INTERNATIONAL STANDARD

ISO 13584-24

First edition
2003-11-01

Industrial automation systems and integration — Parts library —

Part 24: Logical resource: Logical model of supplier library

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

Partie 24: Ressource logique: Modèle logique de fournisseur

Reference number
ISO 13584-24:2003(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 13584-24 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 4, *Industrial data*.

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 42: Description methodology: Methodology for structuring part families*

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

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

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

— Parts 10 to 19 specifiy the conceptual descriptions;

— Parts 20 to 29 specify the logical resources;

— Parts 30 to 39 specifiy the implentation resources;

— Parts 40 to 49 specifiy the description methodology;

— Parts 100 to 199 specifiy the view exchange protocol.

Should further parts of ISO 13584 be published, they will follow the same numbering pattern.

Introduction

ISO 13584 is an International Standard 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.

This International Standard 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 and view exchange protocol. The series are described in ISO 13584-1. This part of ISO 13584 is a member of the logical resources series.

This part of ISO 13584 specifies the generic resources needed for supplier library modelling and exchange. It also provides the EXPRESS integrated information models that permit the exchange of libraries that consist either of definitions of families of parts, representations of families of parts, or definitions of new representation categories that may be provided for any family of parts. Knowledge of EXPRESS as defined in ISO 10303-11 is required to understand this part of ISO 13584. Basic knowledge of ISO 13584-20 and ISO 13584-42 is also required.

Industrial automation systems and integration — Parts library —

Part 24: Logical resource: Logical model of supplier library

1 Scope

This part of ISO 13584 specifies generic EXPRESS resource constructs that support the description of different kinds of information about supplier libraries. It also contains a set of integrated EXPRESS information models for representing supplier libraries for the purpose of exchange. These integrated information models integrate EXPRESS resource constructs from different parts of ISO 13584 and ISO 10303 into a single schema. Supplier libraries may consist of definitions or representations of families of parts. Supplier libraries may also define new representation categories. Supplier libraries may consist only of dictionary elements, or they may also contain specifications of permitted instances.

When used together with view exchange protocols, these integrated information models also permit the exchange of one or several representation categories for the parts defined in a parts library.

NOTE 1 View exchange protocols are defined in the view exchange protocol series of ISO 13584.

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

— Generic resource constructs for representing hierarchies of families of parts. The parts in the families may be components or assembled parts, and may be abstract parts or physical parts.

— Generic resource constructs for representing implicitly the definitions of the different parts that belong to a family of parts.

— Generic resource constructs for representing the different kinds of possible representations of the different parts that belong to a family of parts.

— Generic resource constructs for representing families of materials, together with their definitions and possible representations.

— Library integrated information models that gather generic resource constructs from different parts of ISO 13584 and ISO 10303 into one single schema for representing supplier libraries for the purpose of exchange. The supplier libraries may consist either of definitions of families of parts, or of representations of families of parts or of definitions of new representation categories that may be provided for any family of parts.

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

— Description of assembled parts that may contain an unlimited number of constituent components.

— Specification of a software system able to manage supplier libraries represented according to the information models defined in this part of ISO 13584.

— Description of the different representation categories that a supplier library may contain.

NOTE 2 The description of the different representation categories that a supplier library may contain are defined in the view exchange protocol series of parts of ISO 13584.

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 3166:1993, *Codes for the representation of names of countries*

ISO 6093:1985, *Information processing — Representation of numerical values in character strings for information interchange*

ISO/IEC 8824-1:2003, *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation*

ISO 8859-1:1987, *Information processing — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1*

ISO 8879:1986, *Information processing — Text and office systems — Standard Generalized Markup Language (SGML)*

ISO/IEC 9075:1992, *Information technology — Database languages — SQL*

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

ISO 10303-21:2002, *Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure*

ISO 10303-41:2000, *Industrial automation systems and integration — Product data representation and exchange — Part 41: Integrated generic resource: Fundamentals of product description and support*

ISO 10303-42:2003, *Industrial automation systems and integration — Product data representation and exchange — Part 42: Integrated generic resource: Geometric and topological representation*

ISO 10303-43:2003, *Industrial automation systems and integration — Product data representation and exchange — Part 43: Integrated generic resource: Representation structures*

ISO 10303-227:2001, *Industrial automation systems and integration — Product data representation and exchange — Application protocol: Plant spatial configuration*

ISO 13584-1:2001, *Industrial automation systems and integration — Parts library — Part 1: Overview and fundamental principles*

ISO 13584-20:1998, *Industrial automation systems and integration — Parts library — Part 20: Logical resource: Logical model of expressions*

ISO 13584-26:2000, *Industrial automation systems and integration — Parts library — Part 26: Logical resource: Information supplier identification*

ISO 13584-31:1999, *Industrial automation systems and integration — Parts library — Part 31: Implementation resources: Geometric programming interface*

ISO 13584-42:1998, *Industrial automation systems and integration — Parts library — Part 42: Description methodology: Methodology for structuring part families*

IEC 61360-2:2002, *Standard data element types with associated classification scheme for electric components — Part 2: EXPRESS dictionary schema*

IAB RFC 1700:1994, *Internet architecture board internet standard (STD 2): Assigned numbers*

IAB RFC 1739:1994, *Internet architecture board proposed standard protocol: A primer on Internet and TCP/IP tools*

IAB RFC 1808:1995, *Internet architecture board proposed standard protocol: Relative uniform resource locators (URL)*

IAB RFC 1866:1995, *Internet architecture board proposed standard protocol: Hypertext markup language 2.0 (HTML-2.0)*

IAB RFC 2045:1996, *Internet architecture board draft standard protocol: Multipurpose internet mail extensions (MIME)*

IAB RFC 2049:1996, *Internet architecture board draft standard protocol: MIME conformance criteria (MIME-CONF)*

IAB RFC 2068:1997, *Internet architecture board proposed standard protocol: Hypertext transfer protocol HTTP/1.1 (HTTP-1.1)*

IAB RFC 2231:1997, *Internet architecture board Proposed standard protocol: MIME parameter value and encoded word extensions: Character sets, languages and continuations*

IAB RFC 2400:1998, *Internet architecture board internet standard (STD 1): Internet official protocol standard*

3 Terms, definitions and abbreviations

For the purposes of this document, the following terms and definitions apply. Some of these terms and definitions are repeated for convenience from:

— ISO 10303-1:1994;

— ISO 10303-11:1994;

— ISO 13584-1:2001;

— ISO 13584-31:1999.

— ISO 13584-42:1998.

3.1

absolute uniform resource locator
a string, the content of which uniquely identifies a network resource over the Internet

NOTE 1 The structure of absolute uniform resource locators are defined in IAB RFC 1738 [2].

NOTE 2 The information represented in an absolute uniform resource locator includes the following:

— the protocol to be used by the Internet client to access the resource, and

— the Internet address of the Internet server.

3.2

abstract part
a part that is only defined by a partial specification and that cannot be materially provided by the organisation that defines the specification, e.g., International Standard, functional specification (compare with a physical part)
[ISO 13584-1]

3.3

applicable property
a property that is defined for some family of parts and that shall apply to any part that belongs to this family of parts

EXAMPLE For a generic family of screws, the threaded diameter is an applicable property. This characteristic applies to any screw.

3.4

application
a group of one or more processes creating or using product data
[ISO 10303-1, definition 3.2.2]

3.5

application context
the conditions that define the intended use of product data within an application
[ISO 10303-1, definition 3.2.4]

3.6

application programming interface
API
a set of functions that may be triggered by a program

3.7

application protocol
AP
a part of ISO 10303 that describes the use of (ISO 10303) integrated resources satisfying the scope and information requirements for a specific application context
[ISO 10303-1, definition 3.2.7]

3.8

assembled item
an item that is defined as a composition of other items

3.9

atomic item
an item that is not defined as a composition of other items

NOTE A part that consists of several subassemblies may be described as an atomic item if its class definition does not define its constituent subassemblies.

3.10

augmented Backus-Naur form
ABNF
the augmented version of Backus-Naur syntax notation defined in clause 2 of IAB RFC 1808

NOTE The notational conventions used in ABNF are summarised for convenience in the informative annex T.

3.11

basic semantic unit
BSU
the entity that provides an absolute and universally unique identification of certain objects of the application domain (e.g. classes, data element types)
[ISO 13584-42, definition 3.4.1]

3.12

characteristic of a part (part characteristic)
a constant property, characteristic of a part, of which the value is fixed once the part is defined

NOTE Changing the value of a characteristic of a part would mean changing the part.

EXAMPLE For a ball-bearing, the inner and outer diameters are part characteristics.

3.13

class extension
the set of all instances satisfying the class definition

3.14

class valued property
a property that has one single value for a whole family of parts. Its value is not defined individually for every single part of that family, but for the family itself
[ISO 13584-42, definition 3.4.2]

NOTE Class valued properties may be used to capture some commonality between different families when such a commonality is not captured by the hierarchy structure.

3.15

common dictionary schema
the information model for a dictionary, using the EXPRESS modelling language, resulting from a joint effort between ISO TC184/SC4/WG2 and IEC SC3D
[ISO 13584-42, definition 3.4.3]

NOTE The common dictionary schema is specified in IEC 61360-2. Its content is provided in the informative annex D of ISO 13584-42.

3.16

completely defined instance
any particular instance of a class extension (compare with: partially defined instance)

3.17

conformance class
a subset of a standard for which conformance may be claimed

NOTE Adapted from ISO 10303-1.

3.18

conformance requirement
a precise text definition of a characteristic required to be present in a conforming implementation
[ISO 10303-1, definition 2.1.14]

3.19

conforming implementation
an implementation which satisfies the conformance requirements defined by one or several conformance classes of a standard

3.20

conformity; conformance
the fulfilment by an implementation of all specified requirements
[ISO 10303-31, definition 3.2.25]

3.21

context
the circumstances relevant to something under consideration

3.22

context-dependent characteristic of a part
a property of a part whose value depends on some context parameter(s)

NOTE For a given part a context-dependent characteristic is mathematically defined as a function whose domain is defined by some context parameter(s) that defines the part 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.

3.23

context parameter
a variable of which the value characterises the context in which it is intended to insert a part

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

3.24

data
a representation of facts, concepts or instructions in a formal manner suitable for communication, interpretation, or processing by human beings or computers
[ISO 10303-1, definition 3.2.14]

3.25

data element type
DET
a unit of data for which the identification, description and value representation have been specified
[ISO 13584-42, definition 3.4.4]

3.26

data exchange
the storing, accessing, transferring, and archiving of data
[ISO 10303-1, definition 3.2.15]

3.27

data type
a domain of values
[ISO 10303-11, definition 3.2.4]

3.28

database oriented navigation
within a user library, a navigation that results from querying the data delivered by library data suppliers as library delivery files

3.29

definition table
the definition of the set of parts that belong to a family of parts by means of a two-dimensional array finite or infinite where each row defines a part and where each column describes the corresponding values of a part characteristics

NOTE 1 In ISO 13584, only those family of parts that are associated with a class extension include the description of their definition tables.

NOTE 2 In ISO 13584, a definition table may be defined either explicitly or implicitly by means of basic domains, set operators and relational algebra operators.

3.30

derivation function
relationship serving to compute the value of a property from the values of other properties

3.31

derived characteristics
in a family of parts associated with a definition table, a set of part characteristics that does not belong to the (library data supplier defined) key of this definition table

NOTE A functional dependency exists between the identification characteristics and the derived characteristics. In ISO 13584, this functional dependency is represented by a derivation function.

3.32

dictionary
a table consisting of a series of entries. One meaning corresponds to each entry in the dictionary and one dictionary entry identifies one single meaning
[ISO 13584-1]

NOTE 1 In ISO 13584, the kinds of meaning intended to constitute dictionary entries are: supplier, class, property, program library, type, table and document.

NOTE 2 In ISO 13584, the information that represents a dictionary entry is split into three entities: a **basic_semantic_unit** (**BSU**), that provides for reference, a **dictionary_element** that describes the dictionary entry by means of attributes, and, possibly, a **content_item** entity that describes the dictionary entry by describing its content.

3.33

dictionary data
the set of data that describes hierarchies of families of parts and properties of these parts
[ISO 13584-42, definition 3.4.6]

NOTE The dictionary data should be exchanged using the common dictionary schema.

3.34

dictionary element
the set of attributes that constitutes the dictionary description of certain objects of the application domain (e.g. classes, data element types)
[ISO 13584-42, definition 3.4.7]

3.35

document oriented navigation
within a user library navigation that follows hypertext links to navigate between documents delivered by library data suppliers as library external files

3.36

entity
a class of information defined by common properties
[ISO 10303-11, definition 3.2.5]

3.37

entity data type
a representation of an entity. An entity data type establishes a domain of values defined by common attributes and constraints
[ISO 10303-11, definition 3.2.6]

3.38

entity (data type) instance
a named unit of data that represents a unit of information within the class defined by an entity. It is a member of the domain established by an entity data type
[ISO 10303-11, definition 3.2.7]

3.39

exchange structure
a computer-interpretable format used for storing, accessing, transferring, and archiving data
[ISO 10303-1, definition 3.2.17]

3.40

family of parts
a simple or generic family of parts

3.41

feature
an aspect of an item that can be captured by a class structure and set of properties and that cannot exist independently of the item

EXAMPLE 1 A form feature is an aspect of a part that conforms to some preconceived shape stereotype associated to dimensioning properties. It may be represented as an instance of a class that captures this shape stereotype.

EXAMPLE 2 In a piping component, an outlet is an aspect of a part that conforms to some preconceived function stereotype that is associated with properties (e.g., its name, its role). It may be represented as a feature.

3.42

functional model of a part
the library data that represent one representation category of a part in an integrated library
[ISO 13584-1]

3.43

functional view of a part
a data that represents one representation category of a part in product data
[ISO 13584-1]

NOTE The structure of a functional view does not depend on the part it represents.

3.44

generic family of parts
a grouping of simple or generic families of parts done for purposes of classification or for factoring common information

3.45

hypertext markup language
HTML
a particular implementation of ISO 8879 (SGML) that enables access to the network resources available on the Internet

NOTE 1 The current version of HTML, HTML/2.0, is defined by IAB RFC 1866.

NOTE 2 An HTML document encapsulates uniform resource locators (URL) as a means of accessing network resources available on the Internet.

3.46

general model of a part
the library data that carries the definition and identity of a part in an integrated library
[ISO 13584-1]

3.47

http file
a MIME-like file that may contain reference to other Internet resources by means of http URLs

NOTE 1 http URLs are URLs used to locate network resources via the HTTP protocol. Their particular syntax and semantics are defined in the RFC that specifies the HTTP protocol.

NOTE 2 The current version of the HTTP protocol, HTTP/1.1, is defined by IAB RFC 2068.

3.48

hypertext transfer protocol
HTTP
a particular application-level network protocol between an Internet server and an Internet client defined by a RFC from the IAB

NOTE The current version of the HTTP protocol, HTTP/1.1, is defined by IAB RFC 2068.

3.49

identification characteristics
a simple family of parts associated with a definition table, a set of part characteristics that constitute the (library data supplier-defined) key of this definition table

NOTE 1 When there exist several sets of part characteristics that might constitute a key of the definition table (candidate keys), the particular set that defines the identification characteristics is chosen by the library data supplier.

NOTE 2 Identification characteristics identify a part within its family.

NOTE 3 It is not allowed by ISO 13584 to reuse the same values of the identification characteristics at any time for two different parts, i.e., for two parts of which some non-identification characteristics are different. If such a situation is anticipated, some additional identification characteristics, such that a version, shall be added to discriminate both parts.

NOTE 4 A functional dependency exists between the identification characteristics and the other part characteristics. In ISO 13584, this functional dependency is represented by a derivation function.

3.50

implementation method
a technique used by computers to exchange data that is described using the EXPRESS data specification language

NOTE Adapted from ISO 10303-1.

3.51

implementation resources
the capabilities of a software system that shall be available to claim conformance to a particular conformance class of a view exchange protocol or both view exchange protocol and library integrated information model

3.52

information
facts, concepts or instructions
[ISO 10303-1, definition 3.2.20]

3.53

information model
a formal model of a set of facts, concepts or instructions to meet a specific requirement
[ISO 10303-1, definition 3.2.21]

3.54

instance
a named value
[ISO 10303-11, definition 3.2.8]

3.55

integrated library
operational system consisting of a library management system and a user library
[ISO 13584-1]

3.56

Internet
the network that complies with the standard rules defined by the Internet Architecture Board (IAB)

3.57

Internet architecture board
IAB
the organisation that specifies by means of Request For Comments (RFC) the standard rules that shall be followed by the Internet client and Internet server

NOTE Current versions of standard documents from the Internet Architecture Board (IAB) may be found on the Internet Official Server at the following uniform resource locator: http//DS.INTERNIC.NET.

3.58

Internet assigned numbers authority
IANA
the central coordinator for the assignment of unique parameter values for Internet protocols

NOTE Currently assigned values for the various series of protocol parameters for the Internet protocol suite are currently available as: ftp://ftp.isi.edu/in-notes/iana/assignments/.

3.59

Internet client
program that establishes connections over the Internet for the purpose of sending requests

NOTE Requests conformant with the hypertext transfer protocol are defined by IAB RFC 2068.

3.60

Internet server
program that accepts connections in order to service requests by sending back responses

NOTE Requests conformant with the hypertext transfer protocol are defined by IAB RFC 2068.

3.61

is-a relationship
the inheritance relationship defined in the object oriented paradigm

NOTE In ISO 13584 the is-a relationship holds between a family of parts and a generic family of parts to which the former family belongs.

3.62

is-case-of relationship
a relationship providing a formal expression of the fact that an object conforms to the partial specification defined by another object

NOTE In ISO 13584, all the properties and data types visible or applicable for some family of parts may be imported by all the families of parts that declare to be case-of the former family. These properties and data types may then be used to describe the latter families.

3.63

is-part-of
the aggregation part/whole relationship

NOTE In ISO 13584 the is-part-of relationship holds between a family of constituent parts and a family of assembled parts to which the constituent parts belongs.

3.64

is-view-of
a relationship providing a formal expression of the fact that an object is a representation of another object according to a particular perspective

EXAMPLE A set of geometric entities might provide a drafting representation of a particular screw. If both the set of geometric entities and the particular screw are represented as object, the is-view-of relationship holds between the former object and the latter object (in a drafting perspective).

3.65

item
a thing that can be captured by a class structure and a set of properties
[ISO 13584-42, definition 3.4.8]

3.66

library
See: Parts library, supplier library, user library
[ISO 13584-1]

3.67

library data supplier
an organisation that delivers a supplier library in the standard format defined in ISO 13584 and is responsible for its content
[ISO 13584-1]

3.68

library delivery file
a population of EXPRESS entity instances conforming to a library integrated information model and represented according to one of the implementation methods specified in ISO 10303

NOTE A library delivery file specifies the structure and the content of a supplier library. It may reference library external files.

3.69

library end-user
the user of an integrated library
[ISO 13584-1]

NOTE The library end-user:

— consults the data contained in the library;
— selects a given part;
— requests the transmission of a selected view of this part from the library system.

3.70

library exchange context
the set of one library delivery file and zero, one or more library external files that represent together a supplier library

3.71

library external file
a file, referenced from a library delivery file, that contributes to the definition of a supplier library

NOTE The structure and the format of a library external file is specified in the library delivery file that references it.

3.72

library integrated information model
LIIM
an EXPRESS schema that integrates resource constructs from different EXPRESS schemas for representing supplier libraries for the purpose of exchange and that is associated with conformance requirements

NOTE Three library integrated information models are defined in this part of ISO 13584 for representing different kinds of supplier libraries.

3.73

library management system
LMS
a software system enabling the library end-user to use the content of an integrated library
[ISO 13584-1]

NOTE This software system is not standardised.

3.74

library part
a part associated with a set of data that represents it in a library
[ISO 13584-1]

3.75

library part data
the data that represent a part in a library
[ISO 13584-1]

3.76

library specification of a class
the explicit representation of a class extension in a supplier library

NOTE In the ISO 13584 series, every class is intentionally defined through a dictionary element. A dictionary element allows defining a kind of part, characterized by abstract properties. A class extension enables the library data supplier to define those parts of the kind defined by a dictionary element that are existing in some context. They may be e.g., sold by the library data supplier.

3.77

local coordinate system
LCS
an orthogonal right-handed coordinate system used to orientate and to locate geometrical entities in space
[ISO 13584-31]

3.78

mandatory property
a property that shall have a value in any completely defined instance of a class (compare with: optional property)

NOTE As in the EXPRESS language, a NULL value for a property in an instance is represented by the fact that this property is associated with no value. No difference is made, at the data model level, between a still non-assigned value, and a NULL value.

3.79

MIME-like file
a file whose format is described by the MIME-defined set of fields of information

NOTE 1 Standard values for MIME-defined fields of information may be registered by IANA. In the context of ISO 13584-24, only such registered values are allowed for use to describe the format of files exchanged as library external file.

NOTE 2 IANA registration procedures for MIME-related facilities are defined by RFCs from the IAB. They are currently defined by IAB RFC 2048 [5].

NOTE 3 Currently assigned values for the various series of protocol parameters for the Internet protocol suite are currently available as: ftp://ftp.isi.edu/in-notes/iana/assignments/.

3.80

multi-purpose Internet mail extensions
MIME
a set of fields of information that enables one to specify the format of a file on the Internet

NOTE 1 The MIME set of fields of information, and their meaning, are currently defined by IAB RFC 2045, IAB RFC 2046 [3], IAB RFC 2047 [4], IAB RFC 2049, and IAB RFC 2231.

NOTE 2 Particular values for MIME fields of information may be registered by IANA. Such values are intended to be recognised by any Internet agent.

3.81

network resource
an addressable unit of information or service that can be accessed through a network

3.82

object view coordinate
OVC
a Cartesian coordinate system used to describe geometric representations of library parts

3.83

optional property
a property that does not need a value for some completely defined instance of some class

EXAMPLE Consider the class that describes an assembled part consisting of bolt + nut + optional washer components. It contains a characteristic, called *the_washer*, whose type is defined by a washer class. For those instances of this class that do not contain a washer, the *the_washer* characteristic is not assigned a value.

3.84

part
a material or functional element that is intended to constitute a component of different products
[ISO 13584-1]

3.85

partially defined instance
an abstraction of several different instances of a class extension that may prove useful to capture in some state of a design process

NOTE This part of ISO 13584 does not specify whether partially defined instances may be created by a library in a user modelling system. This is implementation dependent.

3.86

parts library
an identified set of data and possibly programs which may generate information about a set of parts
[ISO 13584-1]

3.87

physical part
a part that can exist in several equivalent copies and which is capable of being supplied by the library data supplier who describes the library data for this part (compare to: abstract part)
[ISO 13584-1]

3.88

population
a collection of entity data type instances
[ISO 10303-11, definition 3.2.11]

3.89

product
a thing or substance produced by a natural or artificial process
[ISO 10303-1, definition 3.2.26]

3.90

product data
a representation of information about a product in a formal manner suitable for communication, interpretation, or processing by human beings or by computers
[ISO 10303-1, definition 3.2.27]

3.91

property
an information that may be represented by a data element type
[ISO 13584-42, definition 3.4.10]

3.92

relative uniform resource locator
a simplified URL that uniquely identifies a network resource provided by the same Internet server as the one which served the document where the URL is used

NOTE 1 A relative uniform resource locator does not contain the location of the Internet server.

NOTE 2 The relative URL content and structure is currently defined by IAB RFC 1808.

3.93

representation
a description, drawing or depiction of something
[ISO 10303-227]

3.94

representation category
a concept used to distinguish between different possible user requirements regarding a part representation
[ISO 13584-1]

NOTE In the model defined in this International Standard, this distinction is formally expressed in terms of a view logical name and the view control variables.

3.95

representation property
a property of a representation of a part that is not a part characteristics

EXAMPLE In a functional model class that represents some simulation model for the different parts of a parts family, all the coefficients of these simulation models are representation properties. Unlike part characteristics, the values of these properties may be changed for a given part without changing the part. For instance, It is the case when the coefficients are more accurately computed.

3.96

request for comments
RFC
a document issued by the Internet Architecture Board(IAB) to specify a rule to be followed on the Internet

NOTE This part of ISO 13584 references, in its Normative reference clauses, several IAB RFCs for representing and accessing documents which constitute provisions of this part of ISO 13584. Parties to agreements based on this part of ISO 13584 are encouraged to investigate the possibility of applying the most recent RFCs released by the IAB for representing documents associated with a supplier library.

3.97

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

NOTE Adapted from ISO 10303-1.

3.98

simple family of parts
a set of parts of which each part may be described by the same group of properties

3.99

standard data
a requirement on a software system defined by means of EXPRESS entity (data type) instances that are supposed to be recognised by this software system

3.100

standardised identification hierarchy
a dictionary data that is defined by a standardisation organisation
[ISO 13584-42, definition 3.4.11]

3.101

structured query language
SQL
the query language for relational database defined by ISO/IEC 9075

3.102

supplier library
a set of data, and possibly of programs, for which the supplier is defined and that describes in the standard format defined in ISO 13584 a set of parts and/or a set of representation of parts
[ISO 13584-1]

3.103

uniform resource locator
URL
a mechanism specified by the Internet Architecture Board(IAB) to uniquely identify a network resource on the Internet

NOTE 1 An URL may be either an absolute URL or a relative URL.

NOTE 2 The URL mechanism is currently defined by IAB RFC 1738 [2] and IAB RFC 1808.

3.104

user library
the information that results from the integration of one or more supplier libraries by the library management system and possibly from a later adaptation performed by the user
[ISO 13584-1]

3.105

user modeling system
a software system enabling the library end-user to use the part representations generated by an integrated library

3.106

view control variable
a variable of enumerated type, that may be associated with a view logical name and intended to further specify the perspective adopted by the user regarding a part. (e.g., for geometry: 2D, wire frame, solid)
[ISO 13584-1]

3.107

view exchange protocol
VEP
a part of ISO 13584 that describes the use of resource constructs and of representation transmission interfaces that satisfy the information requirement for the exchange of one representation category of parts

3.108

view logical name
the identifier of a representation category corresponding to a perspective that can be adopted by a user regarding a part (e.g., geometry, inertia, kinematics, etc.)
[ISO 13584-1]

3.109

visible property
a property that is defined for some family of parts and that may or not apply to the different parts of this family of parts

EXAMPLE For a generic family of screws, the non-threaded length is a visible property: it is clearly defined for any screw, but only those screws with a non-threaded part have a value for this property.

4 Structure of ISO 13584-24

ISO 13584-24 has three main parts.

— The generic resources part provides resource constructs that are generic in nature. They are intended to be used both inside and outside the ISO 13584 Standard series. This intent was taken into account in their design.

— The parts library specific resources part provides resource constructs that are specific to the parts library application domain. They were not designed with the intent to be used outside the ISO 13584 Standard series.

— The library integrated information models part provides EXPRESS schemas that integrate resource constructs from the previous schemas, other parts of ISO 13584 and other International Standards for representing supplier libraries for the purpose of exchange.

4.1 Generic resources

The generic resources consist of the following EXPRESS schemas:

— **ISO13584_instance_resource_schema**,

— **ISO13584_library_expressions_schema**,

— **ISO13584_table_resource_schema**,

— **ISO13584_variable_semantics_schema**,

— **ISO13584_domain_resource_schema**.

These schemas provide resource constructs that are generic in nature. They may be used outside ISO 13584, and particularly in all the applications that use a data dictionary conformant with the ISO/IEC dictionary schema specified in IEC 61360-2 and whose content is duplicated in an informative annex of ISO 13584-42.

4.1.1 ISO13584_instance_resource_schema

The **ISO13584_instance_resource_schema** provides the resource constructs needed to describe instances of classes, or values of properties, whose corresponding data types are specified in accordance with the **ISO13584_ISO61360_dictionary_schema** specified in IEC 61360-2 or with the **ISO13584_extended_dictionary_schema** specified in this part of ISO 13584.

4.1.2 ISO13584_library_expressions_schema

The **ISO13584_library_expressions_schema** provides the resource constructs for representing expressions that evaluate to a value belonging to any of the data types defined in the **ISO13584_ISO61360_dictionary_schema**, specified in IEC 61360-2 or in the **ISO13584_extended_dictionary_schema** specified in this part of ISO 13584.

4.1.3 ISO13584_table_resource_schema

The **ISO13584_table_resource_schema** defines the set of resource constructs needed to describe tables and algebraic operations in tables. The algebraic operations in tables considered in the **ISO13584_table_resource_schema** include relational algebra, and set operations. This schema may be used to represent any kind of tables, regardless of whether they relate to the parts library application domain.

4.1.4 ISO13584_variable_semantics_schema

The **ISO13584_variable_semantics_schema** provides the resource constructs needed to reference the current values of the different elements that characterise an instance of a class, whether this class is modelled according to the **ISO13584_IEC61360_dictionary_schema**, or according to its extensions defined in the **ISO13584_extended_dictionary_schema**. Following the data model defined in ISO 13584-20 for representing variables and their value-assignment mechanism, these resources are defined as subtypes of **variable_semantics**.

NOTE **variable_semantics** is defined in ISO 13584-20.

4.1.5 ISO13584_domain_resource_schema

The **ISO13584_domain_resource_schema** provides the resource constructs needed to represent the set of allowed values that constitutes the domain of a variable in a given context. This set of allowed values may be independent of any other variables, or it may depend on the values of some other variables. The resource constructs introduced in this schema enable the characterisation of both kinds of domains. These resources are generic in nature and can be used for various purposes and in different application contexts. In this part of ISO 13584, these resources are used in the **ISO13584_library_content_schema** to define the extension of a class.

