```

```
                cl.definition[1]\class.its_superclass);
        END_IF;

        RETURN(s);
    END_IF;

    END_FUNCTION; -- compute_known_applicable_data_types
    (*
```

### F.3.10.15 List_to_set function

The **list_to_set** function creates a SET from a LIST named **l**, the type of element for the SET will be the same as that in the original LIST.

EXPRESS specification:

```
*)
FUNCTION list_to_set(l: LIST [0:?] OF GENERIC:type_elem):
        SET OF GENERIC: type_elem;

LOCAL
        s: SET OF GENERIC: type_elem := [];
END_LOCAL;

REPEAT i := 1 TO SIZEOF(l);
        s := s + l[i];
END_REPEAT;

RETURN(s);
END_FUNCTION; -- list_to_set
(*
```

### F.3.10.16 Check_properties_applicability function

The **check_properties_applicability** function checks that only those properties that are not applicable for a class by inheritance may become applicable to this class by virtue of being referenced by the **described_by** attribute.

EXPRESS specification:

```
*)
FUNCTION check_properties_applicability(cl: class): LOGICAL;
LOCAL
        inter: SET OF property_bsu := [];
END_LOCAL;

IF EXISTS(cl.its_superclass)
THEN
        IF (SIZEOF(cl.its_superclass.definition)=1)
        THEN
                inter := (list_to_set(cl.described_by) *
                        cl.its_superclass.definition[1]\class.
                        known_applicable_properties);
```

```
            RETURN(inter = []);
      ELSE
            RETURN(UNKNOWN);
      END_IF;
ELSE
      RETURN(TRUE);
END_IF;

END_FUNCTION; -- check_properties_applicability
(*
```

### F.3.10.17 Check_datatypes_applicability function

The **check_datatypes_applicability** function checks that only those datatypes that are not applicable for a class by inheritance may become applicable to this class by virtue of being referenced by the **defined_types** attribute.

EXPRESS specification:

```
*)
FUNCTION check_datatypes_applicability(cl: class): LOGICAL;
LOCAL
      inter: SET OF data_type_bsu := [];
END_LOCAL;

IF EXISTS(cl.its_superclass)
THEN
      IF (SIZEOF(cl.its_superclass.definition) = 1)
      THEN
            inter := cl.defined_types *
                  cl.its_superclass.definition[1]\class.
                  known_applicable_data_types;
            RETURN(inter = []);
      ELSE
            RETURN(UNKNOWN);
      END_IF;
ELSE
      RETURN(TRUE);
END_IF;

END_FUNCTION; -- check_datatypes_applicability

(*
```

### F.3.10.18 One_language_per_translation function

The **one_language_per_translation** function checks that the languages of **translation** in **administrative_data** are unique.

EXPRESS specification:

```
*)
FUNCTION one_language_per_translation (adm: administrative_data)
```

```
                              : LOGICAL;
        LOCAL
             count: INTEGER;
             lang: language_code;
        END_LOCAL;

        REPEAT i := 1 TO SIZEOF (adm.translation);
             lang := adm.translation[i].language;
             count := 0;
             REPEAT j :=1 TO SIZEOF (adm.translation);
                  IF lang = adm.translation[j].language
                  THEN
                       count := count+1;
                  END_IF;
             END_REPEAT;
             IF count >1
             THEN RETURN (FALSE);
             END_IF;
        END_REPEAT;
        RETURN(TRUE);

    END_FUNCTION; -- one_language_per_translation
    (*
```

#### F.3.10.19 Allowed_values_integer_types function

The **allowed_values_integer_types** function computes the set of **integer_type** values allowed by a **non_quantitative_int_type.** If the parameter is indeterminate, it returns the indeterminate value.

EXPRESS specification:

```
    *)
    FUNCTION allowed_values_integer_types (nqit: non_quantitative_int_type)
                              : SET OF integer_type;
    LOCAL
         s : SET OF integer_type :=[];
    END_LOCAL;

    REPEAT i:=1 TO SIZEOF (nqit.domain.its_values);
         s := s + nqit.domain.its_values[i].value_code;
    END_REPEAT;
    RETURN(s);

    END_FUNCTION; -- allowed_values_integer_types
    (*
```

#### F.3.10.20 Is_class_valued_property function

The **is_class_valued_property** function returns TRUE if the property **prop** is defined as a class valued property for class **cl** by means of a **sub_class_properties** attribute in class **cl** or in any of its superclasses. If some class **dictionary_definition**s are not available to compute all the superclasses of **cl**, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION is_class_valued_property(
      prop: property_BSU; cl: class_BSU): LOGICAL;
      IF (SIZEOF(cl.definition) = 0)
      THEN
           RETURN (UNKNOWN);
      ELSE
           IF NOT (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
                +'.ITEM_CLASS') IN TYPEOF(cl.definition[1]))
           THEN
                RETURN (FALSE);
           END_IF;
           IF prop IN cl.definition[1].sub_class_properties
           THEN RETURN (TRUE);
           END_IF;
           IF NOT EXISTS(cl.definition[1].its_superclass)
           THEN
           (* end of chain reached, didn't meet super so far *)
                RETURN(FALSE);
           END_IF;
           RETURN(is_class_valued_property(prop,
                     cl.definition[1].its_superclass));
      END_IF;

END_FUNCTION; -- is_class_valued_property
(*
```

### F.3.10.21 Class_value_assigned function

The **class_value_assigned** function returns the set of values of the property **prop** that have been assigned to a class **cl** by means of a **class_constant_values** attribute in class **cl** or in any superclass of class **cl**. If several values are assigned in several superclasses the function returns the set of all assigned values. If some class **dictionary_definitions** are not available to compute all the superclasses of **cl**, only the values computed are returned.

EXPRESS specification:

```
*)
FUNCTION class_value_assigned(prop: property_BSU;
      cl: class_BSU) : SET OF primitive_value;
      LOCAL
           val:SET OF primitive_value :=[];
           cva : SET OF class_value_assignment :=[];
      END_LOCAL;
      IF (SIZEOF(cl.definition) = 0)
      THEN
           RETURN (val);
      END_IF;
      IF NOT (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
           +'.ITEM_CLASS') IN TYPEOF(cl.definition[1]))
      THEN
```

```
            RETURN (val);
        END_IF;
        IF EXISTS(cl.definition[1])
        THEN
            cva:= QUERY
                (a <* cl.definition[1].class_constant_values
                | a.super_class_defined_property = prop);
            REPEAT i :=1 TO SIZEOF (cva);
                val := val + cva[i].assigned_value;
            END_REPEAT;
            IF NOT EXISTS(cl.definition[1].its_superclass)
            THEN
                RETURN (val);
            ELSE RETURN (val + class_value_assigned
                (prop,cl.definition[1].its_superclass));
            END_IF;
        END_IF;
    END_FUNCTION; -- class_value_assigned


    END_SCHEMA; -- ISO13584_IEC61360_dictionary_schema
    (*
```

## F.4 ISO13584_IEC61360_language_resource_schema

The following schema provides resources for permitting strings in various languages. It has been extracted from the dictionary schema, since it could be used in other schemata. It is largely based on the **support_resource_schema** from ISO 10303-41, "Fundamentals of Product Description and Support", and can be seen as an extension to that. It allows for the usage of one specific language throughout an exchange context (physical file) without the overhead introduced when multiple languages are used. See Figure F.13 - **ISO13584_IEC61360_language_resource_schema** and **support_resource_schema**, for a graphical depiction.

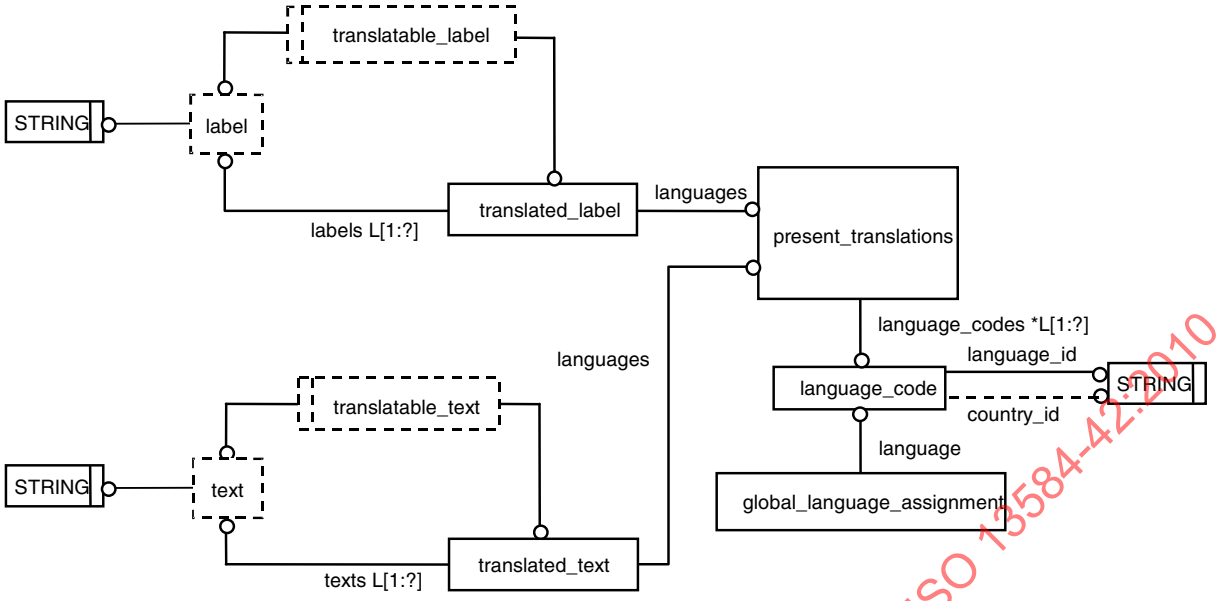



**Figure F.13 — ISO13584_IEC61360_language_resource_schema and support_resource_schema**

EXPRESS specification:

```
*)
SCHEMA ISO13584_IEC61360_language_resource_schema;

REFERENCE FROM support_resource_schema(identifier, label, text);

(*
```

NOTE The **support_resource_schema** schema referenced above can be found in ISO 10303-41.

### F.4.1 ISO13584_IEC61360_language_resource_schema type and entity definitions

This subclause contains the EXPRESS type and entity definitions in the **ISO13584_IEC61360_language_resource_schema**.

#### F.4.1.1 Language_code

The **language_code** entity enables to identify a language according to ISO 639-1 . It contains two codes:

— the language as defined in ISO 639-1 or ISO 639-2 , and, optionally

— the country code, as defined in ISO 3166-1 , specifying in which country the language is spoken.

EXPRESS specification:

```
*)
ENTITY language_code;
      language_id: identifier;
      country_id: OPTIONAL identifier;
WHERE
      WR1: (LENGTH (language_id) = 2) OR (LENGTH (language_id) = 3);
```

```
      WR2: LENGTH (country_id) = 2;
END_ENTITY; -- language_code
(*
```

Attribute definitions:

**language_id**: the code that specifies the language as defined by ISO 639-1 or ISO 639-2 .

**country_id**: the code that specifies the country where the language is spoken as defined by ISO 3166-1 .

Formal propositions:

**WR1**: the length of **language_id** value shall be equal to 2 or 3.

**WR2**: the length of a **country_id** value shall be equal to 2.

#### F.4.1.2 Global_language_assignment

The **global_language_assignment** entity specifies the language for **translatable_label** and **translatable_text**, if **label** and **text** are selected, respectively (i.e., without explicit language indication as is done in **translated_label** and **translated_text)**.

EXPRESS specification:

```
*)
ENTITY global_language_assignment;
      language: language_code;
END_ENTITY; -- global_language_assignment
(*
```

Attribute definitions:

**language**: the code of the assigned language.

#### F.4.1.3 Present_translations

The **present_translations** entity serves to indicate the languages used for **translated_label** and **translated_text**.

EXPRESS specification:

```
*)
ENTITY present_translations;
      language_codes: LIST [1:?] OF UNIQUE language_code;
UNIQUE
      UR1: language_codes;
END_ENTITY; -- present_translations
(*
```

Attribute definitions:

**language_codes**: the list of unique language codes corresponding to the language in which a translation is made.

Formal proposition:

**UR1**: for each list of **language_code** there shall be a unique instance of **present_translations**.

#### F.4.1.4 Translatable_label

A **translatable_label** defines a type of values that can be **label**s or **translated_label**s.