4.2 Parts library specific resources

The parts library specific resources consist of the following EXPRESS schemas:

— **ISO13584_extended_dictionary_schema**,

— **ISO13584_library_content_schema**,

— **ISO13584_external_file_schema**,

— **ISO13584_method_schema**.

These schemas provide resource constructs that are specific to the parts library application domain. They enable the representation, within a supplier library, of hierarchies of item classes and representations for a library of those item classes.

4.2.1 ISO13584_extended_dictionary_schema

The **ISO13584_extended_dictionary_schema** contains the extensions of the ISO/IEC dictionary schema required for parts library representing and exchange. These extensions include the following:

— representation of classes of features;

— representation of functional model classes and functional view classes;

— representation of properties of functional model classes and functional view classes; and

— association of tables and documents with classes defined in a supplier library.

4.2.2 ISO13584_library_content_schema

In a dictionary, classes are intentionally defined. Their sets of possible instances are neither explicitly nor implicitly specified. The role of the **ISO13584_library_content_schema** is to enable the

representation of class extensions. The set of possible instances is either explicitly described using a simple set structure, or implicitly defined by using the resources of the **ISO13584_domain_resource_schema**.

4.2.3 ISO13584_external_file_schema

The **ISO13584_external_file_schema** defines a mechanism and provides the resource constructs for referencing external files, whether these external files conform to an EXPRESS information model, and whether their file formats conform to an implementation method defined in ISO 10303. The **ISO13584_external_file_schema** includes, in particular, all the material needed to support Internet-oriented documents and access mechanisms in a supplier library.

4.2.4 ISO13584_method_schema

The **ISO13584_method_schema** provides the resource constructs for representing methods for creating library item representations in a user modelling system.

4.3 Library integrated information models

A library integrated information model is an EXPRESS schema that integrates resource constructs from the above schemas and from other parts of ISO 13584 or other International Standards for representing supplier libraries for the purpose of exchange. Library integrated information models are associated with conformance requirements. The library integrated information models defined in this part of ISO 13584 enable the exchange of three kinds of libraries between a library data supplier and a library end-user:

— libraries that consist of hierarchies of classes of parts, materials or features,

— libraries that consist of representations of parts, materials or features, and

— libraries that define new representation categories capable of being provided for any family of parts, materials or features.

NOTE Each library integrated information model defined in this part of ISO 13584 allows the exchange of only one of the above three kinds of libraries. However, due to the dictionary mechanism, these three kinds of libraries may be exchanged separately as three different supplier libraries, and then, these three different supplier libraries might be integrated in the same user library.

4.3.1 ISO13584_g_m_iim_schema and LIIM 24-1

The **ISO13584_g_m_iim_schema** specifies the information requirements for exchanging hierarchies of item classes, where items may be parts, materials or features. This schema is associated with a set of standard data, that defines the formats of library external files that may be referenced by **ISO13584_g_m_iim_schema** entity data type instances, and with implementation methods for the library delivery file. Together with the standard data specified in annex E and with the implementation methods specified in annex F, the **ISO13584_g_m_iim_schema** constitutes the library integrated information model LIIM 24-1. Conformance requirements to LIIM 24-1 are defined in clause 16 of this part of ISO 13584.

4.3.2 ISO13584_f_m_iim_schema and LIIM 24-2

The **ISO13584_f_m_iim_schema** specifies the information requirements for exchanging hierarchies of library representations of library-defined item classes. This schema is associated with a set of standard data, that defines the formats of library external files that may be referenced by **ISO13584_f_m_iim_schema** entity data type instances, and with implementation methods for the library delivery file. Together with the standard data specified in annex I and with the implementation methods specified in annex J, the **ISO13584_f_m_iim_schema** constitutes the library integrated information model LIIM 24-2. Conformance requirements to LIIM 24-2 are defined in clause 17 of this part of ISO 13584.

4.3.3 ISO13584_f_v_iim_schema and LIIM 24-3

The **ISO13584_f_v_iim_schema** specifies the information requirements for exchanging hierarchies of classes that define new representation categories capable of being provided for any library-defined item classes. This schema is associated with a set of standard data, that defines the formats of library external files that may be referenced by **ISO13584_f_v_iim_schema** entity data type instances, and with implementation methods for the library delivery file. Together with the standard data specified in annex M and with the implementation methods specified in annex N, the **ISO13584_f_v_iim_schema** constitutes the library integrated information model LIIM 24-3. Conformance requirements to LIIM 24-3 are defined in clause 18 of this part of ISO 13584.

NOTE 1 Classes that define representation categories are functional view classes. Functional view classes are defined in ISO 13584-1.

NOTE 2 Only supplier-defined functional view classes need to be exchanged using the library integrated information model LIIM 24-3. Functional view classes defined by view exchange protocols are already stored in the dictionary of any implementation that claims conformance to these view exchange protocols.

5 Fundamental concepts and assumptions

The following concepts and assumptions apply to this part of ISO 13584.

5.1 Conceptual model of a supplier library

The conceptual model of a supplier library is described in Part 10 of ISO 13584. This conceptual model uses the object oriented paradigm for representing as a class:

— a family of parts; such a class is called a general model class,

— the representations of the different parts of a parts family belonging to the same representation category; such a class is called a functional model class, and

— the definitions of the different representation categories capable of being provided for any family of parts; such a class is called a functional view class.

These classes are structured hierarchically following the simple inheritance relationship.

This part of ISO 13584 provides the EXPRESS resource constructs for representing such a hierarchy of classes for the purpose of exchange.

5.2 Implicit versus explicit description of a parts library

At the lower level of the hierarchy of general model classes are represented simple families of parts. Two approaches may be used for representing simple families of parts.

NOTE a simple family of parts is a set of parts where each part may be described by the same group of properties (see clause 3).

5.2.1 Explicit modelling of simple families of parts: by set extension

Explicit description consists of representing separately each part of the family, and gathering all these descriptions within a set structure.

EXAMPLE Let's assume a family *F* is to be captured. It is described by part characteristics *a*, *b* and *c*, and contains 90 different parts defined by:

— *a* = 1, 2, ..., 10;

— b = 1, 2, ..., 10;

— c = (a + b) / 2, and;

— there exists no part such that a = b.

A simplified information model stated in EXPRESS is presented in Figure 1. Following this information model, the family *F* might be described, using the ISO 10303-21 implementation method, as presented in Figure 2.

```
ENTITY dic_part;
     family_name: STRING;
     properties: LIST [1:?] OF property_value;
END_ENTITY;

ENTITY property_value;
     property_name: STRING;
     value: REAL;
END_ENTITY;
```

Figure 1 — Simplified example of an explicit information model for families of parts

```
#10=DIC_PART('F', (#100, #101, #102));
     #100=PROPERTRY_VALUE('a', 1.0);
     #101=PROPERTRY_VALUE('b', 2.0);
     #102=PROPERTRY_VALUE('c', 1.5);
...
#90=DIC_PART('F', (#900, #901, #902));
     #900=PROPERTRY_VALUE('a', 10.0);
     #901=PROPERTRY_VALUE('b', 9.0);
     #902=PROPERTRY_VALUE('c', 9.5);
```

Figure 2 — Example of explicit description of a family of parts

The set of instances is defined by set extension: every instance of the set is explicitly defined.

NOTE As shown by the above example the description and the data management of explicitly described families is rather simple provided that the number of instances is not too large.

5.2.2 Implicit modeling of simple families by entity data type

Using an entity-data-type-oriented information modeling language, implicit description consists of capturing all the set of instances in an entity data type, using the capabilities of the modeling language to restrict the implicit content of this set of instances.

EXAMPLE Using the EXPRESS language, the family *F* defined in the previous example might be modeled according to Figure 3. Such an information model implicitly describes that there exist 90 value-different instances of the *F* family. The set of instances is defined by intention.


```
ENTITY F;
     a: REAL;
     b: REAL;
DERIVE
     c: REAL := (a + b) / 2;
WHERE
     WR1: SELF.a IN
          [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0];
     WR2: SELF.b IN
          [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0];
     WR3: SELF.a <> SELF.b;
END_ENTITY;
```

Figure 3 — Example of implicit description of a parts family in the EXPRESS language

NOTE From the user perspective, an implicitly-decribed parts family is a black-box where the user puts in some parameters values and gets back a set of part instances that conform to those values. Within the black-box, the family may be represented using tables, query functions, expressions, etc.

5.2.3 Explicit and implicit description of classes in this part of ISO 13584

This part of ISO 13584 lets the library data suppliers choose which approach they want to use for describing the content of their families. A family is always described by a class, but the set of all instances satisfying a class definition may be represented either explicitly, or implicitly.

Advantages of the implicit approach for family representation are:

— Storage reduction for families that contain a large number of parts.

EXAMPLE 1 More than 5 billion of different screws conforming to ISO screw standards exist.

— The capability to model families of customisable parts where some properties may have any value in some continuous ranges.

EXAMPLE 2 In the catalogue of a linear bearings supplier, the length of the guide way of a linear bearing may be ordered for any value within some continuous range.

— The capability to model not only the characteristics of the parts, but also the context parameters whose values do not belong to part definition but that enable a part to be selected through the requirements that must be fulfilled for it to meet the user's needs.

EXAMPLE 3 Assume that, in a family of parts called F, there exists a context parameter called *requirement*, that may take any integer value between 1 and 9. Assume that, for each value, there exists only one part that fulfil the corresponding requirements, the part whose attribute *a* equals "requirement -1", and whose attribute *b* equals "requirement +1". The corresponding constraints may be modelled in EXPRESS. The family F might be modelled as shown in Figure 4. Using such a description, a part of the family may be selected either through the *a* or *b* characteristics, or through the *requirement* context parameter.


```
ENTITY F;
      requirement: INTEGER;
      a: REAL;
      b: REAL;
WHERE
      WR1: SELF.a IN
           [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0];
      WR2: SELF.b IN
           [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0];
      WR3: SELF.a <> SELF.b;
      WR4: {1 < SELF.requirement <= 9};
      WR5: (SELF.a = SELF.requirement - 1.0) AND
           (SELF.b = SELF.requirement + 1.0);
END_ENTITY;
```

Figure 4 — Capturing context parameters in an implicit description

NOTE In Figure 4, the constraints are directly defined in EXPRESS. They cannot be exchanged using any of the implementation methods defined in ISO 10303. The **ISO13584_domain_resource_schema** documented in clause 10 of this part of ISO 13584 provides EXPRESS resource constructs that permit implementations to exchange such constraints.

Advantages of the explicit approach for family representing include:

— Easiness of description of a library by the library data supplier;

— Easiness of implementation of LMS that support only explicit description of class extension;

— Easiness of modeling assemblies when the number of possible configurations is rather small;

— Easiness of splitting the properties associated with a part in a general model and various functional models and of gathering them again during user access.

5.3 Direct use of EXPRESS versus meta-modelling for implicit description

Two approaches may be used for implicitly representing families of parts using the EXPRESS language.

5.3.1 Direct use of the EXPRESS language for modelling classes

The use of a direct EXPRESS model of a class, as shown in Figures 3 and 4 has the following advantages:

— the notion of a class in this International Standard may be mapped onto the EXPRESS notion of an entity data type;

— some required aspects of classes are enforced by the syntax of the EXPRESS language;

EXAMPLE 1 The constraint that an attribute shall be followed by a unique data type is a syntactic constraint.

— other required aspects of classes are enforced by the semantics of the EXPRESS language;

EXAMPLE 2 The constraint that only inherited or defined attributes may be referred to in the description of an entity data type is a semantic constraint.

— a mapping of a population of an EXPRESS model to an external file format exists.

EXAMPLE 3 The mapping defined in ISO 10303-21 is such a mapping.

Directly modelling classes in EXPRESS has the following disadvantages:

— classes and instances are modelled and processed completely differently;

EXAMPLE 4 An EXPRESS schema is represented in ASCII and is intended to be compiled. A set of instances is represented, e. g., according to ISO 10303-21 and is intended to be read by a STEP processor.

— only those concepts that exist in the EXPRESS language may be used in a class description.

EXAMPLE 5 In EXPRESS, it is not possible to capture the fact that the *requirement* attribute in Figure 4 is semantically different from the attributes *a* and *b*. The value of the *requirement* attribute may be changed without changing the part, whereas changing either *a* or *b* would change the part.

5.3.2 Meta-modelling of classes using EXPRESS

A meta-model is a model that provides the ability to represent other models.

Using the meta-modelling approach, a class is represented as a set of instances of the EXPRESS entities belonging to a meta-model.

Figure 5 shows a simple meta-model of a part class. The actual information model of a part class is located in the **ISO13584_library_content_schema** in clause 12. The class shown in Figure 4 directly modelled in EXPRESS might be represented in a similar way as shown in Figure 6 using the meta-model of Figure 5. Note that the actual information model of constraints is located in the **ISO13584_domain_resource_schema** in clause 10.

This part of ISO 13584 uses a meta-modelling approach for representing the three kinds of classes described in clause 5.1.

NOTE As shown in Figure 1, another information model may be defined for representing instances. Therefore, class and instance descriptions may be modelled at the same level and exchanged within the same population of EXPRESS entity data type instances. The **ISO13584_instance_resource_schema**, documented in clause 6, provides EXPRESS resource constructs for representing class instances.

```
...
ENTITY part_class;
     characteristics: SET [0:?] OF property_definition;
     domain: SET [0:?] OF constraint;
END_ENTITY;

ENTITY property_definition;
     property_name: STRING;
     value_domain: data_type_specification;
END_ENTITY;

ENTITY data_type_specification;
     type_name: STRING;
END_ENTITY;
...
```

Figure 5 — Simple meta-model of a part class in EXPRESS

```
...
#10=PART_CLASS((#20,#30,#40),(#50,#60,#70,#80,#90));
#20=PROPERTY_DEFINITION("a", #120);
#30=PROPERTY_DEFINITION("b", #120);
#40=PROPERTY_DEFINITION("requirement", #120);
```



```
#50= ... /* constraint: a is in [1.0, 2.0, ..., 10.0] */
#60= ... /* constraint: b is in [1.0, 2.0, ..., 10.0] */
#70=CONSTRAINT(.NOT_EQUAL.,#20,#30); /* SELF.a <> SELF.b */
#80=CONSTRAINT(.EQUAL., #20, #100); /* SELF.a = value of #100 */
#90= ... /* constraint that b = SELF.requirement + 1.0 */
#100=LESS_EXPRESSION(#40, #110); /* SELF.requirement - 1.0 */
#110=REAL_VALUE(1.0);
#120=DATA_TYPE_SPECIFICATION('REAL_TYPE');
...
```

Figure 6 — Model of a part family using a meta-modelling approach

5.4 Two level description of a supplier library and the ISO/IEC common dictionary schema

Each parts family that constitutes a supplier library has two levels of description:

— The first level describes the concepts about the parts family. Each parts family is both informally defined by a natural language description and formally defined by a set of attributes. This first level constitutes the dictionary definition of the corresponding parts family. It is represented as a **dictionary_element**. A **dictionary_element** may be exchanged. It may also be stored in the semantic dictionary of the receiving system.

— The second level describes the set of parts belonging to the family. This second level constitutes the library specification of the corresponding parts family. It is represented as a **content_item** intended to be stored in a user library.

The **dictionary_element** describes only the abstract concept to which the parts family corresponds. It provides the information that determines whether a part may belong to that parts family. It does not define the set of parts that constitute the parts family.

The **content_item**, when present, completes the above description by specifying all the parts belonging to the parts family.

A large degree of independence separates these two levels of description. First, some parts families have only a dictionary element, and not a formally defined content. This is the case for the abstract parts families described in the terminology or dictionary standards.

EXAMPLE IEC 61360-4 is an example of dictionary standard. This Standard describes, through a hierarchical structure, many families of parts that exist in electronics.

Secondly, the extension of a class, modelled by **content_item**, may change without altering the dictionary elements that apply to this class. This is the case when a product standard or a catalogue is updated by adding or withdrawing some parts of a parts family.

This part of ISO 13584 provides for representing either the first level, or both the first and second levels:

a) Parts libraries that contain only **dictionary_element**s are modelled through the **dictionary** entity defined in the **ISO13584_extended_dictionary_schema**. They correspond to conformance classes 1, 2 and 3 of library integrated information model 24-1 specified in clause 16 of this part of ISO 13584.

b) Parts libraries that contain both **dictionary_element**s and **content_item**s are modelled through the **library** entity defined in the **ISO13584_library_content_schema**. They correspond to conformance class 4, 5 and 6 of library integrated information model 24-1 specified in clause 16 of this part of ISO 13584.

5.4.1 Common dictionary description for ISO 13584 and IEC 61360

The **ISO13584_IEC61360_dictionary_schema**, documented in an informative annex of ISO 13584-42: 1998, is a common resource for ISO 13584 and for IEC 61360. This schema has been defined by ISO TC 184/SC4/WG2 and IEC SC3D for the exchange of those **dictionary_element**s that address their common scope. This scope consists of families of parts linked by the *is-a* relationship and described by a set of properties.

5.4.2 Dictionary descriptions for ISO 13584

The resource constructs needed to represent the **dictionary_element**s that are in scope for ISO 13584 and out of scope for IEC 61360, are provided in the **ISO13584_extended_dictionary_schema** defined in clause 11 of this part of ISO 13584. The scope of the **ISO13584_extended_dictionary_schema** is an extension of the **ISO13584_IEC61360_dictionary_schema**. It references all the resources defined in the **ISO13584_IEC61360_dictionary_schema**.

NOTE The **dictionary_element**s that are in scope for ISO 13584 and out of scope for IEC 61360 include: representation of functional model classes, representation of functional view classes, aggregation relationships.

5.4.3 Interoperability of ISO 13584 and IEC 61360

This part of ISO 13584 specifies a number of options that may be supported by an implementation. These options have been grouped into conformance classes. Conformance requirements are defined in clause 15 of this part of ISO 13584. In particular, conformance to a particular conformance class requires that all entities, types, and associated constraints defined as part of that conformance class be supported.

Conformance class 0 of the library integrated information model 24-1, documented in clause 16 of this part of ISO 13584, only requires an implementation to support the common ISO and IEC requirements defined in the **ISO13584_IEC61360_dictionary_schema**. Thus, an ISO 13584 implementation that conforms to that conformance class is able to handle any instance population of the ISO/IEC common **ISO13584_IEC61360_dictionary_schema**, using an implementation method defined for conformance class 0. Moreover, any conformance class of the library integrated information model 24-1 including the requirements defined in conformance class 0, is able to do the same.

NOTE The achievement of such an interoperability has introduced more complexity in both the ISO/IEC common **ISO13584_IEC61360_dictionary_schema**, that provides hooks for the extensions needed for ISO 13584, and in the **ISO13584_extended_dictionary_schema**.

5.5 Independence between dictionary_elements and content_items: the BSU mechanism

In the ISO 13584 Standard series, each piece of information intended to constitute a dictionary entry is represented through three entities:

a) the **basic_semantic_unit** (**BSU**) entity carries the universal identification of this piece of information;

b) the **dictionary_element** entity contains the set of attributes that constitutes the dictionary description of this piece of information;

EXAMPLE 1 Name, definition and type of value are examples of attributes that constitute the dictionary description of a piece of information.

c) the **content_item** entity represents the possible values of this piece of information.

EXAMPLE 2 The **content_item** of a class, represented by the **class_extension** entity, specifies the possible instance values of this class.

NOTE In ISO 13584, the pieces of information intended to constitute dictionary entries are: supplier, class, property, program library, type, table and document.

5.5.1 Reference between several EXPRESS schema populations via the BSU mechanism

All the references between dictionary entries shall be done through their **BSU**s. Thus, when a dictionary entry is intended to be referenced in some population, only its **BSU** entity instance is requested to belong to this population. Reference to a **BSU** stands for a reference to the complete piece of information that constitute a dictionary entry, whether its **dictionary_element** and **content_item** entities belong to the same population. Thus, this three levels model ensures independence between the **dictionary_element** entity instance, the **content_item** entity instance, and the **BSU** entity instance for any dictionary entry. This mechanism also provides a means to implement references between dictionary entries belonging to different EXPRESS schema populations, possibly stored in different exchange files or data repositories.

EXAMPLE 1 The **content_item** of a class may references the properties that characterise the instances of the class through their **BSU**s. If the **dictionary_element**s that describe those properties are assumed to be available on the receiving system, the sending system may decide not to send these **dictionary_element**s together with the **content_item**s that reference them.

EXAMPLE 2 In product model data, a product may be associated with a particular value for some property. When this property is modeled according to the **ISO13584_IEC61360_dictionary_schema**, only the **BSU** of this property needs to be represented in product data.

5.5.2 Expressing constraints between dictionary entries

All the constraints between **dictionary_element**s and/or **content_item**s of dictionary entries are expressed by means of references to their **BSU**s. But these **dictionary_element**s and/or **content_item**s may not be available in the exchange context where the **BSU**s are available. Thus, when the role of a constraint is to restrict the allowed set of values of an attribute of either a **dictionary_element** or a **content_item**, the constraint may only be checked when and where the corresponding instance is available, otherwise the constraint shall be ignored. This behaviour is specified, at the EXPRESS code level, either by using the **definition_available_implies** function, for **dictionary_element** availability, or by checking the size of the aggregate value of the **referenced_by** attribute of a **basic_semantic_unit**, for **content_item** availability.

NOTE **definition_available_implies** and **basic_semantic_unit** are defined in the **ISO13584_IEC61360_dictionary_schema** specified in IEC 61360-2, and duplicated for convenience in informative annex D of ISO 13584-42.

5.6 ISO 13584 and the Internet

5.6.1 Documents represented within a library exchange context

A library exchange context includes a library delivery file conforming to one conformance class of a library integrated information model and several library external files. These library external files are referenced from the library delivery file using **external_item** entities. Such entities specify the content of the external files, the protocol that shall be used to process them and the external file addresses that identify the various library external files within the library exchange context.

This part of ISO 13584 includes provisions for exchanging documents as library external files.

5.6.2 Support of the HTTP protocol and local Internet server

One particular network protocol for accessing documents is the HTTP protocol. This protocol provides for accessing documents whose contents are represented by means of several files, possibly encoded

according to different file formats defined as MIME-like file, and possibly referencing other Internet resources by means of HTTP URLs.

This part of ISO 13584 includes provisions for exchanging documents by means of library external files intended to be accessed using the HTTP protocol. Such library external files are called http files. Only http files whose values of MIME-defined fields of information registered by IANA are allowed for use in the context of this part of ISO 13584.

When a conformance class of a LIIM or a conformance class of a VEP specifies that library external files according to the HTTP protocol shall be supported, an implementation that claims conformance to this conformance class shall include:

a) the capabilities of an Internet server, as defined in IAB RFC 2068, for the purpose of storing documents;

b) the capabilities of an Internet client, as defined in IAB RFC 2068, and the capabilities defined in IAB RFC 2049 for the purpose of recognising and browsing these documents; and

c) a data repository, and its management system, to store and retrieve the data exchanged in the library delivery file.

The mechanisms defined in clause 13 of this part of ISO 13584 ensure that the hyperlink references specified by the library data supplier remain valid on the local Internet server. They also enable the library end-user to switch from document navigation to data retrieval and selection.

5.6.3 Particular HTTP formats to be supported by an implementation

In the context of the HTTP protocol, files exchanged over the Internet are MIME-like files. MIME defines the format of an exchanged file by means of several fields of information, called header fields and parameters, that are supposed to be added in the header of the files by the HTTP server.

In the context of this part of ISO 13584, the values of MIME-defined fields of information that characterise the format of the http files that are exchanged as a library external files are represented explicitly in the **http_file** entity that references them. When the supplier library is compiled on the receiving system site, these values are supposed to be used both to filter the files that the implementation intends to record, and to initiate the receiving site Internet server. These values are not duplicated in any header part of the library external files.

This part of ISO 13584 uses the following MIME-defined header fields and parameters to specify the format of a file:

NOTE 1 These MIME-defined header fields and parameters are documented in IAB RFC 2045 and IAB RFC 2184.

— *MIME-Version* that defines the version of MIME: the value of this attribute is implicit, and it equals 1.0;

— *Content-Type* that defines the media type and subtype of data in the file;

EXAMPLE 1 "image" is an example of media type; "jpeg" is an example of subtype.

— *Content-Transfer-Encoding* that defines the encoding, or lack of encoding, of the file data for the purpose of exchange in the library exchange context.

EXAMPLE 2 "base64" is an example of data encoding defined in IAB RFC 2045.

— *Character set* parameter that specifies a method of converting a sequence of octets into a sequence of characters;

EXAMPLE 3 "ISO-8859-1" is an example of MIME "character set". The corresponding method of converting a sequence of bytes into a sequence of characters is defined in ISO 8859-1.

NOTE 2 ISO 8859-1 defines an 8-bit character encoding that includes most of the specific characters of western languages.

— *Language* parameter that specifies, either implicitly or explicitly, the language of a document.

This part of ISO 13584 does not specify the formats that are allowed for use for the http files belonging to a library exchange context. It only specifies that:

— only values of MIME header fields and parameters registered by IANA are allowed;

— the local Internet client defined in the previous clause shall meet, at the minimum, the MIME conformance criteria defined in IAB RFC 2049 for the purpose of recognising and browsing http files;

— the local Internet client defined in the previous clause shall recognise "text" documents conforming to HTML as defined by IAB RFC 1866.

Apart from private agreement between the sender and the receiver, this part of ISO 13584 also strongly recommends to restrict to a small number of formats:

— that are mature;

— that are stable;

— that are unambiguously characterised by MIME Content-Type;

— whose specifications are publicly available, or that are associated with public domain Internet-available readers.

5.6.4 Remote access to a document through the Internet

This part of ISO 13584 contains provisions that enable a library data supplier to deliver documents within a library exchange context. It also contains provisions for providing a location where a document may be found by means of an absolute URL. Remote access to such documents is not required for the implementation that claims conformance to any library integrated information model defined in this part of ISO 13584.