EXPRESS specification:

```
*)
TYPE translatable_label = SELECT(label, translated_label);
END_TYPE; -- translatable_label
(*
```

#### F.4.1.5 Translated_label

The **translated_label** entity defines the labels that are translated and the corresponding languages of translation.

EXPRESS specification:

```
*)
ENTITY translated_label;
      labels: LIST [1:?] OF label;
      languages: present_translations;
WHERE
      WR1: SIZEOF(labels) = SIZEOF(languages.language_codes);
END_ENTITY; -- translated_label
(*
```

Attribute definitions:

**labels**: the list of **label**s that are translated.

**languages**: the list of **languages** in which each label is translated.

Formal propositions:

**WR1**: the number of **label**s contained in the **labels** list shall be equal to the number of languages provided in the **languages.language_codes** attribute.

Informal propositions:

**IP1**: the content of **labels[i]** is in the language identified by **languages.language_codes[i]**.

#### F.4.1.6 Translatable_text

A **translatable_text** defines a type of values that can be **text**s or **translated_text**s.

EXPRESS specification:

```
*)
TYPE translatable_text = SELECT(text, translated_text);
END_TYPE; -- translatable_text
```

```
(*
```

#### F.4.1.7 Translated_text

The **translated_text** entity defines the **text**s that are translated and the corresponding **language**s of translation.

EXPRESS specification:

```
*)
ENTITY translated_text;
      texts: LIST [1:?] OF text;
      languages: present_translations;
WHERE
      WR1: SIZEOF(texts) = SIZEOF(languages.language_codes);
END_ENTITY; -- translated_text
(*
```

Attribute definitions:

**texts**: the list of **text**s that are translated.

**languages**: the list of languages in which each text is translated.

Formal propositions:

**WR1**: the number of **text**s contained in the **texts** list shall be equal to the number of languages provided in the **languages.language_codes** attribute.

Informal propositions:

**IP1**: the content of **texts[i]** is in the language identified by **languages.language_codes[i]**.

### F.4.2 ISO13584_IEC61360_language_resource_schema function definitions

This subclause contains a function that is referenced in WHERE clauses to assert data consistency.

#### F.4.2.1 Check_label_length function

The **check_label_length** function checks that no label in **l** exceeds the length indicated by **l_length**.

EXPRESS specification:

```
*)
FUNCTION check_label_length(l: translatable_label;
      l_length: INTEGER): BOOLEAN;

IF 'ISO13584_IEC61360_LANGUAGE_RESOURCE_SCHEMA.TRANSLATED_LABEL'
      IN TYPEOF(l)
THEN
      REPEAT i :=1 TO SIZEOF(l.labels);
            IF LENGTH(l.labels[i]) > l_length
            THEN
                  RETURN(FALSE);
            END_IF;
```

```
      END_REPEAT;

      RETURN(TRUE);

ELSE (* the argument l is a single string *)
      RETURN(LENGTH(l) <= l_length);
END_IF;
END_FUNCTION; -- check_label_length
(*
```

### F.4.3 ISO13584_IEC61360_language_resource_schema rule definition

The rule **single_language_assignment** asserts that only one language may be assigned to be used in **translatable_label** and **translatable_text**.

EXPRESS specification:

```
*)
RULE single_language_assignment FOR(global_language_assignment);
WHERE
      SIZEOF(global_language_assignment) <= 1;
END_RULE; -- single_language_assignment


END_SCHEMA; -- ISO13584_IEC61360_language_resource_schema
(*
```

## F.5 ISO13584_IEC61360_class_constraint_schema

This clause defines the requirements for the **class**_constraint_schema. The following EXPRESS declaration introduces the **ISO13584_IEC61360_class_constraint_schema** block and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA ISO13584_IEC61360_class_constraint_schema;

REFERENCE FROM ISO13584_IEC61360_dictionary_schema (
      class_BSU,
      property_BSU,
      definition_available_implies,
      is_subclass,
      data_type,
      simple_type,
      complex_type,
      named_type,
      allowed_values_integer_types);

REFERENCE FROM ISO13584_extended_dictionary_schema
      (data_type_typeof,
      data_type_class_of,
```

```
        data_type_type_name);

    REFERENCE FROM ISO13584_instance_resource_schema
        (Boolean_value,
        compatible_class_and_class,
        complex_value,
        dic_class_instance,
        entity_instance_value,
        int_level_spec_value,
        integer_value,
        level_spec_value,
        number_value,
        primitive_value,
        rational_value,
        real_level_spec_value,
        real_value,
        right_values_for_level_spec,
        same_translations,
        simple_value,
        string_value,
        translatable_string_value,
        translated_string_value,
        property_or_data_type_BSU);