6 ISO13584_instance_resource_schema

This clause defines the requirements for the **ISO13584_instance_resource_schema**. The following EXPRESS declaration introduces the **ISO13584_instance_resource_schema** block and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA ISO13584_instance_resource_schema;

REFERENCE FROM ISO13584_IEC61360_dictionary_schema
     (all_class_descriptions_reachable,
     basic_semantic_unit,
     class,
     class_BSU,
     condition_det,
```



```
      content_item,
      data_type_BSU,
      definition_available_implies,
      dependent_p_det,
      is_subclass,
      level,
      list_to_set,
      non_quantitative_code_type,
      non_quantitative_int_type,
      property_BSU,
      version_type);

REFERENCE FROM ISO13584_IEC61360_language_resource_schema
      (translatable_label,
      present_translations);

REFERENCE FROM ISO13584_extended_dictionary_schema
      (a_priori_semantic_relationship,
      abstract_functional_model_class,
      applicable_properties,
      data_type_class_of,
      data_type_level_spec,
      data_type_level_value_typeof,
      data_type_type_name,
      data_type_typeof,
      functional_view_v_c_v,
      data_type_non_quantitative_code_type,
      data_type_non_quantitative_int_type);

REFERENCE FROM ISO13584_library_content_schema
      (allowed_properties,
      explicit_functional_model_class_extension,
      explicit_item_class_extension,
      functional_model_class_extension,
      fm_free_model_properties_list,
      item_class_extension,
      gm_identification_characteristics_list,
      method_variables,
      selectable_properties_list);

REFERENCE FROM ISO13584_external_file_schema
      (program_reference,
      representation_reference,
      property_value_external_item);

REFERENCE FROM geometry_schema
      (axis1_placement,
      axis2_placement_2d,
      axis2_placement_3d,
      placement);
```



```
    REFERENCE FROM representation_schema
          (representation,
          representation_context,
          representation_item);

    REFERENCE FROM product_definition_schema
          (product,
          product_category,
          product_definition,
          product_definition_formation);

    REFERENCE FROM product_property_definition_schema
          (property_definition);

    USE FROM person_organization_schema
          (address,
          organization,
          person,
          person_and_organization,
          personal_address,
          organizational_address );

    USE FROM date_time_schema
          (date,
          date_and_time,
          local_time,
          calendar_date,
          ordinal_date,
          week_of_year_and_day_date);

    REFERENCE FROM geometry_schema
          (geometric_representation_context);
    (*
```

NOTE The schemas referenced above can be found in the following documents:

ISO13584_IEC61360_dictionary_schema	IEC 61360-2

(which is duplicated for convenience in informative annex D of ISO 13584-42),

ISO13584_IEC61360_language_resource_schema	IEC 61360-2

(which is duplicated for convenience in informative annex D of ISO 13584-42),

ISO13584_extended_dictionary_schema	This part of ISO 13584,
ISO13584_library_content_schema	This part of ISO 13584,
ISO13584_external_file_schema	This part of ISO 13584,
geometry_schema	ISO 10303-42,
representation_schema	ISO 10303-43,
product_definition_schema	ISO 10303-41,
product_property_definition_schema	ISO 10303-41,
person_organization_schema	ISO 10303-41.
date_time_schema	ISO 10303-41.

6.1 Introduction to the ISO13584_instance_resource_schema

The **ISO13584_instance_resource_schema** defines the resources needed to describe instances of classes and values of properties whose data types are specified in accordance with this International Standard.

NOTE The approach followed in this schema is largely based on the SDAI approach for describing EXPRESS types and instances, which is specified in ISO 10303-22 [7].

The **ISO13584_instance_resource_schema** models:

— the description of instances of properties whose types are specified either by the **ISO13584_IEC61360_dictionary_schema** resource constructs or by the **ISO13584_extended_dictionary_schema** resource constructs;

— the description of instances of classes whose dictionary elements are specified either by **ISO13584_IEC61360_dictionary_schema** resource constructs or **ISO13584_extended_dictionary_schema** resource constructs;

— the description of instances of classes whose contents are specified by **ISO13584_library_content_schema** resource constructs;

— the mechanisms for computing the data type to which an instance belongs and for ensuring type checking.

The **ISO13584_instance_resource_schema** does not model:

— the specification of the internal representation of a created instance or a selected property in a user library during a user selection process.

6.2 Fundamental concepts and assumptions for the ISO13584_instance_resource_schema

Description of classes requires capabilities for representing values of properties and instances of classes.

EXAMPLE To represent the domain of a data type that is restricted by a constraint defined by a table, one needs to represent the various values contained in the table. These values are values of properties or instances of classes.

NOTE 1 A value of a property is an instance of a class when the data type of this property is a **class_instance_type**.

NOTE 2 **class_instance_type** is defined in the **ISO13584_IEC61360_dictionary_schema** documented in IEC 61360-2 (which is duplicated for convenience in informative annex D of ISO 13584-42).

6.2.1 Two-fold description of classes and instance representation

In this International Standard, a class may be specified at two levels of abstraction, either in intention by means of dictionary data, or as a class extension, by means of the resource constructs defined in the **ISO13884_library_content_schema** contained in this part of ISO 13584. Figure 7 shows a planning model of this two-fold class description, with the associated instance descriptions.

A dictionary description, provided by a **class** entity, defines the properties that are applicable to the class and their data types. An instance of a class that is only described by a **class** entity is represented as a **dic_class_instance**. It may include any of these applicable properties, and each property may have any value conformant with its data type.

A library specification, represented as a **model_class_extension**, further specifies the class by defining:

— what subset of the applicable properties have their values provided in the library to describe the different instances;

EXAMPLE 1 In the **class** dictionary definition of a screw class, the "mass" property may be listed as an applicable property. Nevertheless, the library data supplier may decide not to provide the "mass" values for the various screws represented by the screw class.

— the allowed set of values for those properties provided;

— the subset of these properties that is needed to completely identify one instance in this class;

— how the other property values may be derived from the above properties (derivation functions).

NOTE 1 Derivation functions are used to specify how values of some properties may be computed from values of other properties.

An instance of such a class, represented either as a **lib_item_instance**, in the case of an **item_class**, or as a **lib_f_model_instance**, in the case of a **functional_model_class**, shall only be defined:

— by properties that are provided in the class extension description, and

— by property values belonging to the allowed set of values for the class instance.

Each instance of such a class shall be associated with all the properties that are needed to identify one instance within its class. When such an instance is a partially defined instance, some of these properties may be associated with no value.

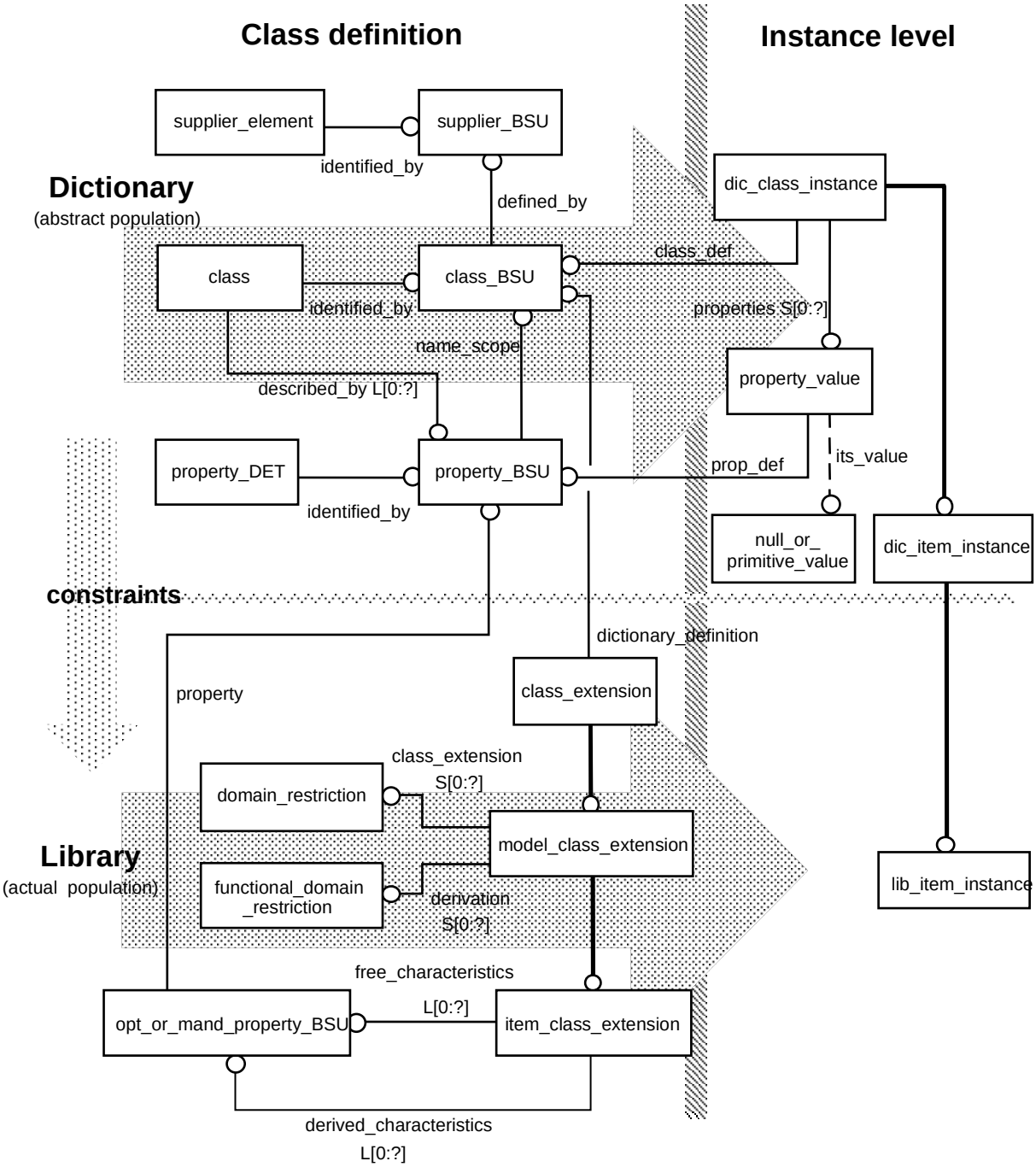


Figure 7 — Planning model of the relationships between class definition and the instance level

NOTE 2 Classes only specified by a dictionary element are able to represent standardised identification hierarchies intended to provide common semantics for computer interpretable data exchange. Classes also specified by a library specification are able to represent supplier catalogues or product standards.

NOTE 3 When a class is represented by an **item_class**, the properties needed to completely identify an instance are computed by the **gm_identification_characteristics_list** function. When a class corresponds with a **functional_model_class**, the properties needed to completely characterise an instance are computed by the **selectable_properties_list** function.

EXAMPLE 2 The component classes and data element types defined in IEC 61360-4 describe the components and properties used in the electronic applications area. This description constitutes a set of **dictionary_element**s conformant to this International Standard. An instance of a component class defined in IEC 61360-4 (for instance, a *fixed_linear_resistor*) may be modelled as a **dic_class_instance**.

NOTE 4 When a class is only defined through a dictionary element, such as the various classes defined in IEC 61360-4, the properties applicable to this class are only associated with a data type. In an instance of such a class, each property may have whatever value belonging to its data type.

EXAMPLE 3 For describing its own parts catalogue, John Croke Inc. does not need only to capture the fact that the components produced in the factory are kind of "*light_dependent_resistors*", as defined in IEC 61360-4. It also wants to describe which particular "*light_dependent_resistors*" the company is able to provide. This can be done by a library specification.

In the library specification, John Croke Inc describes the different *light_dependent_resistors* contained in its catalogue. It also specifies that the *resistance* is the only property needed to identify any resistor when such a resistor is ordered in its company. It may also provide tables that enable the system to generate the values of all the other properties he wants to provide for its *light_dependent_resistors* from the value of the *resistance*.

A resistor that is an instance of *light_dependent_resistors* class may be modeled as a **lib_component_instance** (a subtype of **lib_item_instance**). It will contain a **property_value** that references its *resistance* property. The values of the other properties of its *light_dependent_resistors* may also be associated with this **lib_component_instance**. In this case, these values should comply with the values computed by the derivation function specified in the **item_class_extension**.

NOTE 5 *resistor_noise_index* is an example of property of resistors whose definition is provided in IEC 61360-4.

6.2.2 Representation of a context-dependent characteristic value

A context-dependent characteristic is a property whose value depends on some context parameter(s). If an instance is associated with a value of such a property, values shall also be provided for all the context parameters on which this value depends.

EXAMPLE The life span of a ball-bearing is a context-dependent characteristic. Its value depends on the radial load, the axial load and the rotational speed supported by that bearing. If an instance of a class that represents a ball-bearing family is associated with a value for the life span property, it shall also be associated with values for the radial load, the axial load and the rotational speed context parameters.

6.2.3 Optional properties

In the **ISO13584_library_content_schema**, a property of an item belonging to a library may be specified as optional. Such a property may be assigned a NULL value for some class instances. In the **ISO13584_instance_resource_schema**, the value of such a property is represented as an OPTIONAL value for the corresponding **property_value**.

NOTE This part of ISO 13584 does not make any difference between a NULL value and an indeterminate value (?). In a **property_value**, a lack of value is represented as indeterminate. In a table **column**, that is a list, a lack of value is represented by a NULL value**.**

6.3 ISO13584_instance_resource_schema type definitions

This clause defines the types for modelling the values that may be assigned to properties in the **ISO13584_instance_resource_schema**.

6.3.1 Null_value

The **null_value** entity provides for the definition of data values which may be indeterminate value according to the ? value of the EXPRESS language.

EXPRESS specification:

```
*)
```



```
ENTITY null_value;
END_ENTITY; -- null_value
(*
```

6.3.2 Primitive_value

A **primitive_value** is any value that may be assigned to a property in the **ISO13584_extended_dictionary_schema**.

EXPRESS specification:

```
*)
TYPE primitive_value = SELECT(
     simple_value,
     complex_value);
END_TYPE; -- primitive_value
(*
```

6.3.3 Null_or_primitive_value

A **null_or_primitive_value** is any value, included the **null_value**, that may be assigned to a property in the **ISO13584_extended_dictionary_schema**.

EXPRESS specification:

```
*)
TYPE null_or_primitive_value = SELECT(
     null_value,
     primitive_value);
END_TYPE; -- null_or_primitive_value
(*
```

6.3.4 Simple_value

A **simple_value** is an unstructured value belonging to the EXPRESS NUMBER, STRING or BOOLEAN types. A STRING value may be represented in different languages.

NOTE 1 A **translatable_string_value** is considered as a **simple_value** as only one particular **string_value** is supposed to be displayed in any context

NOTE 2 The function that chooses in any context the particular value to be displayed shall be implementation dependent and is outside the scope of this part of ISO 13584.

EXPRESS specification:

```
*)
TYPE simple_value = SELECT(
     number_value,
     translatable_string_value,
     boolean_value);
END_TYPE; -- simple_value
(*
```


6.3.5 Null_or_simple_value

A **null_or_simple_value** is an unstructured value belonging to the EXPRESS NUMBER, STRING or BOOLEAN types, included the **null_value**.

EXPRESS specification:

```
*)
TYPE null_or_simple_value = SELECT(
     null_value,
     simple_value);
END_TYPE; -- null_or_simple_value
(*
```

6.3.6 Number_value

A **number_value** is a value belonging to the EXPRESS NUMBER type.

EXPRESS specification:

```
*)
TYPE number_value = SELECT(
     integer_value,
     real_value);
END_TYPE; -- nunmber_value
(*
```

6.3.7 Null_or_number_value

A **null_or_number_value** is a value belonging to the EXPRESS NUMBER type, included the **null_value**.

EXPRESS specification:

```
*)
TYPE null_or_number_value = SELECT(
     null_value,
     number_value);
END_TYPE; -- nunmber_value
(*
```

6.3.8 Integer_value

An **integer_value** is a value belonging to the EXPRESS INTEGER type.

EXPRESS specification:

```
*)
TYPE integer_value = INTEGER;
END_TYPE; -- integer_value
(*
```

6.3.9 Null_or_integer_value

A **null_or_integer_value** is a value belonging to the EXPRESS INTEGER type, included the **null_value**.

EXPRESS specification:

```
*)
TYPE null_or_integer_value = SELECT(
     null_value,
     integer_value);
END_TYPE; -- null_or_integer_value
(*
```

6.3.10 Real_value

A **real_value** is a value belonging to the EXPRESS REAL type.

EXPRESS specification:

```
*)
TYPE real_value = REAL;
END_TYPE; -- real_value
(*
```

6.3.11 Null_or_real_value

A **null_or_real_value** is a value belonging to the EXPRESS REAL type, included the **null_value**.

EXPRESS specification:

```
*)
TYPE null_or_real_value = SELECT(
     null_value,
     real_value);
END_TYPE; -- null_or_real_value
(*
```

6.3.12 Boolean_value

A **boolean_value** is a value belonging to the EXPRESS BOOLEAN type.

EXPRESS specification:

```
*)
TYPE boolean_value = BOOLEAN;
END_TYPE; -- boolean_value
(*
```


6.3.13 Null_or_boolean_value

A **null_or_boolean_value** is a value belonging to the EXPRESS BOOLEAN type, included the **null_value**.

EXPRESS specification:

```
*)
TYPE null_or_boolean_value = SELECT(
     null_value,
     boolean_value);
END_TYPE; -- null_or_boolean_value
(*
```

6.3.14 Translatable_string_value

A **translatable_string_value** defines a type of value that is a STRING that may be translated in various languages.

NOTE 1 A **translatable_string_value** is considered as a **simple_value** as only one particular **string_value** is supposed to be displayed in any context

NOTE 2 The function that chooses in any context the particular value to be displayed shall be implementation dependant and is outside the scope of this part of ISO 13584.

EXPRESS specification:

```
*)
TYPE translatable_string_value = SELECT(string_value,
     translated_string_value);
END_TYPE; -- translatable_string_value
(*
```

6.3.15 Translated_string_value

A **translated_string_value** entity defines the **string_value**s that are translated in various languages, and the corresponding languages of translation.

EXPRESS specification:

```
*)
ENTITY translated_string_value;
     string_values: LIST [1:?] OF string_value;
     languages: present_translations;
WHERE
     WR1: SIZEOF(string_values) = SIZEOF(languages.language_codes);
END_ENTITY; -- translated_string_value
(*
```

Attribute definitions:

string_values: the list of **string_value**s that represent the string in various languages.

languages: the list of languages in which the same string is represented as a **string_value**.

Formal propositions:

WR1: the number of **string_value**s contained in the **string_values** list shall be equal to the number of languages provided in the **languages.language_codes** attribute.

Informal propositions:

IP1: the content of **string_values[i]** is in the language identified by **languages.language_codes[i]**.

IP2: within a supplier library, all the **translated_string_value**s shall refer to **present_translations** that are value equal: same list of languages in the same order. Only in a user library that integrates libraries from different sources there may exist different **present_translations** entities with different values.

6.3.16 String_value

A **string_value** is a value belonging to the EXPRESS STRING type.

EXPRESS specification:

```
*)
TYPE string_value = STRING;
END_TYPE; -- string_value
(*
```

6.3.17 Null_or_translatable_string_value

A **null_or_translatable_string_value** is a value belonging to the EXPRESS STRING type, included the **null_value**.

EXPRESS specification:

```
*)
TYPE null_or_translatable_string_value = SELECT(
     null_value,
     translatable_string_value);
END_TYPE; -- null_or_translatable_string_value
(*
```

6.3.18 Complex_value

A **complex_value** is any value that can be represented as an entity instance.

EXPRESS specification:

```
*)
TYPE complex_value = SELECT(
     entity_instance_value,
     level_spec_value,
     dic_class_instance);
END_TYPE; -- complex_value
(*
```


6.3.19 Null_or_complex_value

A **null_or_complex_value** is any value that can be represented as an entity instance, included the **null_value**.

EXPRESS specification:

```
*)
TYPE null_or_complex_value = SELECT(
     null_value,
     complex_value);
END_TYPE; -- null_or_complex_value
(*
```

6.3.20 Entity_instance_value

An **entity_instance_value** is a value that is an instance of some EXPRESS ENTITY data type.

This part of ISO 13584 makes distinctions among three kinds of references to EXPRESS ENTITY data types.

a) **defined_entity_instance_value** enables reference to instances of EXPRESS ENTITY data types that are explicitly referenced as possible property data types in the **ISO13584_IEC61360_dictionary_schema**.

b) **controlled_entity_instance_value** enables reference to either:

   1) instances of EXPRESS ENTITY data types that are explicitly referenced as possible property data type in the **ISO13584_extended_dictionary_schema**, or

   2) instances of EXPRESS ENTITY data types that are defined in ISO 10303-41.

NOTE 1 The ISO 10303-defined EXPRESS ENTITY data types considered in the **controlled_entity_instance_value** select type definition are those entities that are considered as possibly relevant in the development process of the view exchange protocol series of parts of ISO 13584.

NOTE 2 The **compatible_type_and_value** function enables to check whether a **controlled_entity_instance_value** is type compatible with the EXPRESS ENTITY data type that defines the data type of the property to which the **controlled_entity_instance_value** is associated in a **property_value** couple.

NOTE 3 **compatible_type_and_value** function and **property_value** entity are defined in this clause of ISO 13584-24.

c) an **uncontrolled_entity_instance_value** enables reference to any other EXPRESS ENTITY data types.

NOTE 4 The data type of such instances cannot be checked by the **compatible_type_and_value** function defined in this clause of ISO 13584-24.

NOTE 5 Each view exchange protocol shall specify which **entity_instance_type**s (and subtypes) and which **entity_instance_value**s are allowed in a library exchange context that conforms to this view exchange protocol.

EXPRESS specification:

```
*)
TYPE entity_instance_value = SELECT(
     defined_entity_instance_value,
     controlled_entity_instance_value,
     uncontrolled_entity_instance_value);
END_TYPE; -- entity_instance_value
(*
```

6.3.21 Null_or_entity_instance_value

A **null_or_entity_instance_value** is a value that is an instance of some EXPRESS ENTITY data types, included the **null_value**.

EXPRESS specification:

```
*)
TYPE null_or_entity_instance_value = SELECT(
     null_value,
     entity_instance_value);
END_TYPE; -- null_or_entity_instance_value
(*
```

6.3.22 Defined_entity_instance_value

A **defined_entity_instance_value** is an instance of one of the EXPRESS ENTITY data types that are explicitly referenced as possible property data types in the **ISO13584_IEC61360_dictionary_schema**.

NOTE 1 These instances are completely type controllable in the framework defined in the **ISO13584_instance_resource_schema**.

NOTE 2 The **ISO13584_IEC61360_dictionary_schema** is defined in IEC 61360-2 and duplicated for convenience in informative annex D of ISO 13584-42.

EXPRESS specification:

```
*)
TYPE defined_entity_instance_value = SELECT(
     placement,
     axis1_placement,
     axis2_placement_2d,
     axis2_placement_3d);
END_TYPE; -- defined_entity_instance_value
(*
```

6.3.23 Controlled_entity_instance_value

A **controlled_entity_instance_value** is an instance of one of the EXPRESS ENTITY data types that are explicitly referenced as possible property data types in the **ISO13584_extended_dictionary_schema**, or one of the EXPRESS ENTITY data types that are defined in ISO 10303-41.

NOTE These instances are completely type controllable in the framework defined in the **ISO13584_instance_resource_schema**.

EXPRESS specification:

```
*)
TYPE controlled_entity_instance_value = SELECT(
     STEP_entity_instance_value,
     PLIB_entity_instance_value);
END_TYPE; -- controlled_entity_instance_value
(*
```

6.3.24 STEP_entity_instance_value

A **STEP_entity_instance_value** entity is an instance of one of the EXPRESS ENTITY data types defined in ISO 10303 that is considered as possibly relevant in the development process of the view exchange protocol series of parts of ISO 13584.

EXPRESS specification:

```
*)
TYPE STEP_entity_instance_value = SELECT(
     product_category,
     product,
     product_definition,
     product_definition_formation,
     property_definition,
     person_organization_select,
     representation,
     representation_context,
     geometric_representation_context,
     representation_item,
     date,
     date_and_time,
     local_time,
     calendar_date,
     ordinal_date,
     week_of_year_and_day_date,
     person,
     organization,
     person_and_organization,
     address,
     personal_address,
     organizational_address );
END_TYPE; -- STEP_entity_instance_value
(*
```

6.3.25 PLIB_entity_instance_value

A **PLIB_entity_instance_value** entity is an instance of one of the EXPRESS ENTITY data types that are explicitly referenced as possible property data types in the **ISO13584_extended_dictionary_schema**.

EXPRESS specification:

```
*)
TYPE PLIB_entity_instance_value = SELECT(
     program_reference,
     representation_reference,
     property_value_external_item);
END_TYPE; -- PLIB_entity_instance_value
(*
```

6.3.26 Uncontrolled_entity_instance_value

An **uncontrolled_entity_instance_value** entity is an instance of any other EXPRESS ENTITY data types.

NOTE These instances are not type controllable in the framework defined in the **ISO13584_instance_resource_schema**.

EXPRESS specification:

```
*)
ENTITY uncontrolled_entity_instance_value
ABSTRACT SUPERTYPE;
END_ENTITY; -- uncontrolled_entity_instance_value
(*
```

6.3.27 Property_or_data_type_BSU

A **property_or_data_type_BSU** is a select type used for type control. It is either a **property_BSU** or a **data_type_BSU**.

EXPRESS specification:

```
*)
TYPE property_or_data_type_BSU = SELECT(
     property_BSU,
     data_type_BSU);
END_TYPE; -- property_or_data_type_BSU
(*
```

6.4 ISO13584_instance_resource_schema entity definitions

This clause defines the entities in the **ISO13584_instance_resource_schema**.

6.4.1 Level_spec_value

level_spec_value is an ARRAY of four optional numbers, that carries the additional meaning:

— the first value corresponds to the minimum value of some property;

— the second value corresponds to the nominal value of some property;

— the third value corresponds to the typical value of some property;

— the fourth value corresponds to the maximum value of some property.

NOTE The four numbers shall be either INTEGER or REAL.

EXPRESS specification:

```
*)
ENTITY level_spec_value
ABSTRACT SUPERTYPE OF(ONEOF(
     int_level_spec_value,
     real_level_spec_value));
     values: ARRAY [1:4] OF OPTIONAL NUMBER;
END_ENTITY; -- level_spec_value
(*
```

Attribute definitions:

values: an array of four optional numbers that represent the values associated with the described entity.

6.4.2 Null_or_level_spec_value

A **null_or_level_spec_value** is a value that is either a **level_spec_value** or a **null_value**.

EXPRESS specification:

```
*)
TYPE null_or_level_spec_value = SELECT(
     null_value,
     level_spec_value);
END_TYPE; -- null_or_level_spec_value
(*
```

6.4.3 Int_level_spec_value

A **int_level_spec_value** is a **level_spec_value** whose values are INTEGER.

EXPRESS specification:

```
*)
ENTITY int_level_spec_value
SUBTYPE OF(level_spec_value);
     SELF\level_spec_value.values: ARRAY [1:4] OF OPTIONAL INTEGER;
END_ENTITY; -- int_level_spec_value
(*
```

Attribute definitions:

values: an inherited attribute specialised to contain integer elements only.

6.4.4 Null_or_int_level_spec_value

A **null_or_int_level_spec_value** is the **null_value** or a **level_spec_value** whose values are INTEGER.

EXPRESS specification:

```
*)
TYPE null_or_int_level_spec_value = SELECT(
     null_value,
     int_level_spec_value);
END_TYPE; -- null_or_int_level_spec_value
(*
```

6.4.5 Real_level_spec_value

A **real_level_spec_value** is a **level_spec_value** whose values are REAL.

EXPRESS specification:

```
*)
ENTITY real_level_spec_value
SUBTYPE OF(level_spec_value);
     SELF\level_spec_value.values: ARRAY [1:4] OF OPTIONAL REAL;
END_ENTITY; -- real_level_spec_value
(*
```

Attribute definitions:

values: an inherited attribute specialised to contain real elements only.

6.4.6 Null_or_real_level_spec_value

A **null_or_real_level_spec_value** is the **null_value** or a **level_spec_value** whose values are REAL.

EXPRESS specification:

```
*)
TYPE null_or_real_level_spec_value = SELECT(
     null_value,
     real_level_spec_value);
END_TYPE; -- null_or_real_level_spec_value
(*
```

6.4.7 Class instances

A class instance is a value that is an instance of a class described by means of the EXPRESS resource constructs defined in the ISO 13584 Standard series for modelling classes of general models, classes of functional models or classes of functional views.

NOTE 1 The EXPRESS resource constructs for modelling classes of general models, classes of functional models or classes of functional views are defined in the **ISO13584_IEC61360_dictionary_schema**, **ISO13584_extended_dictionary_schema** and **ISO13584_library_content_schema**.

NOTE 2 The **ISO13584_IEC61360_dictionary_schema** is defined in IEC 61360-2 and duplicated for convenience in informative annex D of ISO 13584-42: 1998. The **ISO13584_extended_dictionary_schema** and the **ISO13584_library_content_schema** are defined in this part of ISO 13584.

NOTE 3 A class instance refers, through its **class_def** attribute, to the class it belongs to, and, through its **properties** attribute, to a set of **property_value**s. According to the nature of the **class** associated with the **class_BSU**, constraints specify which property values shall be provided.

NOTE 4 The properties that are to be represented in a class instance entity may be further specified in a view exchange protocol.

6.4.7.1 Dic_class_instance

A **dic_class_instance** is an instance of a class as defined by the **ISO13584_IEC61360_dictionary_schema** or by the **ISO13584_extended_dictionary_schema**. This instance refers, through its **class_BSU**, to the class to which it belongs. It may refer, through its **case_of** attributes, to classes that define partial specifications to which the instance also conforms. The list of **properties** characterises the instance within its class. The list order provides a default order for displaying this instance.

EXAMPLE A capacitor defined as a member of its class "my-capacitor" by supplier "J. Dogs and Sons" might be a "fixed capacitor" as defined by class AAA021-002 of IEC 61360-4. In this case, the instance of a "my-capacitor" class might refer through its **case_of** attribute to the AAA021-002 class of IEC 61360-4.

NOTE 1 The above example corresponds to the case where the **dic_class_instance** is also a **dic_component_instance**.

NOTE 2 When a class is only specified by a dictionary element, any property applicable to this class may be used to describe the class instance.

NOTE 3 When a class is also specified by a library specification, further constraints defined as global rules apply to the properties that are used to characterise the class instance (see **lib_item_instance**, **lib_f_model_instance**).

EXPRESS specification:

```
*)
ENTITY dic_class_instance
ABSTRACT SUPERTYPE OF(ONEOF(dic_item_instance,
          dic_f_model_instance, dic_f_view_instance));
     class_def: class_BSU;
     properties: LIST [0:?] OF property_value;
     case_of: SET [0:?] OF class_BSU;
WHERE
     WR1: (QUERY(prop <* SELF.properties |
          NOT((applicable_properties(
          SELF.class_def, [prop.prop_def])))) = [ ]);
     WR2: QUERY(prop <* SELF.properties
          | (SIZEOF(QUERY (prop1 <* SELF.properties
          | prop1.prop_def = prop.prop_def)) = 1))
          = SELF.properties;
     WR3: check_property_values_translations(QUERY(prop_val <*
          properties | 'ISO13584_INSTANCE_RESOURCE_SCHEMA.' +
          'TRANSLATED_STRING_VALUE' IN TYPEOF(prop_val.its_value)));
END_ENTITY; -- dic_class_instance
(*
```


Attribute definitions:

class_def: the class for which the current entity is an instance.

properties: the list of **property_value**s for the properties of the class instance.

case_of: the classes that define specifications to which the instance conforms.

Formal propositions:

WR1: all the properties referenced in the **properties** list shall be **applicable_properties** for the referenced **class**.

WR2: the **property_BSU**s referenced in the **properties** list shall be different from each other.

WR3: all the **property_value**s of which values are **translated_string_value**s shall be translated in the same language(s).

6.4.7.2 Null_or_dic_class_instance

A **null_or_dic_class_instance** is a **null_value** or an instance of a class as defined by the **ISO13584_IEC61360_dictionary_schema** or by the **ISO13584_extended_dictionary_schema**.

EXPRESS specification:

```
*)
TYPE null_or_dic_class_instance = SELECT(
     null_value,
     dic_class_instance);
END_TYPE; -- null_or_dic_class_instance
(*
```

6.4.7.3 Dic_item_instance

A **dic_item_instance** entity is an instance of a class defined in a **ISO13584_IEC61360_dictionary_schema** conformant dictionary as an **item_class**. The nature of the **item_class** is defined by subtyping.

EXPRESS specification:

```
*)
ENTITY dic_item_instance
SUPERTYPE OF(ONEOF(dic_component_instance,
     dic_material_instance,
     dic_feature_instance) ANDOR lib_item_instance)
SUBTYPE OF(dic_class_instance);
WHERE
     WR1: check_class_type_for_dic_item_instance(SELF);
     WR2: QUERY(prop <* SELF.properties
          | (SIZEOF(prop.prop_def.definition) = 1)
          AND (('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
          '.DEPENDENT_P_DET')
          IN TYPEOF(prop.prop_def.definition[1]))
```



```
          AND (prop.prop_def.definition[1]\dependent_P_DET.depends_on
          >= collects_property_context(prop.prop_def, SELF)))
          = [];
     WR3: compatible_item_caseof_with_class_definition(SELF);
END_ENTITY; -- dic_item_instance
(*
```

Formal propositions:

WR1: the instance class shall be defined as **item_class**.

WR2: each context parameter of the set of context parameters of which every context-dependent property depends, as defined in its dictionary definition, shall be provided either as a global context parameter or in the **context** attribute of the **context_dependent_property_value** where the context-dependent property value is defined.

WR3: if the **dic_item_instance** refers to classes that define the specifications to which it conforms through its inherited **case_of** attribute, these classes must be compatible with the set of case-of classes defined in the **class_def** class definition.

6.4.7.4 Dic_component_instance

A **dic_component_instance** entity is an instance of a class defined as a **component_class** in a dictionary compliant with the **ISO13584_IEC61360_dictionary_schema**.

NOTE A **component_class** describes a family of parts intended to be used in different products.

EXPRESS specification:

```
*)
ENTITY dic_component_instance
SUPERTYPE OF(lib_component_instance)
SUBTYPE OF(dic_item_instance);
WHERE
     WR1: check_class_type_for_dic_item_instance(SELF);
END_ENTITY; -- dic_component_instance
(*
```

Formal propositions:

WR1: the class to which the instance belongs shall be defined as a **component_class**.

6.4.7.5 Dic_material_instance

A **dic_material_instance** entity is an instance of a class defined as a **material_class** in a dictionary compliant with the **ISO13584_IEC61360_dictionary_schema**.

EXPRESS specification:

```
*)
ENTITY dic_material_instance
SUPERTYPE OF(lib_material_instance)
SUBTYPE OF(dic_item_instance);
```



```
WHERE
     WR1: check_class_type_for_dic_item_instance(SELF);
END_ENTITY; -- dic_material_instance
(*
```

Formal propositions:

WR1: the class to which the instance belongs shall be defined as a **material_class**.

6.4.7.6 Dic_feature_instance

A **dic_feature_instance** entity is an instance of a class defined as a **feature_class** in a dictionary compliant with the **ISO13584_extended_dictionary_schema**.

EXPRESS specification:

```
*)
ENTITY dic_feature_instance
SUPERTYPE OF(lib_feature_instance)
SUBTYPE OF(dic_item_instance);
WHERE
     WR1: check_class_type_for_dic_item_instance(SELF);
END_ENTITY; -- dic_feature_instance
(*
```

Formal propositions:

WR1: the class to which the instance belongs shall be defined as a **feature_class**.

6.4.7.7 Lib_item_instance

A **lib_item_instance** entity is a class instance that is both defined by a dictionary element specified as an **item_class**, and a library specification specified either as an **item_class_extension** or as an **explicit_item_class_extension**. The **property_value**s contained in a **lib_item_instance** shall contain all the identification properties defined by the item class extension. It may contain the **supplier_identification** and **supplier_designation** that provide a human-readable identification of the class instance as defined by the class supplier. It may also contain a **user_identification** and a **user_designation** of the class instance. It also contains an **is_global_id** attribute that specifies whether the class items are identified globally, or they are identified through their constituent components. Interpretation of these attributes shall be as follows. When **is_global_id** is FALSE, the item is an assembly. Its human-readable identification shall consists of its **supplier_identification** if any, followed by the set of identifications of its constituent components computed recursively until those components for which **is_global_id** is TRUE. When **is_global_id** is TRUE, the item is identified on its own. If present, **supplier_identification** contains enough information for identifying unambiguously the item, whether it is a component or a sub-system.

NOTE 1 In electronic commerce, the above specifies what pieces of information need to be exchanged when ordering a component: If **is_global_id** is TRUE, the **supplier_identification** value, if it exists, completely identify the item. If **is_global_id** is FALSE, the set of **supplier_identification** values of the item and of all its constituent components until those for which **is_global_id** equals TRUE are needed to completely identifies the assembly. This corresponds to a bill-of-material-like identification. When **is_global_id** equals TRUE and **supplier_identification** does not exist, no human-readable identification string is known.

NOTE 2 The mechanism used to associate **user_identification** and **user_description** to an **item_class** instance is outside the scope of this International Standard.

EXPRESS specification:

```
*)
ENTITY lib_item_instance
SUPERTYPE OF(ONEOF(lib_component_instance,
     lib_material_instance, lib_feature_instance))
SUBTYPE OF(dic_item_instance);
     supplier_identification: OPTIONAL STRING;
     supplier_designation: OPTIONAL translatable_label;
     user_identification: OPTIONAL STRING;
     user_designation: OPTIONAL translatable_label;
     is_global_id: BOOLEAN;
     source_class_content: OPTIONAL version_type;
END_ENTITY; -- lib_item_instance
(*
```

Attribute definitions:

supplier_identification: the OPTIONAL STRING that specifies completely or partially the item identification defined by the library data supplier.

NOTE 3 An assembly has no identification of its own.

supplier_designation: the OPTIONAL **translatable_label** that specifies completely or partially the item designation defined by the library data supplier.

NOTE 4 An assembly has no designation of its own.

user_identification: the OPTIONAL STRING that specifies the item identification defined by the library user.

user_designation: the OPTIONAL **translatable_label** that specifies the item designation defined by the library user.

NOTE 5 The capability of a library end-user to assign **user_identification** and **user_designation** to the different parts of its integrated library is implementation dependent.

is_global_id: a Boolean value that specifies whether the instance is identified by the **supplier_identification** attribute alone, or it is also identified by its constituent components.

source_class_content: the version number that characterises the extension of the class from which the instance was instantiated.

NOTE 6 The **content_version** attribute of a class will change less often than the class **version**. If the user only records one class version per **content_version** value, e. g., the latest one, the **source_class_content** attribute allows to know which class versions should be used to be able to instanciate again this instance if it was created with a previous version.

NOTE 7 This attribute is meaningless and should not exist when a library data supplier explicitly describes a class extension by means of **lib_item_instance**.

Informal propositions:

IP12: the values of the instance properties shall belong to the allowed set of values defined in the **item_class_extension** or in the **explicit_item_class_extension**.

6.4.7.8 Lib_component_instance

A **lib_component_instance** entity is an instance of a **component_class** that is associated with a library specification captured as an **item_class_extension**.

EXPRESS specification:

```
*)
ENTITY lib_component_instance
SUBTYPE OF(dic_component_instance, lib_item_instance);
END_ENTITY; -- lib_component_instance
(*
```

6.4.7.9 Lib_material_instance

A **lib_material_instance** entity is an instance of a **material_class** that is associated with a library specification captured as an **item_class_extension**.

EXPRESS specification:

```
*)
ENTITY lib_material_instance
SUBTYPE OF(dic_material_instance, lib_item_instance);
END_ENTITY; -- lib_material_instance
(*
```

6.4.7.10 Lib_feature_instance

A **lib_feature_instance** entity is an instance of a **feature_class** that is associated with a library specification captured as an **item_class_extension**.

EXPRESS specification:

```
*)
ENTITY lib_feature_instance
SUBTYPE OF(dic_feature_instance, lib_item_instance);
END_ENTITY; -- lib_feature_instance
(*
```

6.4.7.11 Dic_f_model_instance

A **dic_f_model_instance** entity is an instance of a class defined in a **ISO13584_extended_dictionary_schema**-conformant dictionary as a **functional_model_class**.

EXPRESS specification:

```
*)
ENTITY dic_f_model_instance
SUPERTYPE OF(lib_f_model_instance)
SUBTYPE OF(dic_class_instance);
WHERE
     WR1: check_class_type_for_dic_f_model_instance(SELF);
```



```
      WR2: compatible_model_caseof_with_class_definition(SELF);
END_ENTITY; -- dic_f_model_instance
(*
```

Formal propositions:

WR1: the instance class shall be defined as **functional_model_class**.

WR2: if the **dic_f_model_instance** refers to classes that define the specifications to which it conforms through its inherited **case_of** attribute, these classes must be compatible with the set of case-of classes defined in the **class_def** class definition.

6.4.7.12 Lib_f_model_instance

A **lib_f_model_instance** entity is an instance of a **functional_model_class** that is associated with a library specification captured as a **functional_model_class_extension**. The **property_value**s contained in a **lib_f_model_instance** shall contain all the free properties that are defined by the **functional_model_class_extension**.

EXPRESS specification:

```
*)
ENTITY lib_f_model_instance
SUBTYPE OF(dic_f_model_instance);
END_ENTITY; -- lib_f_model_instance
(*
```

Informal propositions:

IP1: the set of values of the instance properties shall belong to the allowed set of values defined in the **functional_model_class_extension**.

6.4.7.13 Dic_f_view_instance

A **dic_f_view_instance** is an instance of a **functional_view_class** conformant with the **ISO13584_extended_dictionary_schema**. This instance refers, through its inherited **class_def** attribute, to the **functional_view_class** to which it belongs. The **property_value**s referenced in a **dic_f_view_instance** are either:

— view control variable values that specify the view level within the representation category the view belongs to and shall always be provided, or

— view properties that define the specific properties of the view.

A **dic_f_view_instance** contains OPTIONAL attributes that specify the **dic_f_model_instance** that produced the view, and the **dic_item_instance** represented by the view.

A **dic_f_view_instance** is a subtype of an ISO 10303-43 **representation**. Therefore, it contains two inherited attributes:

— a **context_of_items** attribute, that contains the **representation_context** that defines the context of the **representation_item** elements that constitute the view, and

— an **items** attribute that contains a set of **representation_item**s.

EXPRESS specification:

```
*)
ENTITY dic_f_view_instance
SUBTYPE OF(dic_class_instance, representation);
      generated_by: OPTIONAL dic_f_model_instance;
      view_of: OPTIONAL dic_item_instance;
DERIVE
      SELF\dic_class_instance.case_of: SET OF class_BSU := [];
WHERE
      WR1: check_class_type_for_dic_f_view_instance(SELF);
      WR2: NOT all_class_descriptions_reachable(
           SELF\dic_class_instance.class_def)
           OR (QUERY(prop <* functional_view_v_c_v(
           SELF\dic_class_instance.class_def)
           | SIZEOF(QUERY(prop2 <* SELF.properties
           | prop2.prop_def = prop))<>1) = []);
      WR3: correct_view_from_model(SELF);
END_ENTITY; -- dic_f_view_instance
(*
```

Attribute definitions:

generated_by: the OPTIONAL **dic_f_model_instance** that produced the view.

view_of: the OPTIONAL **dic_item_instance** to which the **representation** defined by the **dic_f_view_instance** is associated.

Formal propositions:

WR1: the instance class shall be defined as a **functional_view_class**.

WR2: all the view control variables of the **functional_view_class**, as returned by the **functional_view_v_c_v** function, shall be represented in the **properties** set.

NOTE When some of the view control variables are not required to specify the created view, some of the **property_value**s corresponding to the view control variables may not contain any value in their **its_value** attribute.

WR3: the **generated_by functional_model_class** shall be able to create the **class_def** functional view.

6.4.8 Property_value

A **property_value** is the value associated with a particular property of some **dic_class_instance**. If the **its_value** attribute does not exist, this property has no value.

NOTE The properties associated with a **dic_class_instance** through **property_value** are the properties that *define* the instance. When inserted in product model data, other properties may be associated with this instance according to the requirements and information model of this product model data.

EXPRESS specification:

```
*)
ENTITY property_value;
     its_value: OPTIONAL primitive_value;
     prop_def: property_BSU;
WHERE
     WR1: (EXISTS(SELF.its_value) AND (compatible_type_and_value(
          SELF.prop_def, SELF.its_value)))
          OR NOT EXISTS(SELF.its_value);
END_ENTITY; -- property_value
(*
```

Attribute definitions:

its_value: the value associated with the property.

prop_def: the property that describes the instance property to which the **its_value** refers.

Formal propositions:

WR1: the value of the property, if it exists, shall be type compatible with the referenced property.

6.4.9 Context_dependent_property_value

A **context_dependent_property_value** allows to associate with the value of a context dependent characteristics (i.e., **dependent_P_DET**) values of context parameters (i.e., values of **condition_DET**) that specify all or part of the dependent characteristics measure.

NOTE When describing a part instance by a set of property values, it is not possible to represent that a context dependent characteristics (i.e., **dependent_P_DET**) depends on one particular context (i.e., values of **condition_DET**s). In fact, it is a current practice in electronic to have different values of the same context parameters (i.e., values of **condition_DET**s) that define the context of different context dependent characteristics (i.e., **dependent_P_DET**).

EXPRESS specification:

```
*)
ENTITY context_dependent_property_value
SUBTYPE OF(property_value);
     the_context: LIST[1:?] OF property_value;
WHERE
     WR1: QUERY(c <* SELF.the_context | NOT(is_condition_det(c)))
          = [];
     WR2: is_dependent_p_det(SELF\property_value.prop_def);
     WR3: all_context_parameters_referenced(SELF);
END_ENTITY; -- context_dependent_property_value
(*
```

Attribute definitions:

the_context: the list of context parameter values that specifies all or part of the context in which the **SELF\property_value.its_value** context dependent property value is valid; the list order providing a default display order.

Formal propositions:

WR1: every property referenced in the **SELF.the_context** set of **property_value**s shall be a **condition_DET**.

WR2: the **SELF\property_value.prop_def** property shall be a **dependent_P_DET**.

WR3: the set of properties referenced in the **SELF.the_context** set of **property_value**s shall be included in the set of context parameter of which the **SELF\property_value.prop_def** property depends as defined in its dictionary definition.

6.5 ISO13584_instance_resource_schema rule definition

6.5.1 Valued_properties_are_allowed_for_implicit_spec_rule rule

The **valued_properties_are_allowed_for_implicit_spec_rule** rule checks that when a PLIB conformant exchange context includes **lib_item_instance** and **item_class_extension**, and when **lib_item_instance**s reference such an available **item_class_extension**, these **lib_item_instance**s are described by a set of properties that are allowed, as returned by the **allowed_properties** function.

EXPRESS specification:

```
*)
RULE valued_properties_are_allowed_for_implicit_spec_rule FOR
     (lib_item_instance, item_class_extension);
LOCAL
     allowed_valued_properties: LOGICAL := TRUE;
END_LOCAL;

REPEAT i := 1 TO SIZEOF(lib_item_instance);
     IF (SIZEOF(lib_item_instance[i]\dic_class_instance.class_def
          .referenced_by) = 1)
     THEN
          IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA' +
               '.ITEM_CLASS_EXTENSION' IN TYPEOF(lib_item_instance[i]\
               dic_class_instance.class_def.referenced_by[1]))
          THEN
               allowed_valued_properties := allowed_valued_properties
               AND (QUERY(prop <* lib_item_instance[i].properties |
               NOT((allowed_properties(lib_item_instance[i]\
               dic_class_instance.class_def, [prop.prop_def])))) = []);
          END_IF;
     END_IF;
END_REPEAT;

WHERE
     WR1: allowed_valued_properties;
```



```
END_RULE; -- valued_properties_are_allowed_for_implicit_spec_rule
(*
```

6.5.2 Valued_properties_are_allowed_for_explicit_spec_rule rule

The **valued_properties_are_allowed_for_explicit_spec_rule** rule checks that when a PLIB conformant exchange context includes **lib_item_instance** and **explicit_item_class_extension**, and when **lib_item_instance**s reference such an available **explicit_item_class_extension**, these **lib_item_instance**s are described by a set of properties that are applicable for the referenced **explicit_item_class_extension**.

EXPRESS specification:

```
*)
RULE valued_properties_are_allowed_for_explicit_spec_rule FOR
     (lib_item_instance, explicit_item_class_extension);
LOCAL
     allowed_valued_properties: LOGICAL := TRUE;
END_LOCAL;

REPEAT i := 1 TO SIZEOF(lib_item_instance);
     IF SIZEOF(lib_item_instance[i]\dic_class_instance.class_def
          .referenced_by) = 1
     THEN
          IF ('ISO13584_LIBRARY_CONTENT_SCHEMA' +
               '.EXPLICIT_ITEM_CLASS_EXTENSION' IN
               TYPEOF(lib_item_instance[i]\dic_class_instance.
               class_def.referenced_by[1]))
          THEN
               allowed_valued_properties := allowed_valued_properties
               AND (QUERY(prop <* lib_item_instance[i].properties
               | NOT((applicable_properties(
               lib_item_instance[i]\dic_class_instance
               .class_def, [prop.prop_def])))) = []);
          END_IF;
     END_IF;
END_REPEAT;

WHERE
     WR1: allowed_valued_properties;
END_RULE; -- valued_properties_are_allowed_for_explicit_spec_rule
(*
```

6.5.3 Identification_properties_are_valued_for_implicit_spec_rule rule

The **identification_properties_are_valued_for_implicit_spec_rule** rule checks that when a PLIB conformant exchange context includes **lib_item_instance** and **item_class_extension**, and when **lib_item_instance**s reference such an available **item_class_extension**, these **lib_item_instance**s are at least described by the set of identification characteristics as returned by the **gm_identification_characteristics_list** function.

EXPRESS specification:

```
*)
RULE identification_properties_are_valued_for_implicit_spec_rule FOR
     (lib_item_instance, item_class_extension);
LOCAL
     valued_identification_properties: LOGICAL := TRUE;
END_LOCAL;

REPEAT i := 1 TO SIZEOF(lib_item_instance);
     IF (SIZEOF(lib_item_instance[i]\dic_class_instance.
          class_def.referenced_by) = 1)
     THEN
          IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA' +
               '.ITEM_CLASS_EXTENSION' IN
               TYPEOF(lib_item_instance[i]\dic_class_instance.
               class_def.referenced_by[1]))
          THEN
               valued_identification_properties :=
               valued_identification_properties
               AND (NOT all_class_descriptions_reachable(
               lib_item_instance[i]\dic_class_instance.class_def)
               OR (QUERY(prop <*
               gm_identification_characteristics_list(
               lib_item_instance[i]\dic_class_instance.class_def)
               | NOT(prop IN collects_assigned_instance_properties
               (list_to_set(lib_item_instance[i]\
               dic_class_instance.properties)))) = []));
          END_IF;
     END_IF;
END_REPEAT;

WHERE
     WR1: valued_identification_properties;
END_RULE;-- identification_properties_are_valued_for_implicit_spec_rule
(*
```

6.5.4 Identification_properties_are_valued_for_explicit_spec_rule rule

The **identification_properties_are_valued_for_explicit_spec_rule** rule checks that when a PLIB conformant exchange context includes **lib_item_instance** and **explicit_item_class_extension**, and when **lib_item_instance**s reference such an available **explicit_item_class_extension**, these **lib_item_instance**s are at least described by the set of **explicit_item_class_extension instance_identification** properties.

EXPRESS specification:

```
*)
RULE identification_properties_are_valued_for_explicit_spec_rule FOR
     (lib_item_instance, explicit_item_class_extension);
LOCAL
     valued_identification_properties: LOGICAL := TRUE;
```



```
END_LOCAL;

REPEAT i := 1 TO SIZEOF(lib_item_instance);
     IF SIZEOF(lib_item_instance[i]\dic_class_instance.
          class_def.referenced_by) = 1
     THEN
          IF ('ISO13584_LIBRARY_CONTENT_SCHEMA' +
               '.EXPLICIT_ITEM_CLASS_EXTENSION' IN
               TYPEOF(lib_item_instance[i]\dic_class_instance.
               class_def.referenced_by[1]))
          THEN
               valued_identification_properties :=
               valued_identification_properties
               AND (QUERY(prop <*
               lib_item_instance[i]\dic_class_instance.
               class_def.referenced_by[1].instance_identification
               | NOT(prop IN collects_assigned_instance_properties
               (list_to_set(lib_item_instance[i]\
               dic_class_instance.properties)))) = []);
          END_IF;
     END_IF;
END_REPEAT;

WHERE
     WR1: valued_identification_properties;
END_RULE; --identification_properties_are_valued_for_explicit_spec_rule
(*
```

6.5.5 Fm_valued_properties_are_allowed_for_implicit_spec_rule rule

The **fm_valued_properties_are_allowed_for_implicit_spec_rule** rule checks that when a PLIB conformant exchange context includes **lib_f_model_instance** and **functional_model_class_extension**, and when **lib_f_model_instance**s reference such an available **functional_model_class_extension**, these **lib_f_model_instance**s are described by a set of properties that are allowed, as returned by the **allowed_properties** function.

EXPRESS specification:

```
*)
RULE fm_valued_properties_are_allowed_for_implicit_spec_rule FOR
     (lib_f_model_instance, functional_model_class_extension);
LOCAL
     allowed_valued_properties: LOGICAL := TRUE;
END_LOCAL;

REPEAT i := 1 TO SIZEOF(lib_f_model_instance);
     IF (SIZEOF(lib_f_model_instance[i]\dic_class_instance.
          class_def.referenced_by) = 1)
     THEN
          IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA' +
               '.FUNCTIONAL_MODEL_CLASS_EXTENSION' IN
               TYPEOF(lib_f_model_instance[i]\dic_class_instance.
```

```
                class_def.referenced_by[1]))
            THEN
                allowed_valued_properties := allowed_valued_properties
                AND (QUERY(prop <* lib_f_model_instance[i].properties
                | NOT((allowed_properties(lib_f_model_instance[i]\
                dic_class_instance.class_def, [prop.prop_def])))) = []);
            END_IF;
        END_IF;
    END_REPEAT;

    WHERE
        WR1: allowed_valued_properties;
    END_RULE; -- fm_valued_properties_are_allowed_for_implicit_spec_rule
    (*
```

6.5.6 Fm_valued_properties_are_allowed_for_explicit_spec_rule rule

The **fm_valued_properties_are_allowed_for_explicit_spec_rule** rule checks that when a PLIB conformant exchange context includes **lib_f_model_instance** and **explicit_functional_model_class_extension**, and when **lib_f_model_instance**s reference such an available **explicit_functional_model_class_extension**, these **lib_f_model_instance**s are described by a set of properties that are applicable for the referenced **explicit_functional_model_class_extension**.

EXPRESS specification:

```
    *)
    RULE fm_valued_properties_are_allowed_for_explicit_spec_rule FOR(
        lib_f_model_instance,
        explicit_functional_model_class_extension);
    LOCAL
        allowed_valued_properties: LOGICAL := TRUE;
    END_LOCAL;

    REPEAT i := 1 TO SIZEOF(lib_f_model_instance);
        IF SIZEOF(lib_f_model_instance[i]\dic_class_instance.
            class_def.referenced_by) = 1
        THEN
            IF ('ISO13584_LIBRARY_CONTENT_SCHEMA' +
                '.EXPLICIT_FUNCTIONAL_MODEL_CLASS_EXTENSION' IN
                TYPEOF(lib_f_model_instance[i]\dic_class_instance.
                class_def.referenced_by[1]))
            THEN
                allowed_valued_properties := allowed_valued_properties
                AND(QUERY(prop <* lib_f_model_instance[i].properties
                | NOT((applicable_properties(
                lib_f_model_instance[i]\dic_class_instance.
                class_def, [prop.prop_def])))) = []);
            END_IF;
        END_IF;
    END_REPEAT;
```



```
WHERE
     WR1: allowed_valued_properties;
END_RULE; -- fm_valued_properties_are_allowed_for_explicit_spec_rule
(*
```

6.5.7 Fm_free_properties_are_valued_for_implicit_spec_rule rule

The **fm_free_properties_are_valued_for_implicit_spec_rule** rule checks that when a PLIB conformant exchange context includes **lib_f_model_instance** and **functional_model_class_extension**, and when **lib_f_model_instance**s reference such an available **functional_model_class_extension**, these **lib_f_model_instance**s are at least described by the set of identification characteristics as returned by the **gm_identification_characteristics_list** function.

EXPRESS specification:

```
*)
RULE fm_free_properties_are_valued_for_implicit_spec_rule FOR
     (lib_f_model_instance, functional_model_class_extension);
LOCAL
     valued_free_properties: LOGICAL := TRUE;
END_LOCAL;

REPEAT i := 1 TO SIZEOF(lib_f_model_instance);
     IF (SIZEOF(lib_f_model_instance[i]\dic_class_instance.
          class_def.referenced_by) = 1)
     THEN
          IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA' +
               '.FUNCTIONAL_MODEL_CLASS_EXTENSION' IN TYPEOF(
               lib_f_model_instance[i]\dic_class_instance.
               class_def.referenced_by[1]))
          THEN
               valued_free_properties := valued_free_properties
               AND (NOT all_class_descriptions_reachable(
               lib_f_model_instance[i]\dic_class_instance.class_def)
               OR (QUERY(prop <*
               fm_free_model_properties_list(
               lib_f_model_instance[i]\dic_class_instance.
               class_def) | NOT(prop IN
               collects_assigned_instance_properties(
               list_to_set(lib_f_model_instance[i]\
               dic_class_instance.properties)))) = []));
          END_IF;
     END_IF;
END_REPEAT;

WHERE
     WR1: valued_free_properties;
END_RULE; -- fm_free_properties_are_valued_for_implicit_spec_rule
(*
```

6.5.8 Fm_free_properties_are_valued_for_explicit_spec_rule rule

The **fm_free_properties_are_valued_for_explicit_spec_rule** rule checks that when a PLIB conformant exchange context includes **lib_f_model_instance** and **explicit_functional_model_class_extension**, and when **lib_f_model_instance**s reference such an available **explicit_functional_model_class_extension**, these **lib_f_model_instance**s are at least described by the set of **explicit_functional_model_class_extension instance_identification** properties.

EXPRESS specification:

```
*)
RULE fm_free_properties_are_valued_for_explicit_spec_rule FOR(
     lib_f_model_instance, explicit_functional_model_class_extension);
LOCAL
     valued_free_properties: LOGICAL := TRUE;
END_LOCAL;

REPEAT i := 1 TO SIZEOF(lib_f_model_instance);
     IF SIZEOF(lib_f_model_instance[i]\dic_class_instance.
          class_def.referenced_by) = 1
     THEN
          IF ('ISO13584_LIBRARY_CONTENT_SCHEMA' +
               '.EXPLICIT_FUNCTIONAL_MODEL_CLASS_EXTENSION' IN
               TYPEOF(lib_f_model_instance[i]\dic_class_instance.
               class_def.referenced_by[1]))
          THEN
               valued_free_properties := valued_free_properties
               AND (QUERY(prop <* lib_f_model_instance[i]\
               dic_class_instance.class_def.referenced_by[1].
               instance_identification |
               NOT(prop IN collects_assigned_instance_properties(
               list_to_set(lib_f_model_instance[i]\
               dic_class_instance.properties)))) = []);
          END_IF;
     END_IF;
END_REPEAT;

WHERE
     WR: valued_free_properties;
END_RULE; -- fm_free_properties_are_valued_for_explicit_spec_rule
(*
```

6.6 ISO13584_instance_resource_schema function definitions

This clause defines the functions in the **ISO13584_instance_resource_schema**. These functions enable the schema to perform type control for instances.

6.6.1 Compatible_class_and_class function

The function **compatible_class_and_class** checks if the instances of a class **cl2** are compatible with the domain defined by the class **cl1**.

An instance of class **cl2** is compatible with the domain defined by a class **cl1** if one of the following condition holds:

a) the class **cl2**:

— is defined by the same supplier as cl1, and

— has the same code as cl1, and

— has a version less than or equal to the version of cl1.

b) the domain of one superclass of the class **cl2** is compatible with the domain of a class **cl1**.

If a BSU definition required to evaluate the compatibility is not available, the function returns UNKNOWN.

NOTE This compatibility does not ensure that the set of property values that may be associated with an instance of class **cl2** belongs to the allowed set of property values for class **cl1** (a subclass may define more properties than its super-class).

EXPRESS specification:

```
*)
FUNCTION compatible_class_and_class(cl1:class_BSU;
     cl2:class_BSU): LOGICAL;

IF (cl1.defined_by\basic_semantic_unit.code =
     cl2.defined_by\basic_semantic_unit.code) AND
     (cl1\basic_semantic_unit.code =
     cl2\basic_semantic_unit.code) AND
     (cl1\basic_semantic_unit.version >=
     cl2\basic_semantic_unit.version)
THEN (* the two classes have the same identification and are version
     compatible *)
     RETURN(TRUE);
END_IF;

IF (SIZEOF(cl2\basic_semantic_unit.definition) = 0)
THEN (* the superclass of cl2 is not available *)
     RETURN(UNKNOWN);
END_IF;

IF (SIZEOF(cl2\basic_semantic_unit.definition) = 1)
     AND (NOT EXISTS(cl2\basic_semantic_unit.
     definition[1]\class.its_superclass))
THEN (* cl2 has no superclass *)
     RETURN(FALSE);
END_IF;

RETURN(compatible_class_and_class(cl1, cl2\basic_semantic_unit.
     definition[1]\class.its_superclass));

END_FUNCTION; -- compatible_class_and_class
(*
```


6.6.2 Right_values_for_level_spec function

The function **right_values_for_level_spec** checks if the existing values **val** of the **level_spec_value** are in the places defined by the **levels** parameter.

EXPRESS specification:

```
*)
FUNCTION right_values_for_level_spec(
     levels: LIST [1:4] OF UNIQUE level;
     val: level_spec_value): BOOLEAN;

LOCAL
     c_place: BOOLEAN;
     lev: SET [1:4] OF level;
END_LOCAL;

c_place := TRUE;
lev := list_to_set(levels);

IF EXISTS(val.values[1])
THEN
     IF level.min IN lev
     THEN
          lev := lev - [level.min];
     ELSE
          c_place := FALSE;
     END_IF;
END_IF;

IF EXISTS(val.values[2])
THEN
     IF level.nom IN lev
     THEN
          lev := lev - [level.nom];
     ELSE
          c_place := FALSE;
     END_IF;
END_IF;

IF EXISTS(val.values[3])
THEN
     IF level.typ IN lev
     THEN
          lev := lev - [level.typ];
     ELSE
          c_place := FALSE;
     END_IF;
END_IF;
```



```
IF EXISTS(val.values[4])
THEN
     IF level.max IN lev
     THEN
          lev := lev - [level.max];
     ELSE
          c_place := FALSE;
     END_IF;
END_IF;