    REFERENCE FROM ISO13584_aggregate_value_schema
        (aggregate_entity_instance_value,
        list_value,
        set_value,
        bag_value,
        array_value,
        set_with_subset_constraint_value,
        compatible_complete_types_and_value);
    (*
```

NOTE The schemata referenced above can be found in the following documents:

| | |
|---|---|
| **ISO13584_IEC61360_dictionary_schema** (which is duplicated for convenience in this annex). | IEC 61360-2 |
| **ISO13584_extended_dictionary_schema** | ISO 13584-24:2003 |
| **ISO13584_instance_resource_schema** | ISO 13584-24:2003 |
| **ISO13584_aggregate_value_schema** | ISO 13584-25 |

### F.5.1 Introduction to the ISO13584_IEC61360_class_constraint_schema

The **ISO13584_IEC61360_class_constraint_schema** provides EXPRESS constructs allowing to redefine, by restriction, the domain of values of a given property when it is applied to a subclass of the characterization class where the property was defined as visible. This constraint shall only make explicit a restriction of the domain of values that already results from the class structure.

EXAMPLE In ISO 13584-511 , the class *metric threaded bolt/screw* is a class defined as follows: "headed externally threaded fastener with a cylindrical shank, which may be partly or fully threaded and the head may be furnished with a driving feature". This class has, among other, two properties called *type of head,* and *head properties*. The domain of values of the *type of head* property is a non quantitative data type that includes in particular the following values: *hexagon_head*, *octagonal_head* and *round_head*. The *head properties* property is a feature. It means that it has an *item_class* data type, whose domain is a class *head* that defines any kind of head. The *head* class has several subclasses including: *hexagon head*, associated with all the properties allowing to describe a hexagon head (e. g., *width across flats)*, and *round head*, associated with all the properties allowing to describe a round head (e. g., *head diameter*).
The class *metric threaded bolt/screw* has a subclass called *hexagon head screw* defined as follows: "metric externally threaded fastener with a hexagon head threaded up to the head". This class inherits the properties *type of head* and *head*.

From the definition of the *hexagon head screw* subclass, it is clear that the *type of head* property could only take the *hexagon_head* value, and that the *head properties* could only be an instance of *hexagon head* feature class. But these constraints are implicit: they are just stated informally in the definition. Thus these constraints are not computer sensible. The constraints defined in the **ISO13584_IEC61360_class_constraint_schema** would allow to make these two constraints explicit by associating with the *hexagon head screw* class: (1) an **enumeration_constraint** for the *type of head* property (allowing only the *hexagon_head* code) and (2) a **subclass_constraint** *for the head properties* property (allowing only the *hexagon head* feature class).

Constraints are inherited. When a property whose domain of values has already been restricted in a class C through a constraint needs to be further restricted in a subclass of C through another constraint; both constraints apply together. Thus the real domain of values in the C subclass is the intersection of the two domains defined by the two constraints. The proposed mechanism is similar to the type mechanism redefinition operation available in the EXPRESS language.

This schema allows to express constraints that may apply to data types from the type system of the **ISO13584_IEC61360_dictionary_schema**. Rules are used in those entities that reference a constraint to ensure that each constraint may apply to the data type to which it is related.

### F.5.2 ISO13584_IEC61360_class_constraint_schema entity definitions

This clause defines the entities in the **ISO13584_IEC61360_class_constraint_schema**.

#### F.5.2.1 Constraint

The **constraint** entity allows to define a constraint.

EXPRESS specification:

```
*)
ENTITY constraint
ABSTRACT SUPERTYPE OF ( ONEOF (
                    property_constraint,
                    class_constraint));
     constraint_id: OPTIONAL constraint_identifier;
END_ENTITY; -- constraint
(*
```

Attribute definitions:

**constraint_id**: the **constraint_identifier** that identifies the constraint.

#### F.5.2.2 Property_constraint

The **property_constraint** entity is a constraint that restricts the allowed set of instances of a class by a single restriction of the domain of values of one of its properties.

EXPRESS specification:

```
*)
ENTITY property_constraint
ABSTRACT SUPERTYPE OF ( ONEOF (
                    integrity_constraint,
                    context_restriction_constraint))
SUBTYPE OF (constraint);
     constrained_property: property_BSU;
END_ENTITY; -- property_constraint
```

```
(*
```

Attribute definitions:

**constrained_property**: the **property_BSU** for which the constraint applies.

### F.5.2.3 Class_constraint

The **class_constraint** entity is a constraint that restricts the allowed set of instances of a class by constraining several properties or global constraints.

EXPRESS specification:

```
*)
ENTITY class_constraint
ABSTRACT SUPERTYPE OF (configuration_control_constraint)
SUBTYPE OF (constraint);
END_ENTITY; -- class_constraint
(*
```

### F.5.2.4 Configuration_control_constraint

The **configuration_control_constraint** entity allows to restrict the set of instances, called the referenced instances, that a particular instance, called the referencing instance, may reference directly or indirectly by means of a chain of properties. The referencing instance is any instance of a class that references the **configuration_control_constraint** by means of its **constraints** attribute or inherits of a class that does so. The **configuration_control_constraint** entity defines an optional **precondition** that specifies the condition on the referencing instance for the restriction to apply. It defines a **postcondition** that specifies the allowed sets of values for some properties of the referenced instance class.

EXAMPLE A *bolted assembly* consists of the following set of fasteners: one e*xternally threaded fastener*, any number of *washers* and one or more *nuts*. There exist various kinds of threads, including *tapping screw thread, wood screw thread, metric external thread, metric internal thread, imperial internal thread and imperial external thread*. Let's assume that one wants to describe a *metric bolted assembly*. One needs to ensure that whatever be the precise structure of the assembly, both the e*xternally threaded fastener* and all the *nut*s involved in the assembly have metric thread. This can be done by specifying in the *metric bolted assembly* class the **configuration_control_constraint** that ensures that any ISO 13584-511 -described fastener referenced by any instance of this class, or any of its subclass, shall belong to classes that either do not have value for the *type of thread* property (e. g., *washer*), or whose values shall belong to the set: {*metric external thread, metric internal thread*}.

NOTE 1 Both **precondition** and **postcondition** may only restrict properties whose data type is **non_quantitative_code_type**. Such properties may be assigned a value either at the instance level, or at the class level if they are declared as class valued properties, i.e., **sub_class_properties** in a **class**.

NOTE 2 Properties referenced in the **precondition** must be applicable to the class that references the **configuration_control_constraint**.

NOTE 3 In a **configuration_control_constraint**, **filter**s are used both to represent precondition on the referencing instance and to express constraints on the referenced instances.

EXPRESS specification:

```
*)
ENTITY configuration_control_constraint
SUBTYPE OF (class_constraint);
      precondition: SET [0:?] OF filter;
      postcondition: SET [1:?] OF filter;
END_ENTITY; -- configuration_control_constraint
```

```
(*
```

Attribute definitions:

**precondition**: the **filter**s that must hold on the referencing instance for the restriction to apply.

NOTE 4 If the set of filters is empty, the restriction applies on any referencing instance.

**postcondition**: the **filter**s that must hold on a referenced instance for being allowed for reference.

### F.5.2.5 Filter

The **filter** entity is an **enumeration_constraint** that restricts the allowed domain of a property whose data type is either **non_quantitative_code_type** or **non_quantitative_int_type**.

EXPRESS specification:

```
*)
ENTITY filter;
      referenced_property: property_BSU;
      domain: enumeration_constraint;
WHERE
      WR1: definition_available_implies (
           referenced_property,
           (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
           +'.NON_QUANTITATIVE_CODE_TYPE') IN TYPEOF(
           referenced_property.
           definition[1]\property_DET.domain))
           OR
           (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
           +'.NON_QUANTITATIVE_INT_TYPE') IN TYPEOF(
           referenced_property.
           definition[1]\property_DET.domain)));
      WR2: definition_available_implies (
           referenced_property,
           correct_constraint_type(domain,
           referenced_property.definition[1].domain));
END_ENTITY; -- filter
(*
```

Attribute definitions:

**referenced_property**: the property whose domain of values is restricted by the **filter**.

**domain**: the **enumeration_constraint** that restricts the domain of values of the referenced property.

Formal propositions:

**WR1**: the data type of the **referenced_property** shall be either **non_quantitative_code_type** or **non_quantitative_int_type**.

**WR2**: the **domain** shall define a domain of values that may restrict the initial domain of values of the property.

### F.5.2.6 Integrity_constraint

The **integrity_constraint** entity is a particular property constraint that allows to make explicit that for some particular class, as a result of the class definition, and all its subclasses, only a restriction of the domain of values specified by a data type is allowed for a property.

EXAMPLE In the reference dictionary defined for fasteners in ISO 13584-511 , a *metric threaded bolt/screw* has a *head properties* property that may take, as value, a member of any subclass of the *head* feature class. If the *metric threaded bolt/screw* is also a member of the *hexagon head screw* subclass, the *head properties* may only be a member of the *hexagon head* feature class, else the *metric threaded bolt/screw* cannot be a member of the *hexagon head screw* subclass.

NOTE In the example above, the integrity constraint does not change at all the meaning of the *head properties* property inherited from *metric threaded bolt/screw* into *hexagon head screw*. It just makes explicit the fact that in the context of the *hexagon head screw* subclass, only a subset of the values allowed for this property in the context of the *metric threaded bolt/screw* class remains allowed.

EXPRESS specification:

```
*)
ENTITY integrity_constraint
SUBTYPE OF (property_constraint);
      redefined_domain: domain_constraint;
WHERE
      WR1: definition_available_implies (constrained_property,
           correct_constraint_type(redefined_domain,
           constrained_property.definition[1].domain));
END_ENTITY; -- integrity_constraint
(*
```

Attribute definitions:

**redefined_domain**: the constraint that applies on the domain of values of the constrained property.

Formal propositions:

**WR1**: the **redefined_domain** shall define a domain of values that restricts the initial domain of values of the property.

### F.5.2.7 Context_restriction_constraint

The **context_restriction_constraint** entity is a **property_constraint** that restricts the allowed domain of values for the context parameters on which a context dependent property depends.

EXPRESS specification:

```
*)
ENTITY context_restriction_constraint
SUBTYPE OF (property_constraint);
      context_parameter_constraints: SET [1:?] OF property_constraint;
WHERE
      WR1: definition_available_implies(constrained_property,
           QUERY (cp <*SELF.context_parameter_constraints
           | NOT (cp.constrained_property IN
           constrained_property.definition[1].depends_on))=[]);
      WR2: QUERY (cp <*SELF.context_parameter_constraints
```

```
          | NOT (('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
          +'.INTEGRITY_CONSTRAINT') IN TYPEOF (cp))) =[];
     WR3:definition_available_implies(constrained_property,
          'ISO13584_IEC61360_DICTIONARY_SCHEMA.DEPENDENT_P_DET'
          IN TYPEOF(constrained_property.definition[1]));
END_ENTITY; -- context_restriction_constraint
(*
```

Attribute definitions:

**context_parameter_constraints:** the constraint that applies on the domain of values of the context parameters.

Formal propositions:

**WR1**: the set of properties whose domain is constrained by the **context_parameter_constraints** property shall be context parameters on which the constrained property depends.

**WR2**: all the c**ontext_parameter_constraints** shall be **integrity_constraint**s.

**WR3**: the **constrained_property** shall be a **context dependent property dependent_P_DET**.

#### F.5.2.8 Domain_constraint

A **domain_constraint** defines a constraint that restricts the domain of values of a data type.

EXPRESS specification:

```
*)
ENTITY domain_constraint
ABSTRACT SUPERTYPE OF(ONEOF(
     subclass_constraint,
     entity_subtype_constraint,
     enumeration_constraint,
     range_constraint,
     string_size_constraint,
     string_pattern_constraint,
     cardinality_constraint
     ));
END_ENTITY; -- domain_constraint
(*
```

#### F.5.2.9 Subclass_constraint

A **subclass_constraint** restricts the domain of values of a **class_reference_type** to one or several subclasses of the class that defines its initial domain.

EXPRESS specification:

```
*)
ENTITY subclass_constraint
SUBTYPE OF(domain_constraint);
     subclasses: SET [1:?] OF class_BSU;
END_ENTITY; -- subclass_constraint
(*
```

Attribute definitions:

**subclasses**: the **class_BSU**s which redefine the class to which the value of the **constrained_property** shall belong.

### F.5.2.10 Entity_subtype_constraint

An **entity_subtype_constraint** restricts the domain of values of an **entity_instance_type** to a subtype of the ENTITY that defines its initial domain.

EXPRESS specification:

```
*)
ENTITY entity_subtype_constraint
SUBTYPE OF(domain_constraint);
     subtype_names: SET[1:?] OF STRING;
END_ENTITY; -- entity_subtype_constraint
(*
```

Attribute definitions:

**subtype_names**: the set of strings that describe, in the format of the EXPRESS TYPEOF function, the EXPRESS entity data type names that shall belong to the result of the EXPRESS TYPEOF function when it is applied to a value that references the **constrained_property** redefined property.

### F.5.2.11 Enumeration_constraint

An **enumeration_constraint** restricts the domain of values of a data type to a list of values defined in extension. The order defined by the list is the recommended order for presentation purposes. A particular description may optionally be associated with each value of the subset by means of a **non_quantitative_int_type**, of which the i-the value describes the meaning of the i-the value of the subset.

For those subtypes of **number type** that are associated with a **dic_unit** and alternative units, and possibly with a **dic_unit_identifier** and alternative unit identifiers, the constraint applies to the value corresponding to the **dic_unit**, or to the single **dic_unit_identifier**. If both exist, they correspond to the same unit.

For those subtypes of **number type** that are associated with a currency, the constraint applies to the currency specified in their data type definition. If no currency is specified in the data type definition, the constraint shall not be used.

For those values that belong to **translatable_string_type**s, the constraint applies to the string that is in the source language into which the property domain was defined. This source language may be defined in the **source_language** attribute of the **administrative_data** of the property. If this attribute does not exist, this source language is supposed to be known by the dictionary user.

If another **enumeration_constraint** is applied on a property already associated with an **enumeration_constraint** in some superclass, both constraints apply. Thus the allowed set of values is the intersection of both subsets. Concerning the presentation order, and the possible meaning associated with each value, only those meanings defined in the lower **enumeration_constraint** apply.

EXAMPLE 1 , and if, in class C1, this property is associated with an **enumeration_constraint** whose **subset** attribute equals {1, 3, 5, 7}, then in class C1, and any of its subclasses, property P1 may only takes one of the four following values: 1 or 3 or 5 or 7.

EXAMPLE 2 If the data type of property P1 is LIST [1:4] OF INTEGER, and if in class C1 this property is associated with an **enumeration_constraint** whose **subset** attribute equals { {1} , {3, 5}, {7}, {1, 3, 7} } then, in class C1 and any of its subclasses, property P1 may only takes one of the four following values: {1} or {3, 5} or {7} or {1, 3, 7}.

EXPRESS specification:

```
*)
ENTITY enumeration_constraint
SUBTYPE OF (domain_constraint);
      subset: LIST [1:?] OF UNIQUE primitive_value;
      value_meaning: OPTIONAL non_quantitative_int_type;
WHERE
      WR1: (NOT(EXISTS(SELF.value_meaning)))
           OR
           (integer_values_in_range(1, SIZEOF(SELF.subset))
                = allowed_values_integer_types(SELF.value_meaning));
END_ENTITY; -- enumeration_constraint
(*
```

Attribute definitions:

**subset**: the list describing the subset of values that are allowed as possible values for the property identified by **constrained_property**.

**value_meaning**: the set of **dic_value**s that define the meaning of each value belonging to the **subset**.

Formal propositions:

**WR1**: if the **value_meaning non_quantitative_int_type** exists, then the set of **value_code**s of its **dic_value**s shall be in 1 .. SIZE_OF(subset).

### F.5.2.12 Range_constraint

A **range_constraint** entity restricts the domain of values of an ordered type to a subset of values defined by a range.

NOTE 1 Strings are not considered as ordered types and cannot be constrained by a **range_constraint**.

For those subtypes of **number_type** that are associated with a **dic_unit** and alternative units, and possibly with a **dic_unit_identifier** and alternative unit identifiers, the constraint applies to the value corresponding to the **dic_unit**, or to the single **dic_unit_identifier**. If both exists, they correspond to the same unit.

For those subtypes of **number_type** that are associated with a currency, the constraint applies to the currency specified in their data type definition. If no currency is specified in the data type definition, the constraint shall not be used.

NOTE 2 For **non_quantitative_int_type** the constraint applies to the **value_code**.

EXPRESS specification:

```
*)
ENTITY range_constraint
SUBTYPE OF (domain_constraint);
      min_value, max_value: OPTIONAL NUMBER;
      min_inclusive, max_inclusive: OPTIONAL BOOLEAN;
WHERE
      WR1: min_value <= max_value;
      WR2: TYPEOF(min_value) = TYPEOF(max_value);
      WR3: NOT EXISTS (min_value) OR EXISTS (min_inclusive);
```

```
      WR4: NOT EXISTS (max_value) OR EXISTS (max_inclusive);
END_ENTITY; -- range_constraint
(*
```

Attribute definitions:

**min_value**: the number defining the low bound of the range of values; not existing value means no lower bound.

**max_value**: the number defining the high bound of the range of values; not existing value means no upper bound.

**min_inclusive**: specifies whether **min_value** belongs to the range; not existing value means that there is no low bound.

**max_inclusive**: specifies whether **max_value** belongs to the range; not existing value means hat there is no high bound.

Formal propositions:

**WR1**: **min_value** shall be less than or equal to **max_value**.

**WR2**: **min_value** and **max_value** shall have the same data types.

**WR3**: if **min_value** has a value, then **min_inclusive** shall also have a value.

**WR4**: if **max_value** has a value, then **max_inclusive** shall also have a value.

#### F.5.2.13 String_size_constraint

A **string_size_constraint** restricts the length of the STRING values allowed for a STRING type, or any of its subtypes.

NOTE 1 A **string_type** property value domain is either a **string_type**, a **non_translatable_string_type**, a **translatable_string_type**, a **URI_type**, a **non_quantitative_code_type**, a **date_data_type**, a **time_data_type** or a **date_time_data_type**.

NOTE 2 For **non_quantitative_code_type** the constraint applies to the code.

For those values that belong to **translatable_string_type**s, the constraint applies to the string that is in the source language into which the property domain was defined. This source language may be defined in the **source_language** attribute of the **administrative_data** of the property. If this attribute does not exist, this source language is supposed to be known by the dictionary user.

EXPRESS specification:

```
*)
ENTITY string_size_constraint
SUBTYPE OF (domain_constraint);
      min_length: OPTIONAL INTEGER;
      max_length: OPTIONAL INTEGER;
WHERE
      WR1: (min_length >= 0) AND (max_length >= min_length);
END_ENTITY; -- string_size_constraint
(*
```

Attribute definitions:

**min_length**: the minimal length for the strings that are allowed as values for the property identified by the **constrained_property** property.

**max_length**: the maximal lengths for the strings that are allowed as values for the property identified by the **constrained_property** property.

NOTE 3 If the **min_length** value does not exist, 0 is to be understood. If the **max_length** value does not exist, unbounded is to be understood.

Formal propositions:

**WR1**: **min_length** and **max_length** define correct bounds.

### F.5.2.14 String_pattern_constraint

A **string_pattern_constraint** restricts the domain of values of a **string_type**, or of any of its subtypes, to string values that match a particular pattern. The **pattern** syntax is defined by an XML regular expression and the associated matching algorithms that are defined by the XML Schema Part 2: Datatypes recommendation.

NOTE 2 A **string_type** property value domain is either a **string_type**, a **non_translatable_string_type**, a **translatable_string_type**, an **URI_type**, a **non_quantitative_code_type**, a **date_data_type**, a **time_data_type** or a **date_time_data_type**.

For **string_type**, **non_translatable_string_type**, **URI_type**, **non_quantitative_code_type**, **date_data_type**, **time_data_type** or **date_time_data_type** the constraint applies to the (unique) string that is the value of the data type. For **non_quantitative_code_type** the constraint applies to the code.

For those values that belong to **translatable_string_types**, the constraint applies to the string that is in the source language into which the property domain was defined. This source language may be defined in the **source_language** attribute of the **administrative_data** of the property, if this attribute does not exist, this source language is supposed to be known by the dictionary user.

NOTE 3 For **non_quantitative_code_type**, **date_data_type**, **time_data_type** or **date_time_data_type**, the **pattern** shall comply with the informal propositions defined in the corresponding data types.

EXPRESS specification:

```
*)
ENTITY string_pattern_constraint
SUBTYPE OF (domain_constraint);
     pattern: STRING;
END_ENTITY; -- string_pattern_constraint
(*
```

Attribute definition:

**pattern**: the pattern of string values that are allowed as values for the property identified by the constrained property.

Informal proposition:

**IP1**: the **pattern** syntax shall complies with the XML regular expression syntax and the associated matching algorithms that are defined by the XML Schema Part 2: Datatypes recommendation.

EXAMPLE The XML Schema pattern that corresponds to the “[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]” SQL SIMILAR expression is “[0-9]{4}\-[0-9]{2}\-[0-9]{2}”. It allows to match strings as “2009-05-31”.

#### F.5.2.15 Cardinality_constraint

A **cardinality_constraint** restricts the cardinality of an aggregate data type.

NOTE 1 The resulting cardinality range is the intersection of preexisting cardinality ranges and of the one defined by the **cardinality_constraint**.

NOTE 2 **Cardinality_constraint**s are not allowed on **array_type**.

EXPRESS specification:

```
*)
ENTITY cardinality_constraint
SUBTYPE OF (domain_constraint);
      bound_1: OPTIONAL INTEGER;
      bound_2: OPTIONAL INTEGER;
WHERE
      WR1: (bound_1 >= 0) AND (bound_2 >= bound_1);
END_ENTITY; -- cardinality_constraint
(*
```

Attribute definitions:

**bound_1**: the lower bound of the cardinality.

**bound_2**: the upper bound of the cardinality.

NOTE 3 When **bound_1** does not exist, the minimal cardinality is 0. When **bound_2** does not exist, there is no constraint on the maximal cardinality.

Formal propositions:

**WR1**: **bound_1** and **bound_2** define correct bounds.

### F.5.3 ISO13584_IEC61360_class_constraint_schema type definitions

This subclause defines the type in the **ISO13584_IEC61360_class_constraint_schema.**

#### F.5.3.1 Constraint_or_constraint_id

The **constraint_or_constraint_id** is either a **constraint** or a **constraint_identifier**.

EXPRESS specification:

```
*)
TYPE constraint_or_constraint_id =
      SELECT (constraint, constraint_identifier);
END_TYPE; -- constraint_or_constraint_id
(*
```

### F.5.4 ISO13584_IEC61360_class_constraint_schema function definition

This subclause defines the functions in the **ISO13584_IEC61360_class_constraint_schema**.

### F.5.4.1 Integer_values_in_range function

The **integer_values_in_range** function computes the integer values that belong to an integer range defined by its low bound and its high bound. It returns indeterminate (?) when either bounds are indeterminate.

EXPRESS specification:

```
*)
FUNCTION integer_values_in_range(
      low_bound, high_bound: INTEGER): SET OF INTEGER;
LOCAL
      i: INTEGER;
      result: SET OF INTEGER:= [];
END_LOCAL;
      IF EXISTS (low_bound) AND EXISTS (high_bound)
      THEN
           REPEAT i := low_bound TO high_bound;
                result := result + [i];
           END_REPEAT;
           RETURN(result);
      ELSE
           RETURN(?);
      END_IF;
END_FUNCTION; -- integer_values_in_range
(*
```

### F.5.4.2 Correct_precondition function

The **correct_precondition** function checks that the precondition of the **configuration_control_constraint** defined by **cons** uses only properties that are applicable to the **cl** class. It returns a logical that is UNKNOWN when the complete set of applicable properties of the class cannot be computed.

EXPRESS specification:

```
*)
FUNCTION correct_precondition(
      cons: configuration_control_constraint; cl:class): LOGICAL;
LOCAL
      prop: SET OF property_BSU:= [];
END_LOCAL;
      REPEAT i := 1 to SIZEOF (cons.precondition);
           prop := prop + cons.precondition[i].referenced_property;
      END_REPEAT;

      IF prop <= cl.known_applicable_properties
      THEN RETURN (TRUE);
      ELSE
           IF all_class_descriptions_reachable(cl.identified_by)
           THEN RETURN (FALSE);
           ELSE RETURN (UNKNOWN);
           END_IF;
      END_IF;
END_FUNCTION; -- correct_precondition
(*
```

### F.5.4.3 Correct_constraint_type function

The **correct_constraint_type** function checks that the **domain_constraint** defined by **cons** is compatible with the **data_type** defined by **typ**. It returns a logical that is UNKNOWN when the **domain_constraint** defined by **cons** is not one of the subtypes defined in the **ISO13584_IEC61360_class_constraint_schema**.

EXPRESS specification:

```
*)
FUNCTION correct_constraint_type(
     cons: domain_constraint; typ:data_type): LOGICAL;

(*case subclass constraint*)

IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
     +'SUBCLASS_CONSTRAINT') IN TYPEOF(cons)
THEN
     (*the data type shall be class_reference_type*)
     IF NOT
          ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_REFERENCE_TYPE'
          IN TYPEOF (typ))
     THEN RETURN(FALSE);
     END_IF;