IF (c_place)
THEN
     RETURN(TRUE);
ELSE
     RETURN(FALSE);
END_IF;

END_FUNCTION; -- right_values_for_level_spec
(*
```

6.6.3 Compatible_level_type_and_instance function

The function **compatible_level_type_and_instance** checks if a **level_spec_value** is compatible with the domain defined by a **level_type** specification, described in terms of:

— a LIST of unique **level**, that corresponds to the **levels** attribute of the **level_type**, and

— a SET of STRINGs that contains the result of the TYPEOF function, applied to the **value_type** attribute of the **level_type**.

A **level_spec_value** is compatible with the domain defined by a **level_type** if the two following conditions hold:

a) the values of the **level_spec_value**:

   — either are INTEGER, and the result of the TYPEOF function applied to **level_type**.**value_type** does not contain '**REAL_TYPE**', or

   — are REAL, and the result of the TYPEOF function applied to **level_type.value_type** does not contain '**INT_TYPE**';

b) the existing values of the **level_spec_value** are in the places defined by the **levels** attribute of the **level_type**.

EXPRESS specification:

```
*)
FUNCTION compatible_level_type_and_instance(
     levels: LIST [1:4] OF UNIQUE level; value_typeof: SET OF STRING;
     val: level_spec_value): BOOLEAN;

LOCAL
     c_val: BOOLEAN;
END_LOCAL;
```



```
c_val := FALSE;

IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.INT_LEVEL_SPEC_VALUE'
      IN TYPEOF(val))
      AND NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE' IN
      value_typeof)
THEN
      c_val := TRUE;
END_IF;

IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.REAL_LEVEL_SPEC_VALUE'
      IN TYPEOF(val))
      AND NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_TYPE'
      IN value_typeof)
THEN
      c_val := TRUE;
END_IF;

RETURN(c_val AND (right_values_for_level_spec(levels, val)));

END_FUNCTION; -- compatible_level_type_and_instance
(*
```

6.6.4 Compatible_type_and_value function

The function **compatible_type_and_value** checks if a value **val** of a **primitive_value** is type compatible with the types defined by a type **dom** defined by a **property_or_data_type_BSU**. It returns a LOGICAL that is TRUE when they are compatible and FALSE when they are not. This function returns UNKNOWN when some required **basic_semantic_unit** definition is not present, or when the **val** data type is an **uncontrolled_instance_value**.

NOTE The value **val** may or may not exist.

EXPRESS specification:

```
*)
FUNCTION compatible_type_and_value(dom: property_or_data_type_BSU;
      val: primitive_value): LOGICAL;

LOCAL
      temp: SET[0:1] OF 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;

IF data_type_typeof(dom) = []
THEN (* the final domain of the type is not available *)
      RETURN(UNKNOWN);
END_IF;
```



```
(* 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 data_type_typeof(dom))
          AND (SIZEOF(data_type_non_quantitative_int_type(dom)) = 1))
     THEN
          set_integer := [];
          code_type := data_type_non_quantitative_int_type(dom)[1];

          REPEAT j := 1 TO SIZEOF(code_type.domain.its_values);
               set_integer := set_integer +
                    code_type.domain.its_values[j].value_code;
          END_REPEAT;

          RETURN(val IN set_integer);

     ELSE
          RETURN(('ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_TYPE'
               IN data_type_typeof(dom)) OR
               (('ISO13584_IEC61360_DICTIONARY_SCHEMA.NUMBER_TYPE'
               IN data_type_typeof(dom))
               AND NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE'
               IN data_type_typeof(dom))));
     END_IF;
END_IF;

IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.REAL_VALUE' IN TYPEOF(val))
THEN
     RETURN(('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE'
          IN data_type_typeof(dom)) OR
          (('ISO13584_IEC61360_DICTIONARY_SCHEMA.NUMBER_TYPE'
          IN data_type_typeof(dom))
          AND NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_TYPE'
          IN data_type_typeof(dom))));
END_IF;

IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.NUMBER_VALUE' IN TYPEOF(val))
THEN
     RETURN('ISO13584_IEC61360_DICTIONARY_SCHEMA.NUMBER_TYPE'
          IN data_type_typeof(dom));
END_IF;

IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.BOOLEAN_VALUE' IN TYPEOF(val))
THEN
     RETURN('ISO13584_IEC61360_DICTIONARY_SCHEMA.BOOLEAN_TYPE'
          IN data_type_typeof(dom));
END_IF;

IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.TRANSLATABLE_STRING_VALUE'
```

```
      IN TYPEOF(val))
THEN
      IF (('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
           '.NON_QUANTITATIVE_CODE_TYPE') IN  data_type_typeof(dom))
      THEN
           IF (SIZEOF(data_type_non_quantitative_code_type(dom)) = 1)
           THEN
                set_string := [];
                code_type :=
                     data_type_non_quantitative_code_type(dom)[1];

                REPEAT j := 1 TO SIZEOF(code_type.domain.its_values);
                     set_string := set_string +
                          code_type.domain.its_values[j].value_code;
                END_REPEAT;

                RETURN(('ISO13584_INSTANCE_RESOURCE_SCHEMA.STRING_VALUE'
                     IN TYPEOF(val)) AND (val IN set_string));

           ELSE
                RETURN(UNKNOWN);
           END_IF;
      ELSE
           RETURN('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
                '.STRING_TYPE' IN data_type_typeof(dom));
      END_IF;
END_IF;

(* The following express statements deal with complex types *)

IF 'ISO13584_INSTANCE_RESOURCE_SCHEMA.ENTITY_INSTANCE_VALUE'
      IN TYPEOF(val)
THEN
      IF 'ISO13584_INSTANCE_RESOURCE_SCHEMA' +
           '.UNCONTROLLED_ENTITY_INSTANCE_VALUE'
           IN TYPEOF(val)
      THEN
           RETURN(UNKNOWN);
      END_IF;
      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.ENTITY_INSTANCE_TYPE'
           IN data_type_typeof(dom))
        AND (SIZEOF(data_type_type_name(dom)) <> 0)
           AND (data_type_type_name(dom) <= TYPEOF(val))
      THEN
           RETURN(TRUE);
      ELSE
           RETURN(FALSE);
      END_IF;
END_IF;

IF 'ISO13584_INSTANCE_RESOURCE_SCHEMA.DIC_CLASS_INSTANCE'
```



```
      IN TYPEOF(val)
THEN
      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_INSTANCE_TYPE'
           IN data_type_typeof(dom))
           AND (SIZEOF(data_type_class_of(dom)) <> 0)
      THEN
           temp := data_type_class_of(dom);
           RETURN(compatible_class_and_class(temp[1],
                val\dic_class_instance.class_def));
      ELSE
           RETURN(FALSE);
      END_IF;
END_IF;

IF 'ISO13584_INSTANCE_RESOURCE_SCHEMA.LEVEL_SPEC_VALUE' IN TYPEOF(val)
THEN
      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.LEVEL_TYPE'
           IN data_type_typeof(dom))
      THEN
           RETURN(compatible_level_type_and_instance(
                data_type_level_spec(dom),
                data_type_level_value_typeof(dom),
                val));
      ELSE
           RETURN(FALSE);
      END_IF;
END_IF;

RETURN(FALSE);

END_FUNCTION; -- compatible_type_and_value
(*
```

6.6.5 Collects_assigned_instance_properties function

The **collects_assigned_instance_properties** function computes the properties that are referenced in its **properties** attribute of a **dic_class_instance**.

EXPRESS specification:

```
*)
FUNCTION collects_assigned_instance_properties(
      props: SET [0:?] OF property_value): SET OF property_BSU;

LOCAL
      assign_prop: SET OF property_BSU;
           -- assigned properties of the dic_class_instance
END_LOCAL;

assign_prop := [];

REPEAT i := 1 TO SIZEOF(props);
```



```
      assign_prop := assign_prop + props[i].prop_def;
END_REPEAT;

RETURN(assign_prop);

END_FUNCTION; -- collects_assigned_instance_properties
(*
```

6.6.6 Correct_view_from_model function

The **correct_view_from_model** function checks that the **generated_by functional_model_class** attribute of a **fv dic_f_view_instance** is able to create the **class_def** functional view. It returns UNKNOWN when some **dictionary_element** is not available.

EXPRESS specification:

```
*)
FUNCTION correct_view_from_model(fv: dic_f_view_instance): LOGICAL;

IF NOT EXISTS(fv.generated_by)
THEN
      RETURN(UNKNOWN);
END_IF;

IF NOT(SIZEOF(fv.generated_by\dic_class_instance.class_def.
      definition) = 1)
THEN
      RETURN(UNKNOWN);
ELSE
      RETURN(fv.generated_by\dic_class_instance.class_def
           .definition[1].created_view =
           fv\dic_class_instance.class_def);
END_IF;

END_FUNCTION; -- correct_view_from_model
(*
```

6.6.7 Is_condition_det function

The **is_condition_det** function returns TRUE if the property definition of the given **prop property_value** is a **condition_det**. Otherwise, it returns FALSE. If the property definition is not available, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION is_condition_det(prop: property_value): LOGICAL;

IF(SIZEOF(prop.prop_def.definition) > 0) THEN
      RETURN('ISO13584_IEC61360_DICTIONARY_SCHEMA.CONDITION_DET'
           IN TYPEOF(prop.prop_def.definition[1]));
ELSE
```



```
     RETURN(UNKNOWN);
END_IF;

END_FUNCTION; -- is_condition_det
(*
```

6.6.8 Is_dependent_p_det function

The **is_dependent_p_det** function returns TRUE if the property definition of the given **prop property_value** is a **dependent_p_det**. Otherwise, it returns FALSE. If the property definition is not available, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION is_dependent_p_det(prop: property_bsu): LOGICAL;

IF(SIZEOF(prop.definition) > 0) THEN
     RETURN('ISO13584_IEC61360_DICTIONARY_SCHEMA.DEPENDENT_P_DET'
          IN TYPEOF(prop.definition[1]));
ELSE
     RETURN(UNKNOWN);
END_IF;

END_FUNCTION; -- is_dependent_p_det
(*
```

6.6.9 All_context_parameters_referenced function

The **all_context_parameters_referenced** function returns TRUE if the set of **property_value**s of the **cdpv context_dependent_property_value** is included in the set of context parameters of which the **cdpv context_dependent_property_value.prop_def** property depends as defined in its dictionary definition. Otherwise, it returns FALSE.

EXPRESS specification:

```
*)
FUNCTION all_context_parameters_referenced(
     cdpv: context_dependent_property_value): LOGICAL;

IF(SIZEOF(cdpv\property_value.prop_def.definition) > 0) THEN
     RETURN(cdpv\property_value.prop_def.definition[1]\
          dependent_p_det.depends_on
          >= collects_assigned_instance_properties(
          list_to_set(cdpv.the_context)));
ELSE
     RETURN(UNKNOWN);
END_IF;

END_FUNCTION; -- all_context_parameters_referenced
(*
```

6.6.10 Collects_property_context function

The **collects_property_context** function computes the context parameters that define the context of a property **prop** that is referenced in its **properties** attribute by a **dic_class_instance** denoted **inst**. This context consists of the context parameters referenced by the **properties** attribute of a **dic_class_instance**, more, if the property **prop** is referenced by means of a **context_dependent_property_value**, the context parameters referenced by the **context** attribute of this **context_dependent_property_value**. Each property referenced in the **properties** attribute of the **inst dic_class_instance** is returned by the **collects_property_context** function if its definition is not available to decide whether it is a context parameter. The **collects_property_context** function requires that property **prop** be referenced in the **properties** attribute of the **inst dic_class_instance**, otherwise it returns the empty set.

EXPRESS specification:

```
*)
FUNCTION collects_property_context(prop: property_BSU;
     inst: dic_class_instance): SET OF property_BSU;

LOCAL
     assigned_context_parameters: SET OF property_BSU;
          --assigned context parameters of the dic_class_instance
     correct: BOOLEAN; --prop belongs to inst properties
END_LOCAL;

assigned_context_parameters := [];
correct := FALSE;

REPEAT i := 1 TO SIZEOF(inst.properties);

     IF inst.properties[i].prop_def :=: prop
     THEN
          correct := TRUE;
     END_IF;

     IF ((SIZEOF(inst.properties[i].prop_def.definition) = 0)
          OR ((SIZEOF(inst.properties[i].prop_def.definition) = 1)
          AND (('ISO13584_IEC61630_DICTIONARY_SCHEMA.CONDITION_DET')
          IN TYPEOF(inst.properties[i].prop_def.definition[1]))))
     THEN
          assigned_context_parameters := assigned_context_parameters
               + inst.properties[i].prop_def;
     END_IF;

     IF (('ISO13584_INSTANCE_RESOURCE_SCHEMA' +
          '.CONTEXT_DEPENDENT_PROPERTY_VALUE') IN
          TYPEOF(inst.properties[i]))
     THEN
          assigned_context_parameters := assigned_context_parameters
               + collects_assigned_instance_properties(list_to_set(
               inst.properties[i].the_context));
     END_IF;
```



```
END_REPEAT;

IF NOT correct
THEN
     assigned_context_parameters := [];
END_IF;

RETURN(assigned_context_parameters);

END_FUNCTION; -- collects_property_context
(*
```

6.6.11 Check_class_type_for_dic_item_instance function

The **check_class_type_for_dic_item_instance** returns TRUE if the referenced class dictionary definition is type compatible with the given **dic_cl dic_item_instance**, otherwise it returns FALSE. If this dictionary definition is not available, it returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION check_class_type_for_dic_item_instance(
     dic_cl: dic_item_instance): LOGICAL;

IF (SIZEOF(dic_cl.class_def.definition) = 1) THEN

     IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.' +
          'DIC_COMPONENT_INSTANCE') IN TYPEOF(dic_cl)
     THEN
               RETURN(('ISO13584_IEC61360_DICTIONARY_SCHEMA'
                    + '.COMPONENT_CLASS'
                    IN TYPEOF(dic_cl.class_def.definition[1])));
     END_IF;

     IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.' +
          'DIC_MATERIAL_INSTANCE') IN TYPEOF(dic_cl)
     THEN
               RETURN(('ISO13584_IEC61360_DICTIONARY_SCHEMA'
                    + '.MATERIAL_CLASS'
                    IN TYPEOF(dic_cl.class_def.definition[1])));
     END_IF;

     IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.' +
          'DIC_FEATURE_INSTANCE') IN TYPEOF(dic_cl)
     THEN
               RETURN('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
                    + '.FEATURE_CLASS'
                    IN TYPEOF(dic_cl.class_def.definition[1]));
     END_IF;

     IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.' +
          'DIC_ITEM_INSTANCE') IN TYPEOF(dic_cl)
```

```
      THEN
                RETURN(('ISO13584_IEC61360_DICTIONARY_SCHEMA'
                     + '.ITEM_CLASS'
                     IN TYPEOF(dic_cl.class_def.definition[1])));
      END_IF;

ELSE
      RETURN(UNKNOWN);
END_IF;
END_FUNCTION; -- check_class_type_for_dic_item_instance

(*
```

6.6.12 Check_class_type_for_dic_f_model_instance function

The **check_class_type_for_dic_model_instance** returns TRUE if the referenced class dictionary definition is type compatible with the given **dic_cl dic_f_model_instance**, otherwise it returns FALSE. If this dictionary definition is not available, it returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION check_class_type_for_dic_f_model_instance(
      dic_cl: dic_f_model_instance): LOGICAL;

IF (SIZEOF(dic_cl.class_def.definition) = 1)
THEN
      RETURN (('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
           + '.ABSTRACT_FUNCTIONAL_MODEL_CLASS'
           IN TYPEOF(dic_cl.class_def.definition[1])));
ELSE
      RETURN(UNKNOWN);
END_IF;
END_FUNCTION; -- check_class_type_for_dic_f_model_instance

(*
```

6.6.13 Check_class_type_for_dic_f_view_instance function

The **check_class_type_for_dic_f_view_instance** returns TRUE if the referenced class dictionary definition is type compatible with the given **dic_cl dic_f_view_instance**, otherwise it returns FALSE. If this dictionary definition is not available, it returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION check_class_type_for_dic_f_view_instance(
      dic_cl: dic_f_model_instance): LOGICAL;

IF (SIZEOF(dic_cl.class_def.definition) = 1)
THEN
      IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
           '.DIC_F_VIEW_INSTANCE') IN TYPEOF(dic_cl)
```



```
      THEN
               RETURN('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
               + '.FUNCTIONAL_VIEW_CLASS'
               IN TYPEOF(dic_cl.class_def.definition[1]));
      END_IF;

ELSE
      RETURN(UNKNOWN);
END_IF;
END_FUNCTION; -- check_class_type_for_dic_f_view_instance
(*
```

6.6.14 Check_property_values_translations function

The **check_property_values_translations** function returns TRUE if all the given **property_value**s are translated in the same language. Otherwise, it returns FALSE. If none of the **property_value**s are translated, it returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION check_property_values_translations(props:
      LIST OF property_value): LOGICAL;
LOCAL
      translated_string_values: SET OF translated_string_value := [];
END_LOCAL;

REPEAT i := 1 TO SIZEOF(props);
      translated_string_values :=
           translated_string_values + props[i].its_value;
END_REPEAT;

RETURN(same_translations(translated_string_values));

END_FUNCTION; -- check_property_values_translations
(*
```

6.6.15 Same_translations function

The **same_translations** function returns TRUE if all the given **translated_string_value**s are defined using the same languages. Otherwise, it returns FALSE. If no **translated_string_value** is provided, it returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION same_translations(translated_string_values: SET OF
      translated_string_value): LOGICAL;
LOCAL
      comp: translated_string_value;
END_LOCAL;

IF (SIZEOF(translated_string_values) <> 0)
```

```
THEN
      comp := translated_string_values[1];
      REPEAT i := 2 TO SIZEOF(translated_string_values);
           IF (translated_string_values[i].languages <>
                comp.languages)
           THEN
                RETURN(FALSE);
           END_IF;
      END_REPEAT;

      RETURN(TRUE);
ELSE
      RETURN(UNKNOWN);
END_IF;

END_FUNCTION; -- same_translations
(*
```

6.6.16 Compatible_item_caseof_with_class_definition function

The **compatible_item_caseof_with_class_definition** function checks that the **inst dic_item_instance** refers through its **case_of** attribute to compatible classes according to the **inst class_def** class definition.

The case-of relationship is inherited and transitive, thus, the classes that are allowed to be referenced by the **case_of** attribute of a **dic_item_instance** are all the classes that may be attained through transitive closure of the two relationships: case-of and inheritance.

The **compatible_item_caseof_with_class_definition** function returns TRUE if the **inst case_of** set is empty, or if all the classes referenced in the **inst case_of** set belong to the transitive closure of the case-of and inheritance relationships for the **inst class_def** class. It returns UNKNOWN if the **inst class_def** dictionary definition is not available, or if some dictionary definitions of some classes referenced by a **case_of** relationship are not available. Otherwise, it returns FALSE.

EXPRESS specification:

```
*)
FUNCTION compatible_item_caseof_with_class_definition(
      inst: dic_item_instance): LOGICAL;

IF (SIZEOF(inst.case_of) > 0)
THEN
      IF (SIZEOF(inst.class_def.definition) = 1)
      THEN
           RETURN ( inst.case_of
                     <=  item_caseof_closure( [inst.class_def]));
      ELSE
           RETURN(UNKNOWN);
      END_IF;
ELSE
      RETURN(TRUE);
END_IF;
```



```
END_FUNCTION; -- compatible_item_caseof_with_class_definition
(*
```

6.6.17 Compatible_model_caseof_with_class_definition function

The **compatible_model_caseof_with_class_definition** function checks that the **inst dic_f_model_instance** refers through its **case_of** attribute to compatible classes according to the **inst class_def** class definition.

The case-of relationship is inherited and transitive, thus, the classes that are allowed to be referenced by the **case_of** attribute of a **dic_f_model_instance** are all the classes that may be attained through transitive closure of the two relationships: case-of and inheritance.

The **compatible_model_caseof_with_class_definition** function returns TRUE if the **inst case_of** set is empty, or if all the classes referenced in the **inst case_of** set belong to the transitive closure of the case-of and inheritance relationships for the **inst class_def** class. It returns UNKNOWN if the **inst class_def** dictionary definition is not available, or if some dictionary definitions of some classes referenced by a **case_of** relationship are not available. Otherwise, it returns FALSE.

EXPRESS specification:

```
*)
FUNCTION compatible_model_caseof_with_class_definition(
     inst: dic_f_model_instance): LOGICAL;

IF (SIZEOF(inst.case_of) > 0)
THEN
     IF (SIZEOF(inst.class_def.definition) = 1)
     THEN
          RETURN ( inst.case_of
                   <=  model_caseof_closure( [inst.class_def]));
     ELSE
          RETURN(UNKNOWN);
     END_IF;
ELSE
     RETURN(TRUE);
END_IF;

END_FUNCTION; -- compatible_model_caseof_with_class_definition
(*
```

6.6.18 superclass_closure function

The **superclass_closure** function computes all the superclasses of the set of classes defined by the **current** set of **class_BSU**. It returns indeterminate (?) when some dictionary definitions are not available to compute this set.

EXPRESS specification:

```
*)

FUNCTION superclass_closure (
```



```
     current: SET OF class_BSU)    -- which classes
     :SET OF class_BSU;  -- all their superclasses or ?
 LOCAL
       superclasses : SET OF class_BSU := [];
 END_LOCAL;

        compute_superclass_closure (current, superclasses);
        RETURN (superclasses);

END_FUNCTION; -- superclass_closure
(*
```

6.6.19 compute_superclass_closure procedure

The **compute_superclass_closure** procedure computes recursively all the superclasses of the set of classes defined by the **current** set of **class_BSU** and returns them in the **visited** parameter. The **visited** parameter is indeterminate (?) when some dictionary definitions are not available to compute the set of all the superclasses.

EXPRESS specification:

```
*)

PROCEDURE compute_superclass_closure (
     current: SET OF class_BSU;       -- new superclasses
     var visited: SET OF class_BSU); -- already known superclasses

IF EXISTS(current)  THEN
     IF SIZEOF(current) <> 0 THEN
          REPEAT i := 1 TO SIZEOF(current);
               IF SIZEOF (current[i].definition) = 0
               THEN     visited := ?;
                        -- all superclasses cannot be computed
                        SKIP;
               ELSE
                 IF  EXISTS
                   (current[i].definition[1]\class.its_superclass)
                    AND NOT
                   (current[i].definition[1]\class.its_superclass
                   IN visited)
                  THEN  visited := visited
                   + [ current[i].definition[1]\class.its_superclass];
                   compute_superclass_closure(
                   [current[i].definition[1]\class.its_superclass]
                   , visited );
                   END_IF;
               END_IF;
            END_REPEAT;
     END_IF;
ELSE
     visited := ?;       -- all superclasses cannot be computed
```



```
END_IF;
END_PROCEDURE; -- compute_superclass_closure

(*
```

6.6.20 item_caseof_closure function

The **item_caseof_closure** function computes all the item classes of which the set of item classes defined by the **current** set of **class_BSU** are case-of, by means of direct or indirect case-of declaration, or by means of inheritance of a case-of relationship declared by some superclass. It returns indeterminate (?) when some dictionary definitions are not available to compute the result set.

EXPRESS specification:

```
*)
FUNCTION item_caseof_closure (
     current: SET OF class_BSU)    -- which classes
     :SET OF class_BSU;  -- all classes they are caseof or ?
 LOCAL
       caseof : SET OF class_BSU
              := next_item_caseof(superclass_closure (current));
 END_LOCAL;
        compute_item_caseof_closure (caseof, caseof);
        RETURN (caseof);
END_FUNCTION; -- item_caseof_closure
(*
```

6.6.21 next_item_caseof function

The **next_item_caseof** function computes all the item classes of which the set of item classes defined by the **current** set of **class_BSU** are case-of by means of a direct declaration. It returns indeterminate (?) when some dictionary definitions are not available to compute th e result set.

EXPRESS specification:

```
*)

FUNCTION next_item_caseof (
     current: SET OF class_BSU)    -- which classes
     :SET OF class_BSU;  -- classes they are directly caseof or ?
 LOCAL
       caseof : SET OF class_BSU := [];
 END_LOCAL;
     IF  EXISTS(current) THEN
          REPEAT i := 1 TO SIZEOF(current);
               IF SIZEOF (current[i].definition) = 0
                    THEN     caseof := ?;
                    -- all classes they are caseof cannot be computed
                         SKIP;
                    ELSE
                    IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA.'
                         +'ITEM_CLASS_CASE_OF' IN
                         TYPEOF(current[i].definition[1]))
```



```
                    THEN     caseof := caseof
                             + current[i].definition[1]
                             \item_class_case_of.is_case_of;
                    END_IF;
               END_IF;
          END_REPEAT;
          RETURN (caseof);
     ELSE
          RETURN (?); -- all classes they are caseof cannot be computed
     END_IF;
END_FUNCTION; -- next_item_caseof
(*
```

6.6.22 compute_item_caseof_closure procedure

The **compute_item_caseof_closure** procedure computes recursively all the item classes of which the set of item classes defined by the **current** set of **class_BSU** are case-of, by means of direct or indirect case-of declaration, or by means of inheritance of a case-of relationship declared by some superclass. The **compute_item_caseof_closure** procedure returns them in the **visited** attribute. The **visited** attribute is indeterminate (?) when some dictionary definitions are not available to compute the result set.

EXPRESS specification:

```
*)

PROCEDURE compute_item_caseof_closure (
     current: SET OF class_BSU;       -- last found caseof
     var visited: SET OF class_BSU);
          -- already known classes that are caseof (including current)
 LOCAL
       next : SET OF class_BSU ;           -- computed new caseof
 END_LOCAL;
 IF  EXISTS(current) THEN
     IF SIZEOF(current) <> 0 THEN
          next := superclass_closure (current);-- caseof by inheritance
          next := next_item_caseof ( next )+ next;
                                       -- and caseof by transitivity
          REPEAT i := 1 TO SIZEOF(next);
                    IF  NOT  (next[i] IN visited)
                    THEN
                      visited := visited + next[i] ;
                      compute_item_caseof_closure([next[i]], visited );
                    END_IF;
           END_REPEAT;
     END_IF;
ELSE
     visited := ?;
-- all classes that are caseof cannot be computed
END_IF;
END_PROCEDURE; -- compute_item_caseof_closure
```



```
(*
```

6.6.23 model_caseof_closure function

The **model_caseof_closure** function computes all the functional model classes of which the set of functional model classes defined by the **current** set of **class_BSU** are case-of, by means of direct or indirect case-of declaration, or by means of inheritance of a case-of relationship declared by some superclass. It returns indeterminate (?) when some dictionary definitions are not available to compute the result set.

EXPRESS specification:

```
*)
FUNCTION model_caseof_closure (
     current: SET OF class_BSU)    -- which classes
     :SET OF class_BSU;  -- all classes they are caseof or ?
 LOCAL
       caseof : SET OF class_BSU
              := next_model_caseof(superclass_closure (current));
 END_LOCAL;
        compute_model_caseof_closure (caseof, caseof);
        RETURN (caseof);
END_FUNCTION; -- model_caseof_closure
(*
```

6.6.24 next_model_caseof function

The **next_model_caseof** function computes all the functional model classes of which the set of functional model classes defined by the **current** set of **class_BSU** are case-of by means of a direct declaration. It returns indeterminate (?) when some dictionary definitions are not available to compute th e result set.

EXPRESS specification:

```
*)

FUNCTION next_model_caseof (
     current: SET OF class_BSU)    -- which classes
     :SET OF class_BSU;  -- classes they are directly caseof or ?
 LOCAL
       caseof : SET OF class_BSU := [];
 END_LOCAL;
     IF  EXISTS(current) THEN
          REPEAT i := 1 TO SIZEOF(current);
               IF SIZEOF (current[i].definition) = 0
                    THEN     caseof := ?;
                    -- all classes they are caseof cannot be computed
                         SKIP;
                    ELSE
                    IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA.'
                         +'ABSTRACT_FUNCTIONAL_MODEL_CLASS' IN
                         TYPEOF(current[i].definition[1]))
                    THEN     caseof := caseof
                             + current[i].definition[1]
```

```
                              \abstract_functional_model_class.case_of;
                    END_IF;
               END_IF;
          END_REPEAT;
          RETURN (caseof);
     ELSE
          RETURN (?); -- all classes they are caseof cannot be computed
     END_IF;
END_FUNCTION; -- next_model_caseof
(*
```

6.6.25 compute_model_caseof_closure procedure

The **compute_model_caseof_closure** procedure computes recursively all the functional model classes of which the set of functional model classes defined by the **current** set of **class_BSU** are case-of, by means of direct or indirect case-of declaration, or by means of inheritance of a case-of relationship declared by some superclass. The **compute_model_caseof_closure** procedure returns them in the **visited** attribute. The **visited** attribute is indeterminate (?) when some dictionary definitions are not available to compute the result set.

EXPRESS specification:

```
*)

PROCEDURE compute_model_caseof_closure (
     current: SET OF class_BSU;       -- last found caseof
     var visited: SET OF class_BSU);
          -- already known classes that are caseof (including current)
 LOCAL
       next : SET OF class_BSU ;           -- computed new caseof
 END_LOCAL;
 IF  EXISTS(current) THEN
     IF SIZEOF(current) <> 0 THEN
          next := superclass_closure (current);-- caseof by inheritance
          next := next_model_caseof ( next )+ next;
                                     -- and caseof by transitivity
          REPEAT i := 1 TO SIZEOF(next);
                    IF  NOT  (next[i] IN visited)
                    THEN
                      visited := visited + next[i] ;
                      compute_model_caseof_closure([next[i]],visited );
                    END_IF;
           END_REPEAT;
     END_IF;
ELSE
     visited := ?;
-- all classes that are caseof cannot be computed
END_IF;
END_PROCEDURE; -- compute_model_caseof_closure

(*
```



```
*)

END_SCHEMA; -- ISO13584_instance_resource_schema

(*
```

7 ISO13584_library_expressions_schema

This clause defines the requirements for the **ISO13584_library_expressions_schema**. The following EXPRESS declaration introduces the **ISO13584_library_expressions_schema** block and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA ISO13584_library_expressions_schema;

REFERENCE FROM ISO13584_IEC61360_dictionary_schema
      (class_BSU,
      definition_available_implies,
      level,
      list_to_set,
      number_type,
      property_BSU);

REFERENCE FROM ISO13584_generic_expressions_schema
      (binary_generic_expression,
      environment,
      generic_expression,
      generic_literal,
      generic_variable,
      multiple_arity_generic_expression,
      simple_generic_expression,
      unary_generic_expression,
      variable_semantics);

REFERENCE FROM ISO13584_expressions_schema
      (boolean_defined_function,
      expression,
      is_int_expr,
      variable);

REFERENCE FROM ISO13584_instance_resource_schema
      (compatible_class_and_class,
      compatible_level_type_and_instance,
      dic_class_instance,
      entity_instance_value,
      int_level_spec_value,
      level_spec_value,
      property_or_data_type_BSU,
      real_level_spec_value);
```



```
    REFERENCE FROM ISO13584_extended_dictionary_schema
         (applicable_properties,
         data_type_class_of,
         data_type_level_spec,
         data_type_level_value_typeof,
         data_type_type_name,
         data_type_typeof);
    (*
```

NOTE The schemas referenced above can be found in the following documents:
ISO13584_IEC61360_dictionary_schema IEC 61360-2
(which is duplicated for convenience in informative annex D of ISO 13584-42),
ISO13584_generic_expressions_schema ISO 13584-20,
ISO13584_expressions_schema ISO 13584-20,
ISO13584_instance_resource_schema This part of ISO 13584,
ISO13584_extended_dictionary_schema This part of ISO 13584.

7.1 Introduction to the ISO13584_library_expressions_schema

The role of the **ISO13584_library_expressions_schema** is to provide resources for representing expressions that evaluate to a value belonging to any one of the data types defined in the **ISO13584_ISO61360_dictionary_schema**. Therefore, the resource constructs provided in the **ISO13584_library_expressions_schema** enable the specification of the value of any property defined in a dictionary conformant with ISO 13584-42 by means of an expression.

The **ISO13584_library_expressions_schema** extends the type of expressions defined in the **ISO13584_expressions_schema** by subtyping the generic resources defined in the **ISO13584_generic_expressions_schema** for each of the complex types defined in the **ISO13584_IEC61360_dictionary_schema**. For all these expressions, except the **class_instance_expression**s, the expressions defined in this schema consists only of variables and literals.

NOTE The expressions defined in the **ISO13584_expressions_schema** correspond to simple EXPRESS data types.

For a **class_instance_expression**, an operator similar to the EXPRESS entity instance constructor is introduced. This operator enables to specificy how an instance of a class defined according to the **ISO13584_IEC61360_dictionary_schema** may be generated from other expressions.

In ISO 13584, a variable may be associated with a content of no value. It is the case in particular when the variable represents the value of an instance property, and when this property is specified as being optional in a class extension described according to the **ISO13584_library_content_schema**.

The **exists_value** entity is the second operator introduced in the **ISO13584_library_expressions_schema**. This operator applies to any **library_variable**. It shall evaluate to TRUE if the variable has a value, and to FALSE if it does not.

The **instance_comparison_equal** entity is the third operator introduced in the **ISO13584_library_expressions_schema**. This operator applies to any couple of **generic_expression**s. It evaluates to TRUE if both **generic_expression**s evaluate to the same simple value or to the same instance value, and to FALSE if they evaluate to two different values.

The **ISO13584_library_expressions_schema** models:

— the representation for strongly typed variables whose types of value shall belong to one of the **complex_type**s defined in the **ISO13584_IEC61360_dictionary_schema**;

— the representation for strongly typed literals whose types of value shall belong to one of the **complex_type**s defined in the **ISO13584_IEC61360_dictionary_schema**;

— the definition of an operator that evaluates to a class instance of which the property values are specified by means of other expressions;

— the definition of an operator that controls whether a variable has been assigned a value;

— the definition of an operator that controls whether two generic expression evaluate to the same instance value.

The **ISO13584_library_expressions_schema** does not model:

— the representation for strongly typed variables or literals whose types of value belong to one of the **simple_type**s defined in the **ISO13584_IEC61360_dictionary_schema**;

NOTE The representation of the variables and literals whose types of value shall belong to one of the **simple_type**s of **ISO13584_IEC61360_dictionary_schema** is defined in ISO 13584-20: 1998.

— the representation for other types of expressions than the ones defined in the **ISO13584_IEC61360_dictionary_schema** to specify the data types of the values that may be associated with a property;

— the computer-interpretable or human-readable definition of the interpretation function that assigns values to variables.

7.2 Fundamental concepts and assumptions for the ISO13584_library_expressions_schema

7.2.1 Information model of a variable

In accordance with ISO 13584-20, a variable may be modelled by a threefold information model:

— a **generic_variable** representing the syntactical aspect of the variable and enabling to model its involvement in a **generic_expression**,

— a **variable_semantics** capturing the meaning of the variable, i.e., specifying the context within which the variable shall be used together with the interpretation function that associates a value with this variable, and

— an **environment** associating a **variable_semantics** with a **generic_variable**.

NOTE 1 **generic_variable**, **variable_semantics** and **environment** are specified in ISO 13584-20.

NOTE 2 When the value of a variable is to be represented explicitly, this value may be modelled using the resource constructs specified in the **ISO13584_instance_resource_schema**.

7.2.2 Strong typing of variables and expressions

Strong typing requires that each syntactical object be associated with a data type.

For a variable, this information may be modelled by subtyping **generic_variable**. ISO 13584-20 defines **variable** as the subtype of **generic_variable**s that contains all the variables whose data types are **simple_type**s. The role of the **ISO13584_library_expressions_schema** is to define other subtypes of **generic_variable**s that correspond to the **complex_type**s defined in the **ISO13584_IEC61360_dictionary_schema**. A **library_variable** is either a **variable** as defined by ISO 13584-20, or any one of the other subtypes defined in the **ISO13584_library_expressions_schema**.

NOTE ISO 13584-20 defines the following subtypes of **variable**: **numeric_variable**, **boolean_variable** and **string_variable**. It also defines the following subtypes of **numeric_variable**: **int_numeric_variable** and **real_numeric_variable**.

For an expression, this information may be modelled by subtyping **generic_expression**. ISO 13584-20 defines three subtypes of **generic_expression**s that are **numeric_expression**s, **boolean_expression**s and **string_expression**s. The role of the **ISO13584_library_expressions_schema** is to define other subtypes of **generic_expression**s that correspond to the **complex_type**s defined in the **ISO13584_IEC61360_dictionary_schema**. A **library_expression** is either an **expression** as defined by ISO 13584-20, or any one of the other subtypes of **generic_expression**s defined in the **ISO13584_library_expressions_schema**.

7.3 ISO13584_library_expressions_schema type definitions

This clause provides the type definitions of the **ISO13584_library_expressions_schema**. These types specify the expressions and variables that may be associated with a property defined according to ISO 13584-42.

7.3.1 Library_expression

A **library_expression** is either an **expression** as defined in **ISO13584_expressions_schema**, or an expression that evaluates to a value belonging to one of the complex types defined in the **ISO13584_IEC61360_dictionary_schema**.

NOTE 1 **expression** is defined in the **ISO13584_expressions_schema** documented in ISO 13584-20. The **expression** entity data type models expressions that evaluate to a value belonging to one of the following types: number, string and Boolean.

NOTE 2 The complex types defined in the **ISO13584_IEC61360_dictionary_schema** are **level_type**, **entity_instance_type**, and **class_instance_type**. The **ISO13584_IEC61360_dictionary_schema** is defined in IEC 61360-2, and duplicated for convenience in informative annex D of ISO 13584-42.

NOTE 3 An expression that evaluates to a value belonging to a **level_type** is modelled by a **level_spec_expression** entity. An expression that evaluates to a value belonging to an **entity_instance_type** is modelled by an **entity_instance_expression** entity. An expression that evaluates to a value belonging to a **class_instance_type** is modelled by a **class_instance_expression** entity.

EXPRESS specification:

```
*)
TYPE library_expression = SELECT(
     expression,
     level_spec_expression,
     entity_instance_expression,
     class_instance_expression);
END_TYPE; -- library_expression
(*
```

7.3.2 Library_variable

A **library_variable** is either a **variable** as defined in **ISO13584_expressions_schema**, or a variable, that is associated with a value belonging to one of the complex types defined in the **ISO13584_IEC61360_dictionary_schema**, i.e., a **level_spec_variable**, an **entity_instance_variable** or a **class_instance_variable**.

EXPRESS specification:

```
*)
TYPE library_variable = SELECT(
     variable,
     level_spec_variable,
     entity_instance_variable,
     class_instance_variable);
END_TYPE; -- library_variable
(*
```

7.4 ISO13584_library_expressions_schema entity definitions

This clause provides the entity definitions of the **ISO13584_library_expressions_schema**. These entities define a strongly typed set of expressions for all the complex data types that may appear in an ISO 13584 conformant dictionary. They specialise the resources defined in the **ISO13584_generic_expressions_schema** in order to cover the whole type system defined in the **ISO13584_IEC61360_dictionary_schema**.

NOTE Strong typing requires that each syntactical object is associated with a data type.

7.4.1 Level_spec_expression

A **level_spec_expression** entity is an expression of which the range is defined as an ARRAY of four optional numbers, that carries the additional meaning that:

— the first value corresponds to the minimum value of some property;

— the second value corresponds to the nominal value of some property;

— the third value corresponds to the typical value of some property;

— the fourth value corresponds to the maximum value of some property.

The four numbers shall be either NUMBERs or their data type shall be redefined through the **value_type** attribute.

NOTE No operators are defined in the **ISO13584_library_expressions_schema** for **level_spec_expression**.

EXPRESS specification:

```
*)
ENTITY level_spec_expression
ABSTRACT SUPERTYPE OF(simple_level_spec_expression)
SUBTYPE OF(generic_expression);
     levels: LIST[1:4] OF UNIQUE level;
     value_type: number_type;
END_ENTITY; -- level_spec_expression
(*
```

Attribute definitions:

levels: the list of unique **level**s that specifies which of the optional values shall exist when the **level_spec_expression** is evaluated.

value_type: the type of value for the different values in the **level_spec_expression**; this shall be **number_type**, **int_type** or **real_type**.

NOTE The **ISO13584_IEC61360_dictionary_schema** only references **number_type**, **int_type** or **real_type**.

7.4.1.1 Simple_level_spec_expression

A **simple_level_spec_expression** is either a **level_spec_variable** or a **level_spec_literal**.

EXPRESS specification:

```
*)
ENTITY simple_level_spec_expression
ABSTRACT SUPERTYPE OF(ONEOF(
     level_spec_variable,
     level_spec_literal))
SUBTYPE OF(level_spec_expression, simple_generic_expression);
END_ENTITY; -- simple_level_spec_expression
(*
```

7.4.1.2 Level_spec_variable

A **level_spec_variable** is a variable whose value is a **level_spec_value**. As defined in the **ISO13584_instance_resource_schema**, a **level_spec_value** is an ARRAY of four optional numbers, that carry the additional meaning that:

— the first value corresponds to the minimum value of some property;

— the second value corresponds to the nominal value of some property;

— the third value corresponds to the typical value of some property;

— the fourth value corresponds to the maximum value of some property.

The four numbers shall be NUMBERs.

EXPRESS specification:

```
*)
ENTITY level_spec_variable
ABSTRACT SUPERTYPE OF(ONEOF(
     int_level_spec_variable,
     real_level_spec_variable))
SUBTYPE OF(simple_level_spec_expression, generic_variable);
END_ENTITY; -- level_spec_variable
(*
```

7.4.1.3 Int_level_spec_variable

An **int_level_spec_variable** is a **level_spec_variable** whose values, if present are all INTEGER.

EXPRESS specification:

```
*)
ENTITY int_level_spec_variable
SUBTYPE OF(level_spec_variable);
WHERE
     WR1: 'ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_TYPE'
          IN TYPEOF(SELF\level_spec_expression.value_type);
END_ENTITY; -- int_level_spec_variable
(*
```

Formal propositions:

WR1: the **value_type** of the **level_spec_expression** shall contain **int_type**.

7.4.1.4 Real_level_spec_variable

A **real_level_spec_variable** is a **level_spec_variable** whose optional values are all REAL.

EXPRESS specification:

```
*)
ENTITY real_level_spec_variable
SUBTYPE OF(level_spec_variable);
WHERE
     WR1: 'ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE'
          IN TYPEOF(SELF\level_spec_expression.value_type);
END_ENTITY; -- real_level_spec_variable
(*
```

Formal propositions:

WR1: the **value_type** of the **level_spec_expression** shall contain **real_type**.

7.4.1.5 Level_spec_literal

A **level_spec_literal** is a literal whose value is a **level_spec_value** as defined in the **ISO13584_instance_resource_schema**. It is an abstract supertype of **int_level_spec_literal** and **real_level_spec_literal**.

EXPRESS specification:

```
*)
ENTITY level_spec_literal
ABSTRACT SUPERTYPE OF(ONEOF(
     int_level_spec_literal,
     real_level_spec_literal))
SUBTYPE OF(simple_level_spec_expression, generic_literal);
     the_value: level_spec_value;
WHERE
     WR1: compatible_level_type_and_instance(
          SELF\level_spec_expression.levels,
```

```
          TYPEOF(SELF\level_spec_expression.value_type),
          SELF.the_value);
END_ENTITY; -- level_spec_literal
(*
```

Attribute definitions:

the_value: the **level_spec_value** that defines the value of the literal.

Formal propositions:

WR1: the **level_spec_value** shall be compatible with the type defined in the **level_spec_expression**.

7.4.1.6 Int_level_spec_literal

An **int_level_spec_literal** is a literal whose value is an **int_level_spec_value** as defined in the **ISO13584_instance_resource_schema**.

EXPRESS specification:

```
*)
ENTITY int_level_spec_literal
SUBTYPE OF(level_spec_literal);
     SELF\level_spec_literal.the_value: int_level_spec_value;
WHERE
     WR1: 'ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_TYPE'
          IN TYPEOF(SELF\level_spec_expression.value_type);
     WR2: compatible_level_type_and_instance(
          SELF\level_spec_expression.levels,
          ['ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_TYPE'],
          SELF.the_value);
END_ENTITY; -- int_level_spec_literal
(*
```

Formal propositions:

WR1: the **value_type** of the **level_spec_expression** shall contain **int_type**.

WR2: the existing values shall all be INTEGERs.

7.4.1.7 Real_level_spec_literal

A **real_level_spec_literal** is a literal whose value is an **real_level_spec_value** as defined in the **ISO13584_instance_resource_schema**.

EXPRESS specification:

```
*)
ENTITY real_level_spec_literal
SUBTYPE OF(level_spec_literal);
     SELF\level_spec_literal.the_value: real_level_spec_value;
```



```
WHERE
     WR1: 'ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE'
          IN TYPEOF(SELF\level_spec_expression.value_type);
     WR2: compatible_level_type_and_instance(
          SELF\level_spec_expression.levels,
          ['ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE'],
          SELF.the_value);
END_ENTITY; -- real_level_spec_literal
(*
```

Formal propositions:

WR1: the **value_type** of the **level_spec_expression** shall contain **real_type**.

WR2: the existing values shall all be REALs.

7.4.2 Entity_instance_expression

An **entity_instance_expression** is an expression whose value is some EXPRESS ENTITY data type. Each **entity_instance_expression** has a **type_name** attribute that specifies, as a set of STRINGs, the type of the **entity_instance_expression**. This set of STRINGs shall be contained in the result of the EXPRESS TYPEOF function applied to any value to which the **entity_instance_expression** evaluates, in order for this value to be compatible with the type of the expression.

NOTE 1 No operators are defined in the **ISO13584_library_expressions_schema** for **entity_instance_expression**.

NOTE 2 If an EXPRESS ENTITY data type is compatible with the type of an **entity_instance_expression**, all the subtypes of this EXPRESS ENTITY data type are compatible with the type of this **entity_instance_expression**. Thus **entity_instance_expression** supports inheritance.

NOTE 3 View exchange protocols specify which **entity_instance_type**s (and subtypes) and **entity_instance_value**s are permitted.

EXPRESS specification:

```
*)
ENTITY entity_instance_expression
ABSTRACT SUPERTYPE OF(simple_entity_instance_expression)
SUBTYPE OF(generic_expression);
     type_name: SET [1:?] OF STRING;
END_ENTITY; -- entity_instance_expression
(*
```

Attribute definitions:

type_name: the set of STRINGs that specifies the type of the **entity_instance_expression**.

Informal propositions:

IP1: the **type_name** attribute of an **entity_instance_expression** shall be contained in the result of the EXPRESS TYPEOF applied to any **entity_instance_value** to which the **entity_instance_expression** evaluates.

7.4.2.1 Simple_entity_instance_expression

A **simple_entity_instance_expression** is either an **entity_instance_variable** or an **entity_instance_literal**.

EXPRESS specification:

```
*)
ENTITY simple_entity_instance_expression
ABSTRACT SUPERTYPE OF(ONEOF(
     entity_instance_variable,
     entity_instance_literal))
SUBTYPE OF(entity_instance_expression, simple_generic_expression);
END_ENTITY; -- simple_entity_instance_expression
(*
```

7.4.2.2 Entity_instance_variable

An **entity_instance_variable** is a variable whose value is an **entity_instance_value**. As defined in the **ISO13584_instance_resource_schema**, an **entity_instance_value** is a value that is represented by an instance of some EXPRESS ENTITY data type. The type of the entity instance data type is defined by the inherited **type_name** attribute that is intended to be contained in the result of the EXPRESS TYPEOF function applied to any value of this variable, in order for this value to be compatible with the type of the variable.

NOTE The value of an **entity_instance_variable** may belong to any subtype of the type defined by the **type_name** inherited attribute. Thus, the **entity_instance_variable**s support polymorphism.

EXPRESS specification:

```
*)
ENTITY entity_instance_variable
SUBTYPE OF(simple_entity_instance_expression, generic_variable);
END_ENTITY; -- entity_instance_variable
(*
```

7.4.2.3 Entity_instance_literal

An **entity_instance_literal** is a literal whose value is an **entity_instance_value** as defined in the **ISO13584_instance_resource_schema**.

EXPRESS specification:

```
*)
ENTITY entity_instance_literal
SUBTYPE OF(simple_entity_instance_expression, generic_literal);
     the_value: entity_instance_value;
WHERE
     WR1: (SELF\entity_instance_expression.type_name
          <= TYPEOF(SELF.the_value))
          OR (('ISO13584_INSTANCE_RESOURCE_SCHEMA'
          + '.UNCONTROLLED_ENTITY_INSTANCE_VALUE')
          IN TYPEOF(SELF.the_value));
```



```
END_ENTITY; -- entity_instance_literal
(*
```

Formal propositions:

WR1: the value of the **entity_instance_literal** shall be compatible with its data type as defined by the **type_name** attribute, or this value shall be an **uncontrolled_entity_instance_value**, as defined by the **ISO13584_instance_resource_schema**.

7.4.3 Class_instance_expression

A **class_instance_expression** is an expression for which the range is defined by some **class** as defined in the **ISO13584_IEC61360_dictionary_schema**. Each **class_instance_expression** has a **expr_type** attribute that specifies, as a **class_BSU**, the type of the **class_instance_expression**. Any value to which the **class_instance_expression** evaluates shall be an instance of this class or any of its version-compatible subclasses, in order for this value to be compatible with the type of the expression. Thus a **class_instance_expression** supports polymorphism.

NOTE 1 The only operator defined in the **ISO13584_library_expressions_schema** for **class_instance_expression** is the **class_instance_constructor**. This operator enables to specify how an instance of a class defined according to the **ISO13584_IEC61360_dictionary_schema** may be generated from other expressions. This operator is further decomposed according to its arity.

NOTE 2 A subclass of a class (cl1) whose dictionary element complies with the **ISO13584_IEC61360_dictionary_schema** is version-compatible with this class cl1 if it is one of its subclass, or if it is a subclass of a class cl2 such that:

— cl2 is defined by the same supplier as cl1, and

— cl2 has the same code as cl1, and

— cl2 has a version smaller or equal to the version of cl1.

This compatibility requirement is documented in the **compatible_class_and_class** function of the **ISO13584_instance_resource_schema**.

NOTE 3 Note that a **class_instance_expression** supports a double inheritance mechanism:

— the value of the **class_instance_expression** may be any EXPRESS subtype of **dic_class_instance** as defined in clause 6;

— the class of this value may be any version-compatible subclass of the class defined by the **expr_type** attribute.

EXPRESS specification:

```
*)
ENTITY class_instance_expression
ABSTRACT SUPERTYPE OF(ONEOF(
     simple_class_instance_expression,
     unary_class_instance_expression,
     binary_class_instance_expression,
     multiple_arity_class_instance_expression))
SUBTYPE OF(generic_expression);
     expr_type: class_BSU;
```



```
END_ENTITY; -- class_instance_expression
(*
```

Attribute definitions:

expr_type: the **class_BSU** of which associated **class** specifies the type of the **class_instance_expression**.

Informal propositions:

IP1: the value to which the **class_instance_expression** evaluates shall be an instance of the class defined by the **expr_type** attribute or any of its version-compatible subclasses.

NOTE 4 The evaluation function of a **class_instance_expression**, or any **generic_expression**, is outside the scope of this part of ISO 13584.

7.4.3.1 Simple_class_instance_expression

A **simple_class_instance_expression** is either a **class_instance_variable** or a **class_instance_literal**.

EXPRESS specification:

```
*)
ENTITY simple_class_instance_expression
ABSTRACT SUPERTYPE OF(ONEOF(
     class_instance_variable,
     class_instance_literal))
SUBTYPE OF(class_instance_expression, simple_generic_expression);
END_ENTITY; -- simple_class_instance_expression
(*
```

7.4.3.2 Class_instance_variable

A **class_instance_variable** is a variable whose value is a **dic_class_instance**. As defined in the **ISO13584_instance_resource_schema**, a **dic_class_instance** is an instance of some **class** as defined in the **ISO13584_IEC61360_dictionary_schema**.

NOTE 1 Each **class_instance_variable** has an inherited **expr_type** attribute, that specifies, through its **class_BSU**, the **class** of which the value shall be an instance.

NOTE 2 The instance that corresponds to the value of a **class_instance_variable** may belong to any version-compatible subclass of the class specified by its **expr_type** attribute. Thus the **class_instance_variable**s support polymorphism.

EXPRESS specification:

```
*)
ENTITY class_instance_variable
SUBTYPE OF(simple_class_instance_expression, generic_variable);
END_ENTITY; -- class_instance_variable
(*
```


Informal propositions:

IP1: the value of the **class_instance_variable** shall be compatible with the domain defined by its inherited **expr_type** attribute.

7.4.3.3 Class_instance_literal

A **class_instance_literal** is a literal whose value is a **dic_class_instance** or one of its subtypes as defined in the **ISO13584_instance_resource_schema**.

EXPRESS specification:

```
*)
ENTITY class_instance_literal
SUBTYPE OF(simple_class_instance_expression, generic_literal);
     the_value: dic_class_instance;
WHERE
     WR1: compatible_class_and_class(
          SELF\class_instance_expression.expr_type,
          SELF.the_value\dic_class_instance.class_def);
END_ENTITY; -- class_instance_literal
(*
```

Attribute definitions:

the_value: the **dic_class_instance** that defines the value of the literal.

Formal propositions:

WR1: the **dic_class_instance** that is the value of the literal shall be an instance of the class defined by the inherited **expr_type** or any of its version-compatible subclasses.

7.4.3.4 Class_instance_constructor

A **class_instance_constructor** is a **class_instance_expression** that specifies an instance of a class by defining:

— through its **expr_type** inherited attribute, the class the instance belongs to, and

— the values of some instance properties through **library_expression**s that specify their values.

Following the approach defined in the **ISO13584_generic_expressions_schema**, a **class_instance_constructor** is further decomposed according to its arity, i.e., the number of expressions used to specify the values of some instance properties. A **class_instance_constructor** is either an **unary_class_instance_constructor** or a **binary_class_instance_constructor** or a **multiple_arity_class_instance_constructor**.

NOTE 1 The set of the properties that are represented in a **class_instance_constructor** entity shall be applicable properties for the class referenced by the **expr_type** attribute.

NOTE 2 A **class_instance_constructor** does not specify which applicable properties shall be assigned a value. When this resource construct is used, a **class_instance_constructor** may be further constrained.

EXAMPLE In the **ISO13584_library_content_schema**, a **class_instance_constructor** is used to specify how the components that constitute a structured part may be derived from the identification of the structured

part. A WHERE RULE ensures that the components belongs to an **item_class** and that all the identification characteristics of the class are referenced in the **properties** set.

NOTE 3 A **class_instance_constructor** is similar to the entity instance constructor defined in the EXPRESS language.

EXPRESS specification:

```
*)
ENTITY class_instance_constructor
ABSTRACT SUPERTYPE OF(ONEOF(
     unary_class_instance_constructor,
     binary_class_instance_constructor,
     multiple_arity_class_instance_constructor))
SUBTYPE OF(class_instance_expression);
     properties: SET [1:?] OF property_assignment;
WHERE
     WR1: definition_available_implies
          (SELF\class_instance_expression.expr_type,
          applicable_properties(
          SELF\class_instance_expression.expr_type,
          list_to_set(collects_assigned_properties(SELF.properties))));
END_ENTITY; -- class_instance_constructor
(*
```

Attribute definitions:

properties: the set of **property_assignment**s that define the values for the properties of the class instance.

Formal propositions:

WR1: if data are available, then **IP1** holds;

Informal propositions:

IP1: the set of properties that are represented in a **class_instance_constructor** entity shall be applicable properties for the class referenced by the **expr_type** attribute.

7.4.3.5 Property_assignment

A **property_assignment** is the assignment of the value of an optional **library_expression** to a property of a class instance specified by a **class_instance_constructor**. A **property_assignment** also enables a property to specify that it has no value by referencing this property in a **property_assignment** where the **its_value** attribute does not exist.

NOTE If a property is defined in some application context as optional, assigning no value to this property means that it does not exist.

EXPRESS specification:

```
*)
ENTITY property_assignment;
     its_value: OPTIONAL library_expression;
     prop_def: property_BSU;
WHERE
     WR1: (EXISTS(SELF.its_value) AND
          (compatible_type_and_library_expression(
          SELF.prop_def, SELF.its_value)))
          OR NOT EXISTS(SELF.its_value);
END_ENTITY; -- property_assignment
(*
```

Attribute definitions:

its_value: the **library_expression** value assigned to the **prop_def** property.

prop_def: the property that describes the instance property to which the **its_value** refers.

Formal propositions:

WR1: the type of the value assigned to the property shall be compatible with the type of the property.

7.4.3.6 Unary_class_instance_expression

A **unary_class_instance_expression** is a unary operator of which range is a class instance. Its only subtype is the **unary_class_instance_constructor**.

EXPRESS specification:

```
*)
ENTITY unary_class_instance_expression
ABSTRACT SUPERTYPE OF(unary_class_instance_constructor)
SUBTYPE OF(class_instance_expression, unary_generic_expression);
END_ENTITY; -- unary_class_instance_expression
(*
```

7.4.3.7 Binary_class_instance_expression

A **binary_class_instance_expression** is a binary operator of which range is a class instance. Its only subtype is the **binary_class_instance_constructor**.

EXPRESS specification:

```
*)
ENTITY binary_class_instance_expression
ABSTRACT SUPERTYPE OF(binary_class_instance_constructor)
SUBTYPE OF(class_instance_expression, binary_generic_expression);
END_ENTITY; -- binary_class_instance_expression
(*
```

7.4.3.8 Multiple_arity_class_instance_expression

A **multiple_arity_class_instance_expression** is a multiple-arity operator of which range is a class instance. Its only subtype is the **multiple_arity_class_instance_constructor**.

EXPRESS specification:

```
*)
ENTITY multiple_arity_class_instance_expression
ABSTRACT SUPERTYPE OF(multiple_arity_class_instance_constructor)
SUBTYPE OF(class_instance_expression,
     multiple_arity_generic_expression);
END_ENTITY; -- multiple_arity_class_instance_expression
(*
```

7.4.3.9 Unary_class_instance_constructor

A **unary_class_instance_constructor** is a **class_instance_constructor** of which only one property is assigned a value through a **library_expression**.

EXPRESS specification:

```
*)
ENTITY unary_class_instance_constructor
SUBTYPE OF(class_instance_constructor,
     unary_class_instance_expression);
     SELF\class_instance_constructor.properties:
          SET [1:1] OF property_assignment;
DERIVE
     SELF\unary_generic_expression.operand: library_expression
          := collects_referenced_library_expressions(
          SELF.properties)[1];
END_ENTITY; -- unary_class_instance_constructor
(*
```

Attribute definitions:

properties: the **property_assignment** that defines the value for the property of the class instance.

SELF\unary_generic_expression.operand: the **library_expression** that is the operand of the **unary_generic_expression**.

7.4.3.10 Binary_class_instance_constructor

A **binary_class_instance_constructor** is a **class_instance_constructor** of which two properties are assigned a value through a **library_expression**. It is a subtype of **binary_generic_expression**.

EXPRESS specification:

```
*)
ENTITY binary_class_instance_constructor
SUBTYPE OF(class_instance_constructor,
```