     (*the cons.subclasses shall consist of subclasses for the class
     that defined the initial domain of typ.*)
     IF NOT (QUERY (sc <* cons.subclasses |
               definition_available_implies
          (sc,definition_available_implies
          (typ\class_reference_type.domain,is_subclass(sc.definition[1]
          , typ\class_reference_type.domain.definition[1])))= false)
          = [])
     THEN RETURN(FALSE);
     END_IF;

     RETURN (TRUE);
END_IF;
(*case entity subtype constraint*)

IF (('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
     +'ENTITY_SUBTYPE_CONSTRAINT') IN TYPEOF (CONS))
THEN

(* the data type is a class_reference_type*)
     IF NOT (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
     +'.ENTITY_INSTANCE_TYPE') IN TYPEOF (typ))
     THEN RETURN(FALSE);
     END_IF;
(* the subtype_name shall define a subtype for the entity_instance_type of
the constrained *)
```

```
      IF NOT (cons\entity_subtype_constraint.subtype_names
           >= typ\entity_instance_type.type_name)
      THEN RETURN(FALSE);
      END_IF;

RETURN (TRUE);
END_IF;

(*case enumeration_constraint *)

IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
     +'.ENUMERATION_CONSTRAINT') IN TYPEOF (CONS)
THEN

(* all the values belonging to the subset of values shall be compatible with
the typ data type *)
      IF (QUERY (val<*cons.subset |
           NOT compatible_data_type_and_value ( typ, val))<> [])
      THEN RETURN(FALSE);
      END_IF;

RETURN (TRUE);
END_IF;


(*case range_constraint *)

IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA.RANGE_CONSTRAINT'
       IN TYPEOF (CONS))
THEN

(*if the data type is an integer_type then min_value and max_value shall be
INTEGERs.*)
      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.INTEGER_TYPE'
           IN TYPEOF (typ)) AND
           NOT ('INTEGER' IN TYPEOF (cons.min_value))
      THEN RETURN(FALSE);
      END_IF;

(*if the data type is a rational_type then min_value and max_value shall be
rational.*)
      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.RATIONAL_TYPE'
           IN TYPEOF (typ)) AND
           NOT  ('ISO13584_INSTANCE_RESOURCE_SCHEMA.RATIONAL_VALUE'  IN  TYPEOF
(cons.min_value))
      THEN RETURN(FALSE);
      END_IF;

(*if  the  data  type  is  a  real_type  then  min_value  and  max_value  shall  be
REALs.*)
      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE'
           IN TYPEOF (typ)) AND NOT ('REAL' IN TYPEOF (cons.min_value))
      THEN RETURN(FALSE);
      END_IF;
```

```
(*all values of the range shall belong to the allowed values defined by the
type.*)
      IF (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
           + '.NON_QUANTITATIVE_INT_TYPE') IN TYPEOF (typ))
      AND NOT
           (integer_values_in_range(cons.min_value, cons.max_value)
           <= allowed_values_integer_types (typ))
      THEN RETURN(FALSE);
      END_IF;

RETURN (TRUE);
END_IF;

(*case entity string_size_constraint*)

IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
      +'.STRING_SIZE_CONSTRAINT') IN TYPEOF (CONS)
THEN

(* the data type shall be a string_type or any of its subtypes *)
IF NOT     ('ISO13584_IEC61360_DICTIONARY_SCHEMA.STRING_TYPE'
           IN TYPEOF (typ))
THEN RETURN(FALSE);
END_IF;

RETURN (TRUE);
END_IF;

(*case entity string_pattern_constraint *)

IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
      +'.STRING_PATTERN_CONSTRAINT') IN TYPEOF (CONS)
THEN

(* the data type shall be a string_type or any of its subtypes *)
      IF NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA.STRING_TYPE'
           IN TYPEOF (typ))
      THEN RETURN(FALSE);
      END_IF;
RETURN (TRUE);
END_IF;

(*case entity cardinality_constraint *)

IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
      +'.CARDINALITY_CONSTRAINT') IN TYPEOF (CONS)
THEN

(* the data type shall be an aggregate type but not an array*)
      IF (NOT(
      ('ISO13584_IEC61360_DICTIONARY_AGGREGATE_EXTENSION_SCHEMA'
      + '.ENTITY_INSTANCE_TYPE_FOR_AGGREGATE')
```

```
        IN TYPEOF(typ)))
      THEN
           RETURN(FALSE);
      END_IF;

      IF ('ISO13584_IEC61360_DICTIONARY_AGGREGATE_EXTENSION_SCHEMA'
           + '.ARRAY_TYPE' IN TYPEOF(typ.type_structure))
      THEN
           RETURN(FALSE);
      END_IF;
      RETURN (TRUE);
END_IF;

RETURN (UNKNOWN);

END_FUNCTION; -- correct_constraint_type
(*
```

#### F.5.4.4 Compatible_data_type_and_value function

The function **compatible_data_type_and_value** checks if a value **val** of a **primitive_value** is type compatible with the type defined by a type **dom**. It returns a LOGICAL that is TRUE when they are compatible and FALSE when they are not. This function returns UNKNOWN if the **val** data type is an **uncontrolled_instance_value** (see ISO 13584-24:2003) or when its type is not one of the types defined in the **ISO13584_instance_resource_schema**.

NOTE The value **val** may or may not exist.

EXPRESS specification:

```
*)
FUNCTION compatible_data_type_and_value(dom: data_type;
      val: primitive_value): LOGICAL;

LOCAL
      temp: class_BSU;
      set_string: SET OF STRING := [];
      set_integer: SET OF INTEGER := [];
      code_type: non_quantitative_code_type;
      int_type: non_quantitative_int_type;
END_LOCAL;

(* The following express statements deal with simple types *)

IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.INTEGER_VALUE' IN TYPEOF(val))
THEN
      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.' +
           'NON_QUANTITATIVE_INT_TYPE' IN TYPEOF (dom))
      THEN
           set_integer := [];
           int_type := dom;
           REPEAT j := 1 TO SIZEOF(int_type.domain.its_values);
                set_integer := set_integer +
                     int_type.domain.its_values[j].value_code;
           END_REPEAT;
```