```
      binary_class_instance_expression);
      SELF\class_instance_constructor.properties:
           SET [2:2] OF property_assignment;
DERIVE
      SELF\binary_generic_expression.operands:
           LIST [2:2] OF library_expression
           := collects_referenced_library_expressions(SELF.properties);
WHERE
      WR1: SIZEOF(list_to_set(collects_assigned_properties(
           SELF.properties)))
           = SIZEOF(collects_assigned_properties(SELF.properties));
END_ENTITY; -- binary_class_instance_constructor
(*
```

Attribute definitions:

properties: the list of two **property_assignment**s that define the values for the properties of the class instance.

SELF\binary_generic_expression.operands: the two **library_expression**s that are the operands of the **binary_generic_expression**.

Formal propositions:

WR1: a property may be referenced only once in a **class_instance_constructor**.

7.4.3.11 Multiple_arity_class_instance_constructor

A **multiple_arity_class_instance_constructor** is a **class_instance_constructor** of which two or more properties are assigned values through **library_expression**s. It is a subtype of **multiple_arity_generic_expression**.

EXPRESS specification:

```
*)
ENTITY multiple_arity_class_instance_constructor
SUBTYPE OF(class_instance_constructor,
      multiple_arity_class_instance_expression);
      SELF\class_instance_constructor.properties:
           SET [2:?] OF property_assignment;
DERIVE
      SELF\multiple_arity_generic_expression.operands:
           LIST [2:?] OF library_expression
           := collects_referenced_library_expressions(SELF.properties);
WHERE
      WR1: SIZEOF(list_to_set(collects_assigned_properties(
           SELF.properties)))
           = SIZEOF(collects_assigned_properties(SELF.properties));
END_ENTITY; -- multiple_arity_class_instance_constructor
(*
```


Attribute definitions:

properties: the list of **property_assignment**s that define the values for the properties of the class instance.

SELF\multiple_arity_generic_expression.operands: the **library_expression**s that are the operands of the **multiple_arity_generic_expression**.

Formal propositions:

WR1: a property may be referenced only once in a **class_instance_constructor**.

7.4.4 Exists_value

An **exists_value** entity is an operator that applies to any **library_variable** and of which range is BOOLEAN. It shall evaluate to TRUE if the variable has a value, and to FALSE if it does not have value.

NOTE If a property is defined in some application context as optional, assigning no value to the variable that substitutes for the value of this property means that this property does not exist.

EXPRESS specification:

```
*)
ENTITY exists_value
SUBTYPE OF(unary_generic_expression, boolean_defined_function);
      for_variable: library_variable;
DERIVE
      SELF\unary_generic_expression.operand: generic_expression
           := SELF.for_variable;
END_ENTITY; -- exists_value
(*
```

Attribute definitions:

for_variable: the **library_variable** for which the value existence is checked.

Informal propositions:

IP1: the **exists_value** operator shall evaluate to TRUE if the **for_variable** variable has a value, and to FALSE if it does not.

7.4.5 Instance_comparison_equal

An **instance_comparison_equal** entity carries the semantics of the value instance equal (':=:') operator defined in ISO 10303-11 restricted to operands that shall have a value.

NOTE 1 When applied to numeric, Boolean and string data types, the **instance_comparison_equal** is equivalent to value comparison, i.e., to the **comparison_equal** entity.

NOTE 2 The **comparison_equal** entity is defined in ISO 13584-20.

NOTE 3 When applied to instances whose data types are **complex_type**s, **instance_comparison_equal** evaluates to TRUE when both operands evaluate to the same instance entity, i.e., if the implementation

dependent identifiers are the same. It evaluates to FALSE if both operands evaluate to two different entity instances.

NOTE 4 The **complex_type** entity is defined in ISO 13584-42. **complex_type** is a supertype of **level_type**, **class_instance_type** and **entity_instance_type**.

EXPRESS specification:

```
*)
ENTITY instance_comparison_equal
SUBTYPE OF(binary_generic_expression, boolean_defined_function);
WHERE
     WR1: ('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA.LIBRARY_EXPRESSION' IN
          TYPEOF (SELF\binary_generic_expression.operands[1]))
          AND ('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA.LIBRARY_EXPRESSION'
          IN TYPEOF(SELF\binary_generic_expression.operands[2]));
END_ENTITY; -- instance_comparison_equal
(*
```

Formal proposition:

WR1: the **operands generic_expression**s shall be **library_expression**s.

Informal proposition:

IP1: the **instance_comparison_equal** operator shall evaluate to TRUE if **SELF\binary_generic_expression.operand[1]** and **SELF\binary_generic_expression.operand[2]** evaluate to the same simple value or to the same instance value, and to FALSE if they evaluate to two different values.

7.5 ISO13584_library_expressions_schema rule definition

ISO 13584-20: 1998 enables a **variable_semantics** to be bounded with several **generic_variable**s. In an ISO 13584 conformant exchange context, only one **generic_variable** may represent a **variable_semantics**. This is documented in the **two_fold_variable_representation_rule**.

7.5.1 Two_fold_variable_representation_rule rule

The **two_fold_variable_representation_rule** rule ensures that, in an ISO 13584 conformant exchange context, each **variable_semantics** is associated with no more than one **generic_variable**.

EXPRESS specification:

```
*)
RULE two_fold_variable_representation_rule FOR(variable_semantics);
WHERE
     WR1: QUERY(vs <* variable_semantics
          | SIZEOF(syntax_of(vs)) > 1) = [];
END_RULE; -- two_fold_variable_representation_rule
(*
```


Formal propositions:

WR1: a **variable_semantics** may only be represented by one **library_variable**.

1. ISO13584_library_expressions_schema function definitions

This clause defines the functions of the **ISO13584_library_expressions_schema**. The first two functions are not used within the **ISO13584_library_expressions_schema**. They are intended to facilitate the use of the twofold representation of variables that is used throughout this part of ISO 13584.

7.5.2 Syntax_of function

The **syntax_of** function computes the **generic_variable**s that are associated with a **variable_semantics** through an **environment**.

NOTE Due to the **two_fold_variable_representation_rule** rule (see the RULE clause of this schema), in the **ISO13584_library_expressions_schema**, a **variable_semantics** may be associated with no more than one **generic_variable**.

EXPRESS specification:

```
*)
FUNCTION syntax_of(sem: variable_semantics): SET OF generic_variable;

LOCAL
     env: BAG OF environment;
     vars: SET OF generic_variable;
END_LOCAL;

env := USEDIN(sem,
     'ISO13584_GENERIC_EXPRESSIONS_SCHEMA.ENVIRONMENT.SEMANTICS');
vars := [];

REPEAT i := LOINDEX(env) TO HIINDEX(env);
     vars := vars + env[i].syntactic_representation;
END_REPEAT;

RETURN(vars);

END_FUNCTION; -- syntax_of
(*
```

7.5.3 Semantics_of function

The **semantics_of** function computes the **variable_semantics** that is associated with a **generic_variable** through an **environment**.

EXPRESS specification:

```
*)
FUNCTION semantics_of(vars: generic_variable): variable_semantics;
```



```
RETURN(vars.interpretation.semantics);

END_FUNCTION; -- semantics_of
(*
```

7.5.4 Collects_assigned_properties function

The **collects_assigned_properties** function computes the **properties** that appears in the **prop_def** attribute of a **property_assignment** belonging to the **p_a** aggregate of **property_assignment**.

NOTE the value associated with such a property may be indeterminate

EXPRESS specification:

```
*)
FUNCTION collects_assigned_properties(
      p_a: AGGREGATE OF property_assignment): LIST OF property_BSU;

LOCAL
      assign_prop: LIST OF property_BSU;
      -- assigned properties of the
      -- multiple_arity_class_instance_constructor
END_LOCAL;

assign_prop := [];

REPEAT i := 1 TO SIZEOF(p_a);
      assign_prop := assign_prop + p_a[i].prop_def;
END_REPEAT;

RETURN(assign_prop);

END_FUNCTION; -- collects_assigned_properties
(*
```

7.5.5 Collects_referenced_library_expressions function

The function **collects_referenced_library_expressions** computes the **library_expression**s that are used in an aggregate of **property_assignment** to specify the values of some instance properties. The number of these **library_expression**s is used to define the arity of a **class_instance_constructor** operator.

EXPRESS specification:

```
*)
FUNCTION collects_referenced_library_expressions(
      p_a: AGGREGATE OF property_assignment)
      : LIST [1:?] OF library_expression;

LOCAL
      assign_exp: LIST [0:?] OF library_expression := [];
END_LOCAL;
```



```
REPEAT i := 1 TO SIZEOF(p_a);
      IF EXISTS(p_a[i].its_value)
      THEN
           assign_exp := assign_exp + p_a[i].its_value;
      END_IF;
END_REPEAT;

RETURN(assign_exp);

END_FUNCTION; -- collects_referenced_library_expressions
(*
```

7.5.6 Compatible_simple_type_and_expression function

The function **compatible_simple_type_and_expression** checks if the result of an **expression** (**expr**) is type compatible with the domain (**dom**) defined by a **property_or_data_type_BSU**. It returns a LOGICAL that is TRUE when they are compatible and FALSE when they are not. This function returns UNKNOWN when the required **basic_semantic_unit** definition is not available.

EXPRESS specification:

```
*)

FUNCTION compatible_simple_type_and_expression(
      dom: property_or_data_type_BSU; expr: expression): LOGICAL;

IF (data_type_typeof(dom) = [])
THEN
      RETURN(UNKNOWN);
END_IF;

IF (('ISO13584_EXPRESSIONS_SCHEMA.NUMERIC_EXPRESSION' IN TYPEOF(expr))
      AND is_int_expr(expr))
THEN
      IF (('ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_TYPE' IN
           data_type_typeof(dom)) OR
           (('ISO13584_IEC61360_DICTIONARY_SCHEMA.NUMBER_TYPE' IN
           data_type_typeof(dom))
           AND NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE' IN
           data_type_typeof(dom))))
      THEN
           RETURN(TRUE);
      ELSE
           RETURN(FALSE);
      END_IF;
END_IF;

IF ('ISO13584_EXPRESSIONS_SCHEMA.NUMERIC_EXPRESSION' IN TYPEOF(expr))
      AND NOT is_int_expr(expr)
THEN
      IF (('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE' IN
```



```
          data_type_typeof(dom)) OR
          (('ISO13584_IEC61360_DICTIONARY_SCHEMA.NUMBER_TYPE' IN
          data_type_typeof(dom))
          AND NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_TYPE' IN
          data_type_typeof(dom))))
     THEN
          RETURN(TRUE);
     ELSE
          RETURN(FALSE);
     END_IF;
END_IF;

IF ('ISO13584_EXPRESSIONS_SCHEMA.BOOLEAN_EXPRESSION' IN TYPEOF(expr))
THEN
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.BOOLEAN_TYPE' IN
          data_type_typeof(dom))
     THEN
          RETURN(TRUE);
     ELSE
          RETURN(FALSE);
     END_IF;
END_IF;

IF ('ISO13584_EXPRESSIONS_SCHEMA.STRING_EXPRESSION' IN TYPEOF(expr))
THEN
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.STRING_TYPE' IN
          data_type_typeof(dom))
     THEN
          RETURN(TRUE);
     ELSE
          RETURN(FALSE);
     END_IF;
END_IF; -- all simple types have been considered

END_FUNCTION; -- compatible_simple_type_and_expression
(*
```

7.5.7 Compatible_type_and_library_expression function

The function **compatible_type_and_library_expression** checks if the result of a **library_expression** (**expr**) is type compatible with the domain (**dom**) defined by a **property_or_data_type_BSU**. It returns a LOGICAL that is TRUE when they are compatible and FALSE when they are not. This function returns UNKNOWN when the required **basic_semantic_unit** definitions are not present.

EXPRESS specification:

```
*)
FUNCTION compatible_type_and_library_expression(
     dom: property_or_data_type_BSU;
     expr: library_expression): LOGICAL;

LOCAL
```



```
     temp: SET[0:1] OF class_BSU;
END_LOCAL;

IF (data_type_typeof(dom) = [])
THEN (* the final domain of the type is not available *)
     RETURN(UNKNOWN);
END_IF;

(* The following EXPRESS statements deal with level_spec_expression *)
IF ('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA.LEVEL_SPEC_EXPRESSION'
     IN TYPEOF(expr))
THEN
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.LEVEL_TYPE'
          IN data_type_typeof(dom))
          AND (list_to_set(data_type_level_spec(dom))
          = list_to_set(expr\level_spec_expression.levels))
          AND (TYPEOF(expr\level_spec_expression.value_type)
          <= data_type_level_value_typeof(dom))
     THEN
          RETURN(TRUE);
     ELSE
          RETURN(FALSE);
     END_IF;
END_IF;

(* The following EXPRESS statements deal with
     entity_instance_expression *)
IF ('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA.ENTITY_INSTANCE_EXPRESSION'
     IN TYPEOF(expr))
THEN
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.ENTITY_INSTANCE_TYPE'
          IN data_type_typeof(dom))
          AND (data_type_type_name(dom)
          <= expr\entity_instance_expression.type_name)
     THEN
          RETURN(TRUE);
     ELSE
          RETURN(FALSE);
     END_IF;
END_IF;

(* The following EXPRESS statements deal with
     class_instance_expression *)
IF ('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA.CLASS_INSTANCE_EXPRESSION'
     IN TYPEOF(expr))
THEN
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_INSTANCE_TYPE'
          IN data_type_typeof(dom)) AND
          (SIZEOF(data_type_class_of(dom)) = 1)
     THEN
          temp := data_type_class_of(dom);
```



```
          RETURN(compatible_class_and_class(
               temp[1], expr\class_instance_expression.expr_type));
     ELSE
          RETURN(FALSE);
     END_IF;
END_IF;

(* simple types *)
RETURN(compatible_simple_type_and_expression(dom, expr));

END_FUNCTION; -- compatible_type_and_library_expression
(*
```

7.5.8 Compatible_variable_and_expression function

The function **compatible_variable_and_expression** checks if the result of an **expression** (**expr**) is assignment compatible with a variable (**va**). It returns a LOGICAL that is TRUE when they are compatible and FALSE when they are not.

EXPRESS specification:

```
*)
FUNCTION compatible_variable_and_expression(va: generic_variable;
     expr: expression): LOGICAL;

IF ('ISO13584_EXPRESSIONS_SCHEMA.NUMERIC_EXPRESSION' IN TYPEOF(expr))
     AND is_int_expr(expr)
THEN
     IF ('ISO13584_EXPRESSIONS_SCHEMA.INT_NUMERIC_VARIABLE'
          IN TYPEOF(va))
     THEN
          RETURN(TRUE);
     ELSE
          RETURN(FALSE);
     END_IF;
END_IF;

IF ('ISO13584_EXPRESSIONS_SCHEMA.NUMERIC_EXPRESSION' IN TYPEOF(expr))

THEN
     IF ('ISO13584_EXPRESSIONS_SCHEMA.REAL_NUMERIC_VARIABLE'
          IN TYPEOF(va))
     THEN
          RETURN(TRUE);
     ELSE
          RETURN(FALSE);
     END_IF;
END_IF;

IF ('ISO13584_EXPRESSIONS_SCHEMA.BOOLEAN_EXPRESSION' IN TYPEOF(expr))
THEN
     IF('ISO13584_EXPRESSIONS_SCHEMA.BOOLEAN_VARIABLE' IN TYPEOF(va))
```



```
      THEN
           RETURN(TRUE);
      ELSE
           RETURN(FALSE);
      END_IF;
END_IF;

IF ('ISO13584_EXPRESSIONS_SCHEMA.STRING_EXPRESSION'
      IN TYPEOF(expr))
THEN
      IF ('ISO13584_EXPRESSIONS_SCHEMA.STRING_VARIABLE' IN TYPEOF(va))
      THEN
           RETURN(TRUE);
      ELSE
           RETURN(FALSE);
      END_IF;
END_IF;

-- all simple types have been considered
RETURN(UNKNOWN);

END_FUNCTION; -- compatible_variable_and_expression
(*
```

7.5.9 Compatible_variable_and_library_expression function

The function **compatible_variable_and_library_expression** checks if the result of a **library_expression** (**expr**) is assignment compatible with a **library_variable** (**va**). It returns a LOGICAL that is TRUE when they are compatible and FALSE when they are not. This function returns UNKNOWN when the required **basic_semantic_unit** definitions are not present.

EXPRESS specification:

```
*)
FUNCTION compatible_variable_and_library_expression(
      va: library_variable; expr: library_expression): LOGICAL;

(* The following express statements deal with level_spec_expression *)
IF ('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA.LEVEL_SPEC_EXPRESSION'
      IN TYPEOF(expr))
THEN
      IF (('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA.LEVEL_SPEC_VARIABLE'
           IN TYPEOF(va))
           AND ((list_to_set(va\level_spec_expression.levels))
           = list_to_set(expr\level_spec_expression.levels))
           AND (TYPEOF(va\level_spec_expression.value_type)
           <= TYPEOF(expr\level_spec_expression.value_type)))
      THEN
           RETURN(TRUE);
      ELSE
           RETURN(FALSE);
```



```
     END_IF;
END_IF;

(*The following express statements deals with
     entity_instance_expression *)
IF ('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA.ENTITY_INSTANCE_EXPRESSION'
     IN TYPEOF(expr))
THEN
     IF ('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA.ENTITY_INSTANCE_VARIABLE'
          IN TYPEOF(va))
          AND (va\entity_instance_expression.type_name
          <= expr\entity_instance_expression.type_name)
     THEN
          RETURN(TRUE);
     ELSE
          RETURN(FALSE);
     END_IF;
END_IF;

(* The following express statements deals with
     class_instance_expression *)
IF ('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA.CLASS_INSTANCE_EXPRESSION'
     IN TYPEOF(expr))
THEN
     IF ('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA.CLASS_INSTANCE_VARIABLE'
          IN TYPEOF(va))
     THEN
          RETURN(compatible_class_and_class(
               va\class_instance_expression.expr_type,
               expr\class_instance_expression.expr_type));
     ELSE
          RETURN(FALSE);
     END_IF;
END_IF;

(* simple types *)
RETURN(compatible_variable_and_expression(va, expr));

END_FUNCTION; -- compatible_variable_and_library_expression
(*

*)
END_SCHEMA; -- ISO13584_library_expressions_schema
(*
```

8 ISO13584_table_resource_schema

This clause defines the requirements for the **ISO13584_table_resource_schema**. The following EXPRESS declaration introduces the **ISO13584_library_content_schema** block and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA ISO13584_table_resource_schema;

REFERENCE FROM ISO13584_IEC61360_dictionary_schema
     (class_BSU,
     date_type,
     level,
     list_to_set,
     revision_len,
     revision_type,
     value_format_len,
     value_format_type);

REFERENCE FROM ISO13584_generic_expressions_schema
     (binary_generic_expression,
     generic_expression,
     generic_literal,
     generic_variable,
     multiple_arity_generic_expression,
     simple_generic_expression,
     unary_generic_expression,
     used_variables,
     variable_semantics);

REFERENCE FROM ISO13584_expressions_schema
     (boolean_defined_function,
     expression,
     is_sql_mappable,
     SQL_mappable_defined_function,
     variable);

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,
     null_or_boolean_value,
     null_or_complex_value,
     null_or_dic_class_instance,
     null_or_entity_instance_value,
     null_or_int_level_spec_value,
     null_or_integer_value,
     null_or_level_spec_value,
     null_or_number_value,
     null_or_primitive_value,
```



```
        null_or_real_level_spec_value,
        null_or_real_value,
        null_or_simple_value,
        null_or_translatable_string_value,
        number_value,
        primitive_value,
        real_level_spec_value,
        real_value,
        right_values_for_level_spec,
        same_translations,
        simple_value,
        string_value,
        translatable_string_value,
        translated_string_value);

    REFERENCE FROM ISO13584_library_expressions_schema
        (class_instance_expression,
        compatible_variable_and_expression,
        entity_instance_expression,
        entity_instance_variable,
        level_spec_expression,
        level_spec_variable,
        semantics_of,
        syntax_of);
    (*
```

NOTE The schemas referenced above can be found in the following documents:
ISO13584_IEC61360_dictionary_schema IEC 61360-2
(which is duplicated for convenience in informative annex D of ISO 13584-42),
ISO13584_generic_expressions_schema ISO 13584-20,
ISO13584_expressions_schema ISO 13584-20,
ISO13584_instance_resource_schema This part of ISO 13584,
ISO13584_library_expressions_schema This part of ISO 13584.

8.1 Introduction to the ISO13584_table_resource_schema

The **ISO13584_table_resource_schema** defines the set of resources needed to describe tables and algebraic operations in tables.

Tables are represented by a three level data model:

— the **table_identification** identifies a table; this entity enables reference to a table, whether the description and content of this table are or are not represented in the same exchange context;

— the **table_specification** describes a table; this entity specifies the columns of a table and its corresponding key; the table columns are specified by their semantics modelled as a subtype of **variable_semantics** referenced from ISO 13584-20: 1998;

— the **table_extension** defines the values of the columns representing the content of the table; the data type of the values of each column shall conform to the data type of any **generic_variable** that may be associated with the **variable_semantics** that specifies the corresponding column.

NOTE 1 **generic_variable** is defined in ISO 13584-20.

The algebraic operations in tables considered in the **ISO13584_table_resource_schema** include:

— relational algebra, and

— set operations.

The **ISO13584_table_resource_schema** models:

— the tables that may be stored in relational databases;

— the tables in which some columns contain instances of some EXPRESS entity data type;

— the characterisation of each column by its semantics;

— the content of each column as type compatible with any **generic_variable** that may be associated with the column;

— a table as the result of an expression in which the leaves are tables and the operators are either relational operators or set-theory operators;

— the key computation of the result of a table expression;

— a table variable.

The **ISO13584_table_resource_schema** does not model:

— how these tables are used or accessed.

NOTE 2 In the context of this part of ISO13584, tables are used to model the relations between the values of the properties associated with a class. This is done by subtyping the resources defined in this schema.

8.2 Fundamental concepts and assumptions for the ISO13584_table_resource_schema

8.2.1 Description of tables

The following assumptions apply to the portions of this schema that deal with table description:

a) a table is an ordered list of columns;

b) a subset of this list of columns constitutes the table key;

c) a column is associated with a particular semantic. In the same table, two different columns shall correspond to two different semantics;

d) a column contains an ordered list of values of the same type, or subtypes of a common type;

e) the length of all the columns of a table shall be the same;

g) the tuple of values obtained by selecting in each column the value located at the same position is called a line of the table;

h) a **simple_column** contains values of type BOOLEAN, REAL, INTEGER, or STRING;

i) a table intended to be stored in a relational database contains only **simple_column**s;

j) a **complex_column** is a column whose values are represented as EXPRESS ENTITY instances;

k) this schema provides resources for three types of **complex_column**s:

   1) **level_spec_column** enables one to represent an array of four optional values associated with a same semantics;

2) **entity_instance_column** enables one to represent values of which types are defined by an EXPRESS entity data type;

3 **class_instance_column** enables one to represent values of which types are defined by an ISO 13584-conformant specification of a class;

l) both **entity_instance_column** and **class_instance_column** provide for inheritance:

1) **entity_instance_column** may contain instances of various subtypes of the same entity data type;

2) **class_instance_column** may contain instances of various subclasses of the same class;

m) a table is represented by three entities:

1) the **table_identification**, that provides for referencing the table;

2) the **table_specification**, that specifies the semantics of the columns of a table and its corresponding key;

3) the **table_content**, that describes the values of the columns.

8.2.2 Description of table expressions

The following assumptions apply to the portions of this schema that deal with table expressions:

— Tables may be combined by operators to specify other tables. Such a table specification constitutes a **table_expression**.

— In a **table_expression**, a subset of the columns may be defined as a key: if one selects in each of these columns the value corresponding to the same position, this tuple is unique among all the tuples based on the same columns.

— Each column of a **table_expression** may be associated with a variable for which it defines the domain. The semantics of this variable is defined by the **table_expression** it refers to, and by the **variable_semantics** that identifies (uniquely) the column of this table.

EXAMPLE Such a variable may be used in a **select_expression** to perform a traversal of the column values.

NOTE The semantics of such a variable (see: **column_traversal_variable_semantics**) is different from the **variable_semantics** used to identify the column. The latter may stand for a well-defined value when the former just stands for an iterator on the different values of the column. In the SQL language, a variable associated with the latter would be a parameter-reference, a variable associated with the former would be a column-reference.

8.3 ISO13584_table_resource_schema entity definitions

These subclauses contain the EXPRESS entity definitions in the **ISO13584_table_resource_schema**.

8.3.1 Table_identification

A **table_identification** entity identifies a table. Reference to that entity enables reference to both the **table_specification** and the **table_extension** of the corresponding table, whether they are or are not represented in the same exchange context.

NOTE 1 This resource is generic in nature and enables one to represent any kind of table. It is intended to be subtyped for any particular kind of table.

NOTE 2 In this part of ISO 13584, tables are used to model tuples of values that are associated with properties that are visible in some ISO 13584-conformant classes. Such tables are identified by a **table_BSU** (see clause 10).

EXPRESS specification:

```
*)
ENTITY table_identification
ABSTRACT SUPERTYPE;
WHERE
     WR1: SIZEOF(USEDIN(SELF,
          'ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_SPECIFICATION'
          + '.TABLE_IDENTIFIER'))
          <= 1;
     WR2: SIZEOF(USEDIN(SELF,
          'ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_EXTENSION'
          + '.TABLE_IDENTIFIER')) <= 1;
END_ENTITY; -- table_identification
(*
```

Formal propositions:

WR1: the **table_identification** identifies a unique **table_specification** that refers to its **table_identification** through the **table_identifier** attribute;

NOTE 1 The **table_specification** may or may not be present in the same exchange context (see 5.5).

WR2: the **table_identification** identifies a unique **table_extension** that refers to its **table_identification** through the **table_identifier** attribute.

NOTE 2 The **table_extension** may or may not be present in the same exchange context (see 5.5).

8.3.2 Table_specification

A **table_specification** entity describes a table. A table is described in terms of the semantics of its columns. A subset of these columns defines the key of this table.

EXPRESS specification:

```
*)
ENTITY table_specification
SUPERTYPE OF(RDB_table_specification);
     table_identifier: table_identification;
     column_meaning: LIST[1:?] OF UNIQUE variable_semantics;
     key: SET[1:?] OF variable_semantics;
WHERE
     WR1: SELF.key <= list_to_set(SELF.column_meaning);
END_ENTITY; -- table_specification
(*
```

Attribute definitions:

table_identifier: the **table_identification** that identifies the currently specified table.

column_meaning: the list of columns descriptions that are contained in the table.

key: the subset of columns that defines the key of the currently described table.

Formal propositions:

WR1: the columns that define the key shall belong to the list of columns that define the table.

8.3.2.1 RDB_table_specification

The **RDB_table_specification** specialises a **table_specification** to define a relational database table.

EXPRESS specification:

```
*)
ENTITY RDB_table_specification
SUBTYPE OF(table_specification);
WHERE
     WR1: QUERY(col <* SELF\table_specification.column_meaning |
          QUERY(v <* syntax_of(col) |
          NOT('ISO13584_EXPRESSIONS_SCHEMA.VARIABLE'
               IN TYPEOF(v))) <> []) = [];
     WR2: (SIZEOF(USEDIN(SELF\table_specification.table_identifier,
          'ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_EXTENSION' +
          '.TABLE_IDENTIFIER')) = 0)
          OR ('ISO13584_TABLE_RESOURCE_SCHEMA.RDB_TABLE_EXTENSION'
          IN TYPEOF(USEDIN(SELF\table_specification.table_identifier,
          'ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_EXTENSION'+
          '.TABLE_IDENTIFIER')[1]));
END_ENTITY; -- RDB_table_specification
(*
```

Formal propositions:

WR1: the types of the columns shall be simple types; this is ensured by asserting that the variables that may be associated with the **column_meaning**s **variable_semantics** are simple **variable**s.

WR2: the content specification corresponding to a **RDB_table_specification** shall be a **RDB_table_extension**.

8.3.3 Table_extension

The **table_extension** entity defines the content of a given table. The content is given by the list of column values.

A **table_extension** has a **revision_of_content** attribute, that specifies the revision number of the set of values contained in the table, and a **content_revision_date** attribute that specifies when this set of values was defined. A revision shall neither change the set of key values in the table, nor change the list of **columns**. It may only change values of non-key **column**s.

NOTE 1 Changing the set of key values contained in a table and/or the list of columns that constitute its **content** defines a new table. The new table may be represented using a new **table_identification**.

EXPRESS specification:

```
*)
ENTITY table_extension
SUPERTYPE OF(rdb_table_extension);
      table_identifier: table_identification;
      content: LIST[1:?] OF UNIQUE column;
      revision_of_content: revision_type;
      content_revision_date: date_type;
WHERE
      WR1: QUERY(col <* SELF.content | SIZEOF(col.values) <>
           SIZEOF(SELF.content[1].values)) = [];
      WR2: (SIZEOF(USEDIN(SELF.table_identifier,
           'ISO13584_TABLE_RESOURCE_SCHEMA' +
           '.TABLE_SPECIFICATION.TABLE_IDENTIFIER')) = 0) OR
           compatible_list_variable_semantics_and_columns(
           USEDIN(SELF.table_identifier,
           'ISO13584_TABLE_RESOURCE_SCHEMA' +
           '.TABLE_SPECIFICATION.TABLE_IDENTIFIER')[1].
           column_meaning, SELF.content);
      WR3: (SIZEOF(USEDIN(SELF.table_identifier,
           'ISO13584_TABLE_RESOURCE_SCHEMA' +
           '.TABLE_SPECIFICATION.TABLE_IDENTIFIER')) = 0) OR
           no_null_values_in_key_columns(USEDIN(SELF.table_identifier,
           'ISO13584_TABLE_RESOURCE_SCHEMA' +
           '.TABLE_SPECIFICATION.TABLE_IDENTIFIER')[1].
           column_meaning, USEDIN(SELF.table_identifier,
           'ISO13584_TABLE_RESOURCE_SCHEMA' +
           '.TABLE_SPECIFICATION.TABLE_IDENTIFIER')[1].
           key, SELF.content);
      WR4: same_translations_for_table_extension(SELF.content);
END_ENTITY; -- table_extension
(*
```

Attribute definitions:

identified_by: attribute to reference the table for which the current entity describes the content.

content: the list of columns that are contained in the table. It represents the data extension of the table content described.

revision_of_content: the revision index of the current table content.

content_revision_date: the date of the last revision achieved on the table content.

Formal propositions:

WR1: the sizes of the columns are the same.

WR2: if the **table_specification** is available in the same exchange context, the data types of the **column**s shall be compatible with the data types of the **generic_variable**s that may be associated with the **variable_semantics** that define the columns meanings.

WR3: the values of the key shall not contain any null values. In other words, keys whose values are **null_value**s are not allowed.

WR4: all the values that are **translated_string_value**s shall be translated in the same language(s).

Informal propositions:

IP1: all the revisions that correspond to the same **table_identification** shall have the same set of key values and the same **content** LIST. The only possible change from revision to revision is correcting typos in values or providing more accurate values

NOTE 2 This informal proposition specifies the changes allowed from revision to revision.

8.3.3.1 RDB_table_extension

The **RDB_table_extension** entity describes the content of relational database tables. Such tables shall only contain values whose data types are **simple_type**s.

NOTE **simple_type** is defined in ISO 13584-42; it contains: number, real, integer, string and Boolean.

EXPRESS specification:

```
*)
ENTITY RDB_table_extension
SUBTYPE OF(table_extension);
     SELF\table_extension.content: LIST[1:?] OF UNIQUE simple_column;
WHERE
     WR1: (SIZEOF(USEDIN(SELF\table_extension.table_identifier,
          'ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_SPECIFICATION' +
          '.TABLE_IDENTIFIER')) = 0) OR
          ('ISO13584_TABLE_RESOURCE_SCHEMA.RDB_TABLE_SPECIFICATION'
          IN TYPEOF(USEDIN(SELF\table_extension.table_identifier,
          'ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_SPECIFICATION' +
          '.TABLE_IDENTIFIER')[1]));
END_ENTITY; -- RDB_table_extension
(*
```

Attribute definitions:

content: the list of **simple_column**s that constitutes the relational database table.

NOTE **simple_column**s are columns that contain values belonging to **simple_type**s.

Formal propositions:

WR1: the dictionary definition corresponding to a **RDB_table_extension** shall be a **RDB_table_specification**.

8.3.4 Column

A **column** is a list of values belonging to the same type.

EXPRESS specification:

```
*)
ENTITY column
ABSTRACT SUPERTYPE OF(ONEOF(simple_column, complex_column));
     values: LIST [1:?] OF null_or_primitive_value;
INVERSE
     belongs_to: table_extension FOR content;
END_ENTITY; -- column
(*
```

Attribute definitions:

values: the list of the values contained in a column. These values shall be **primitive_value**s.

belongs_to: the **table_extension** to which the column belongs.

8.3.5 Simple_column

A **simple_column** is a list of values belonging to the same **simple_type**.

EXPRESS specification:

```
*)
ENTITY simple_column
ABSTRACT SUPERTYPE OF(ONEOF(boolean_column, formatted_column))
SUBTYPE OF(column);
     SELF\column.values: LIST [1:?] OF null_or_simple_value;
END_ENTITY; -- simple_column
(*
```

8.3.5.1 Boolean_column

A **boolean_column** is a column of null or BOOLEAN values.

EXPRESS specification:

```
*)
ENTITY boolean_column
SUBTYPE OF(simple_column);
     SELF\column.values: LIST [1:?] OF null_or_boolean_value;
END_ENTITY; -- boolean_column
(*
```

8.3.5.2 Formatted_column

A **formatted_column** is a **column** whose **simple_value**s may be associated with a format specification.

EXPRESS specification:

```
*)
ENTITY formatted_column
ABSTRACT SUPERTYPE OF(ONEOF(number_column, string_column))
SUBTYPE OF(simple_column);
     value_format: OPTIONAL value_format_type;
END_ENTITY; -- formatted_column
(*
```

Attribute definitions:

value_format: the description, in the format of a STRING, of the way the values in the column are to be represented.

NOTE 1 The structure of a **value_format_type** is defined in part 42 of this International Standard. This structure is based on the provisions taken from ISO 6093 and ISO 9735.

NOTE 2 When **value_format** is not defined the lay out of the column depends on particular agreements between the sender and the receiver. If there is no particular agreement, the lay out is implementation dependent.

NOTE 3 When the **variable_semantics** that defines the meaning of the **column** is a **SELF_property_value_semantics** associated with a **property_BSU**, and when a **value_format** exits in the **formatted_column**, this **value_format** takes precedence over the **value_format** defined for the **property_BSU**.

NOTE 4 **SELF_property_value_semantics** in defined in clause 9.

8.3.5.3 Number_column

A **number_column** is a **column** of null values or NUMBERs that are associated with a format.

EXPRESS specification:

```
*)
ENTITY number_column
ABSTRACT SUPERTYPE OF(ONEOF(real_column, integer_column))
SUBTYPE OF(formatted_column);
     SELF\column.values: LIST [1:?] OF null_or_number_value;
END_ENTITY; -- number_column
(*
```

Attribute definitions:

values: the values contained in the **column**. The type of these values is specialised so as to contain only NUMBER elements.

8.3.5.4 Real_column

A **real_column** is a **column** of null values or REALs that are associated with a format.

EXPRESS specification:

```
*)
ENTITY real_column
```



```
SUBTYPE OF(number_column);
     SELF\column.values: LIST [1:?] OF null_or_real_value;
WHERE
     WR1: NOT(EXISTS(SELF\formatted_column.value_format)) OR(
          (SELF\formatted_column.value_format) LIKE 'NR2*');
END_ENTITY; -- real_column
(*
```

Formal propositions:

WR1: if the **value_format** is defined, it shall specify a REAL number format.

8.3.5.5 Integer_column

An **integer_column** is a **column** of null values or INTEGERs that are associated with a format.

EXPRESS specification:

```
*)
ENTITY integer_column
SUBTYPE OF(number_column);
     SELF\column.values: LIST [1:?] OF null_or_integer_value;
WHERE
     WR1: NOT(EXISTS(SELF\formatted_column.value_format)) OR(
          (SELF\formatted_column.value_format) LIKE 'NR1*');
END_ENTITY; -- integer_column
(*
```

Formal propositions:

WR1: if the **value_format** is defined, it shall specify an INTEGER number format.

8.3.5.6 String_column

A **string_column** is a **column** of null values or STRINGs that may be translated in various languages and that are associated with a format. When the table column is displayed, even if there exist several translations for each string, only one string shall be displayed for any value.

NOTE The function that chooses in any context the particular value to be displayed shall be implementation dependent and is outside the scope of this part of ISO 13584.

EXPRESS specification:

```
*)
ENTITY string_column
SUBTYPE OF(formatted_column);
     SELF\column.values: LIST [1:?] OF
               null_or_translatable_string_value;
WHERE
     WR1: NOT (EXISTS(SELF\formatted_column.value_format)) OR (
          (((SELF\formatted_column.value_format) LIKE 'A*')
          OR ((SELF\formatted_column.value_format) LIKE 'M*')
```



```
          OR ((SELF\formatted_column.value_format) LIKE 'N*')
          OR ((SELF\formatted_column.value_format) LIKE 'X*'))
          AND NOT((SELF\formatted_column.value_format) LIKE 'NR*'));
     WR2: same_translations_for_string_values(SELF\column.values);
END_ENTITY; -- string_column
(*
```

Formal propositions:

WR1: if the **value_format** is defined, it shall specify a STRING format.

WR2: if the column contains some **translated_string_value**s, all the column values shall be some **translated_string_value**s, and they shall all reference the same **present_translations**.

8.3.6 Complex_column

A **complex_column** is a list of values belonging to the same **complex_type** of which values are represented as EXPRESS ENTITY instances. A **complex_column** can not belong to an **RDB_table_extension**.

EXPRESS specification:

```
*)
ENTITY complex_column
ABSTRACT SUPERTYPE OF(ONEOF(level_spec_column,
     entity_instance_column,
     class_instance_column))
SUBTYPE OF(column);
     SELF\column.values: LIST [1:?] OF null_or_complex_value;
END_ENTITY; -- complex_column
(*
```

8.3.6.1 Level_spec_column

A **level_spec_column** is a list of **level_spec_value**s. According to the **instance_resource_schema**, a **level_spec_value** is an ARRAY of four optional numbers, that carries the additional meaning that:

— the first value corresponds to the minimum value of some property;

— the second value corresponds to the nominal value of some property;

— the third value corresponds to the typical value of some property;

— the fourth value corresponds to the maximum value of some property.

The four numbers shall be either INTEGER or REAL.

EXPRESS specification:

```
*)
ENTITY level_spec_column
ABSTRACT SUPERTYPE OF(ONEOF(
     int_level_spec_column, real_level_spec_column))
SUBTYPE OF(complex_column);
```



```
      levels: LIST [1:4] OF UNIQUE level;
      SELF\column.values: LIST [1:?] OF null_or_level_spec_value;
WHERE
      WR1: QUERY(inst <* SELF\column.values
           | NOT right_values_for_level_spec(SELF.levels, inst)) = [];
END_ENTITY; -- level_spec_column
(*
```

Attribute definitions:

levels: the **levels** that are intended to be associated with values in **level_spec_value**s assigned to this variable.

values: the values contained in the **column**. The type of these values is specialised so as to contain the level values of the column.

Formal propositions:

WR1: the values provided in each **level_spec_value** shall correspond to the values that are specified by the **level**s attribute.

8.3.6.1.1 Int_level_spec_column

An **int_level_spec_column** is a **column** of null values or **int_level_spec_value**.

EXPRESS specification:

```
*)
ENTITY int_level_spec_column
SUBTYPE OF(level_spec_column);
      SELF\column.values: LIST [1:?] OF null_or_int_level_spec_value;
END_ENTITY; -- int_level_spec_column
(*
```

8.3.6.1.2 Real_level_spec_column

A **real_level_spec_column** is a column of null values or **real_level_spec_value**.

EXPRESS specification:

```
*)
ENTITY real_level_spec_column
SUBTYPE OF(level_spec_column);
      SELF\column.values: LIST [1:?] OF null_or_real_level_spec_value;
END_ENTITY; -- real_level_spec_column
(*
```

8.3.6.2 Entity_instance_column

An **entity_instance_column** is a list of null values or **entity_instance_value**s that are instances of some EXPRESS ENTITY data type. Each **entity_instance_column** has a **type_name** attribute that

specifies the set of STRINGs that shall be contained in the result of the EXPRESS TYPEOF function applied to any value of the list.

NOTE Each view exchange protocol specifies which **entity_instance_value** are allowed in a library exchange context that refers to this view exchange protocol. Only these **entity_instance_value**s may belong to an **entity_instance_column**

EXPRESS specification:

```
*)
ENTITY entity_instance_column
SUBTYPE OF(complex_column);
      type_name: SET [1:?] OF STRING;
      SELF\column.values: LIST [1:?] OF null_or_entity_instance_value;
WHERE
      WR1: QUERY(inst <* SELF\column.values
           | (NOT('ISO13584_INSTANCE_RESOURCE_SCHEMA.NULL_VALUE'
           IN TYPEOF(inst)))
           AND (NOT(SELF.type_name <= TYPEOF(inst)))) = [];
END_ENTITY; -- entity_instance_column
(*
```

Attribute definitions:

type_name: set of strings containing the type names that shall be contained in the result of the EXPRESS TYPEOF function when applied to any entity in the column.

Formal propositions:

WR1: all the instances contained in the column shall belong to the entity data type defined by the **type_name** attribute, or to any of its subtype, or are **null_value**s.

8.3.6.3 Class_instance_column

A **class_instance_column** is a list of null values or **dic_class_instance**s that are instances of some **class** as defined by the **ISO13584_instance_resource_schema**. Each **class_instance_column** has a **class_ref** attribute, that specifies, through its **class_BSU**, the class to which any instance of the list shall belong. A **dic_class_instance** of the list may be an instance of this class or any of its subclasses.

EXPRESS specification:

```
*)
ENTITY class_instance_column
SUBTYPE OF(complex_column);
      class_ref: class_BSU;
      SELF\column.values: LIST [1:?] OF null_or_dic_class_instance;
WHERE
      WR1: QUERY(inst <* SELF\column.values
           | (NOT('ISO13584_INSTANCE_RESOURCE_SCHEMA.NULL_VALUE'
           IN TYPEOF(inst)))
           AND (NOT compatible_class_and_class(SELF.class_ref,
           inst\dic_class_instance.class_def))) = [];
```



```
END_ENTITY; -- class_instance_column
(*
```

Attribute definitions:

class_ref: the class for which the elements of the columns shall be instances.

Formal propositions:

WR1: all the instances contained in the column shall belong to the class defined by the **class_ref** attribute or are **null_value**s.

8.3.7 Table expressions

This subclause defines the resources needed to specify a table through operators applied to other tables.

The resource constructs provided in this subclause enable the following:

— to define expressions that allow the combination of basic tables into more complex ones without explicitly defining the values contained in the table produced;

— to define variables that are iterators on the different values of a column, and that may be used to define queries against **table_expression**s;

— to define a query language that allows the definition of a table through the query of any table expression produced by the operators defined above;

— to exchange these tables and the corresponding expressions.

The underlying structure of a table expression is a specialisation of a **generic_expression**. It defines a direct acyclic graph where the internal nodes represent operators, where the leaves are either tables or other **generic_expression**s, and that evaluates to a table.

In order to be conformant to most of the database languages, the following set of table operators is specified:

— relational operators: projection, selection, join and cartesian product;

— set operators: union, intersection and difference.

In the **ISO13584_table_resource_schema**, a column is identified by its semantics represented as a **variable_semantics**, and no capability of changing this semantics is provided. Therefore the **variable_semantics** of the different columns shall be different. This is documented in a where rule in **table_expression**.

NOTE 1 In database languages, changing the name of a column in the case of name clash is often possible. It is known as "renaming".

NOTE 2 The structure of the table expressions has been defined to allow the addition of new operators. The resources for such extensions are provided.

8.3.7.1 Table_expression

A **table_expression** entity defines an expression whose range is a table.

EXPRESS specification:

```
*)
ENTITY table_expression
ABSTRACT SUPERTYPE OF(ONEOF(simple_table_expression,
     unary_table_expression,
     binary_table_expression,
     multiple_arity_table_expression,
     select_expression))
SUBTYPE OF(generic_expression);
DERIVE
     its_columns: LIST[1:?] OF variable_semantics
          := collects_columns(SELF);
     the_key: SET[1:?] OF variable_semantics := return_key(SELF);
     is_SQL_mappable: LOGICAL
          := is_SQL_mappable_table_expression(SELF);
WHERE
     WR1: QUERY(sem <* its_columns
          | SIZEOF(QUERY(sem_2 <* its_columns
          | sem_2 :=: sem)) <> 1) = [];
END_ENTITY; -- table_expression
(*
```

Attribute definitions:

its_columns: the list of columns that define the table resulting from the current **table_expression**. It is computed by the **collects_columns** function.

the_key: the subset of columns that constitutes a key of the table. It is computed by the **return_key** function.

is_sql_mappable: a Boolean value that defines whether the current **table_expression** is mappable to SQL. It is computed by the **is_SQL_mappable_table_expression** function.

Formal propositions:

WR1: each column shall correspond to a different semantic.

8.3.7.2 Column_traversal_variable_semantics

A **column_traversal_variable_semantics** entity defines the semantics of a **generic_variable** intended to play the role of an iterator over the set of values of a column. This **variable_semantics** may be associated with a **generic_variable** by an **environment**.

NOTE 1 **variable_semantics**, **generic_variable** and **environment** are defined in ISO 13584-20.

NOTE 2 The **generic_variable**s associated with **column_traversal_variable_semantics** are used in a **select_expression** to specify the selection criteria.

EXPRESS specification:

```
*)
ENTITY column_traversal_variable_semantics
SUBTYPE OF(variable_semantics);
```



```
      ctxt: table_expression;
      domain: variable_semantics;
WHERE
      WR1: SELF.domain IN SELF.ctxt.its_columns;
      WR2: (SIZEOF(USEDIN(SELF,'ISO13584_GENERIC_EXPRESSIONS_SCHEMA'
           +'.ENVIRONMENT.SEMANTICS')) = 0)
           OR compatible_variable_semantics_and_expression(
           SELF.domain,USEDIN(SELF,'ISO13584_GENERIC_EXPRESSIONS_SCHEMA'
           +'.ENVIRONMENT.SEMANTICS')[1].syntactic_representation);
END_ENTITY; -- column_traversal_variable_semantics
(*
```

Attribute definitions:

ctxt: the **table_expression** of which a column is referenced.

domain: the **variable_semantics** that identifies the column that constitutes the domain of the **column_traversal_variable_semantics**.

Formal propositions:

WR1: the column shall belong to the **table_expression**.

WR2: any variable associated with a **column_traversal_variable_semantics** shall be compatible with the **variable_semantics** that identifies the column.

8.3.7.3 Unary_table_expression

A **unary_table_expression** is a unary operator whose range is a table.

EXPRESS specification:

```
*)
ENTITY unary_table_expression
ABSTRACT SUPERTYPE OF(projection_expression)
SUBTYPE OF(table_expression, unary_generic_expression);
      SELF\unary_generic_expression.operand: table_expression;
END_ENTITY; -- unary_table_expression
(*
```

8.3.7.4 Binary_table_expression

A **binary_table_expression** is a binary operator whose range is a table.

EXPRESS specification:

```
*)
ENTITY binary_table_expression
ABSTRACT SUPERTYPE OF(ONEOF(set_table_expression,
      natural_join_expression))
SUBTYPE OF(table_expression, binary_generic_expression);
```



```
     SELF\binary_generic_expression.operands:
          LIST [2:2] OF table_expression;
END_ENTITY; -- binary_table_expression
(*
```

8.3.7.5 Multiple_arity_table_expression

A **multiple_arity_table_expression** is a multiple arity operator whose range is a table.

EXPRESS specification:

```
*)
ENTITY multiple_arity_table_expression
ABSTRACT SUPERTYPE OF(multiple_arity_cartesian_product)
SUBTYPE OF(table_expression, multiple_arity_generic_expression);
     SELF\multiple_arity_generic_expression.operands:
          LIST [2:?] OF table_expression;
END_ENTITY; -- multiple_arity_table_expression
(*
```

8.3.7.6 Simple_table_expression

The **simple_table_expression** is either a **table_variable** or a **table_literal**.

EXPRESS specification:

```
*)
ENTITY simple_table_expression
ABSTRACT SUPERTYPE OF(ONEOF(
     table_variable,
     table_literal))
SUBTYPE OF(table_expression, simple_generic_expression);
END_ENTITY; -- simple_table_expression
(*
```

8.3.7.7 Table_variable

A **table_variable** is a variable that stands for a table. This table is characterised through the list of its columns and its key.

EXPRESS specification:

```
*)
ENTITY table_variable
SUPERTYPE OF(RDB_table_variable)
SUBTYPE OF(simple_table_expression, generic_variable);
     structure: LIST [1:?] OF variable_semantics;
     its_key: SET [1:?] OF variable_semantics;
WHERE
     WR1: SELF.its_key <= list_to_set(SELF.structure);
END_ENTITY; -- table_variable
(*
```


Formal propositions:

WR1: the key of a **table_variable** shall be a subset of its columns.

8.3.7.8 RDB_table_variable

A **RDB_table_variable** is a **table_variable** that stands for an SQL-mappable table.

EXPRESS specification:

```
*)
ENTITY RDB_table_variable
SUBTYPE OF(table_variable);
WHERE
     WR1: QUERY(col <* SELF\table_expression.its_columns |
          QUERY(v <* syntax_of(col) |
          NOT('ISO13584_EXPRESSIONS_SCHEMA.VARIABLE'
          IN TYPEOF(v))) <> []) = [];
END_ENTITY; -- RDB_table_variable
(*
```

Formal propositions:

WR1: the types of the columns shall be simple types.

NOTE This is ensured by asserting that the variables that may be associated with the **column_meanings variable_semantics** are simple **variable**s.

8.3.7.9 Table_literal

The **table_literal** entity defines a table constant. It allows a basic table to be involved in an expression.

EXPRESS specification:

```
*)
ENTITY table_literal
SUBTYPE OF(simple_table_expression, generic_literal);
     the_value: table_identification;
WHERE
     WR1: SIZEOF(USEDIN(SELF.the_value,
          'ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_SPECIFICATION'
          + '.TABLE_IDENTIFIER')) = 1;
END_ENTITY; -- table_literal
(*
```

Attribute definitions:

the_value: the **table_identification** that is associated to this literal.

Formal propositions:

WR1: the **table_specification** shall be available in the same exchange context.

8.3.7.10 Set_table_expression

A **set_table_expression** is a **binary_table_expression** associated with a set-theory operator.

NOTE Set-theory operators require that the structures of their table operands be the same.

EXPRESS specification:

```
*)
ENTITY set_table_expression
ABSTRACT SUPERTYPE OF(ONEOF(
     union_table_expression,
     intersect_table_expression,
     difference_table_expression))
SUBTYPE OF(binary_table_expression);
WHERE
     WR1: SELF\binary_generic_expression.operands[1]
          \table_expression.its_columns =
          SELF\binary_generic_expression.operands[2]
          \table_expression.its_columns;
END_ENTITY; -- set_table_expression
(*
```

Formal propositions:

WR1: the two operands of the **set_table_expression** shall have the same structure.

NOTE The structure of a **table_expression** is defined by the list of **variable_semantics** that represent semantics of the various columns.

8.3.7.11 Union_table_expression

A **union_table_expression** is a **set_table_expression** that carries the semantics of the set union of two tables.

EXPRESS specification:

```
*)
ENTITY union_table_expression
SUBTYPE OF(set_table_expression);
END_ENTITY; -- union_table_expression
(*
```

8.3.7.12 Intersect_table_expression

An **intersect_table_expression** is a **set_table_expression** that carries the semantics of the set intersection of two tables.

EXPRESS specification:

```
*)
ENTITY intersect_table_expression
SUBTYPE OF(set_table_expression);

END_ENTITY; -- intersect_table_expression
(*
```

8.3.7.13 Difference_table_expression

A **difference_table_expression** is a **set_table_expression** that carries the semantics of a set difference of two tables: all the tuples in the first **table_expression** that do not belong to the second **table_expression**.

EXPRESS specification:

```
*)
ENTITY difference_table_expression
SUBTYPE OF(set_table_expression);
END_ENTITY; -- difference_table_expression
(*
```

8.3.7.14 Multiple_arity_cartesian_product

A **multiple_arity_cartesian_product** is a **multiple_arity_table_expression** that carries the semantics of the cartesian product of a list of tables. The list of columns is the sequence of the column of the different tables in the cartesian product order. The key is the union of the keys of the different tables.

EXPRESS specification:

```
*)
ENTITY multiple_arity_cartesian_product
SUBTYPE OF(multiple_arity_table_expression);
     SELF\multiple_arity_generic_expression.operands:
          LIST [2:?] OF table_expression;
END_ENTITY; -- multiple_arity_cartesian_product
(*
```

8.3.7.15 In_RDB_table_boolean_expression

An **in_RDB_table_boolean_expression** entity is a **boolean_expression** that carries the semantics of the belongs-to operator. It is a **multiple_arity_generic_expression** whose first operand is a **table_expression**. The subsequent **operands** are **expression**s the evaluation of which would constitute a tuple. The **in_RDB_table_boolean_expression** evaluates to TRUE if the tuple corresponds to one line of the table, otherwise it evaluates to FALSE.

The **in_RDB_table_boolean_expression** is only defined for **table_expression** that are SQL mappable and whose **column**s correspond to simple types. The type of each **expression** shall be compatible with the type of the corresponding column and shall be SQL mappable. The **variable**s involved in these **expression**s shall not contain any column iterator associated with a **column_traversal_variable_semantics**.

An **in_RDB_table_boolean_expression** may in turn be involved in another **boolean_expression**. It is therefore defined as a subtype of **boolean_defined_function** and of **SQL_mappable_defined_function**.

EXPRESS specification:

```
*)
ENTITY in_RDB_table_boolean_expression
SUBTYPE OF(multiple_arity_generic_expression,
          boolean_defined_function,
          SQL_mappable_defined_function);
DERIVE
     from_table: generic_expression :=
          SELF\multiple_arity_generic_expression.operands[1];
     tuple: LIST[1:?] OF generic_expression
          := QUERY(element <* SELF\multiple_arity_generic_expression
          .operands | 'ISO13584_EXPRESSIONS_SCHEMA.EXPRESSION'
          IN TYPEOF(element));
WHERE
     WR1: ('ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_EXPRESSION'
          IN TYPEOF(from_table))
          AND is_sql_mappable_table_expression(from_table);
     WR2: SIZEOF(QUERY(simple_expr <*
          QUERY(expr <* SELF\multiple_arity_generic_expression.operands
          | ('ISO13584_EXPRESSIONS_SCHEMA.EXPRESSION'
          IN TYPEOF(expr))) | is_sql_mappable(simple_expr)))
          = SIZEOF(SELF\multiple_arity_generic_expression.operands)- 1;
     WR3: SIZEOF(from_table\table_expression.its_columns)
          = SIZEOF(tuple);
     WR4: compatible_list_variable_semantics_and_expressions
          (SELF.from_table\table_expression.its_columns, SELF.tuple);
     WR5: QUERY(e <* tuple | QUERY(v <* used_variables(e)
          | ('ISO13584_TABLE_RESOURCE_SCHEMA'
          + '.COLUMN_TRAVERSAL_VARIABLE_SEMANTICS')
          IN TYPEOF(v.interpretation.semantics)) <> []) = [];
END_ENTITY; -- in_RDB_table_boolean_expression
(*
```

Attribute definitions:

from_table: the **table_expression** to which the tuple shall belong.

tuple: the list of **expression**s of which evaluation results constitute the tuple to be searched in the **from_table** table expression.

Formal propositions:

WR1: the **from_table** shall be a **table_expression** that is SQL mappable.

WR2: all the other **generic_expression**s of the **multiple_arity_generic_expression** shall be simple **expression**s that are SQL-mappable.

WR3: the size of the tuple shall be equal to the number of columns of the **table_expression**.

WR4: each value of the tuple shall be type-compatible with the corresponding column, and each value shall correspond to a simple type.

WR5: no variable in the tuple shall be associated with a **column_traversal_variable_semantics**.

8.3.7.16 Select_expression

The **select_expression** entity carries the semantics of the select operator of the SQL language. It evaluates to a table where all the lines for which the **condition** evaluates to FALSE are removed.

NOTE 1 A rule enforces the **condition generic_expression** to be a **boolean_expression** entity data type.

The **select_expression** is only defined for cases in which the columns of the **from_table** attribute correspond to simple types. The list of columns and keys of the result are not modified.

NOTE 2 A rule enforces the **from_table generic_expression** to be a **table_expression** entity data type.

The **condition** may contain two kinds of variables:

— variables that stand for some previously defined values, and

— variables that constitute column iterators: they are associated with **column_traversal_variable_semantics** that reference the different columns of the current table expression.

NOTE 3 Variables that stand for previously defined values correspond to constants in an SQL statement.

NOTE 4 Column iterators correspond to column references in an SQL statement.

When the **condition** is evaluated against the different lines of the table, each variable associated with a **column_traversal_variable_semantics** is interpreted as the current value of the corresponding column. The other variables stand for their current values.

EXPRESS specification:

```
*)
ENTITY select_expression
SUBTYPE OF(table_expression, binary_generic_expression);
DERIVE
     from_table: generic_expression :=
          SELF\binary_generic_expression.operands[1];
     condition: generic_expression :=
          SELF\binary_generic_expression.operands[2];
WHERE
     WR1: 'ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_EXPRESSION'
          IN TYPEOF(SELF.from_table);
     WR2: 'ISO13584_EXPRESSIONS_SCHEMA.BOOLEAN_EXPRESSION'
          IN TYPEOF(SELF.condition);
     WR3: QUERY(va <* used_variables(SELF.condition) |
          NOT('ISO13584_EXPRESSIONS_SCHEMA.VARIABLE'
          IN TYPEOF(va))) = [];
     WR4: QUERY(v <* used_variables(SELF.condition) |
          (NOT(check_iterator_context(SELF, v)) OR NOT
```



```
          check_iterator_domain_uniqueness(SELF, v))) = [];
END_ENTITY; -- select_expression
(*
```

Attribute definitions:

from_table: the table expression in which the selection is performed.

condition: the condition that filters the data of the **from_table** expression.

Formal propositions:

WR1: the **from_table** shall be a **table_expression**.

WR2: the **condition** expression shall be a **boolean_expression**.

WR3: the **condition** expression shall only contain simple **variable**s.

WR4: the **generic_variable**s used in the **condition** expression that are associated with a **column_traversal_variable_semantics** shall correspond to different columns of the **from_table table_expression**.

8.3.7.17 Projection_expression

The **projection_expression** carries the semantics of a projection operation. It evaluates to a table containing only the selected columns. The order of the columns is not changed for those that are not removed. If some columns are removed from the key, the key becomes the complete set of columns.

EXPRESS specification:

```
*)
ENTITY projection_expression
SUBTYPE OF(unary_table_expression);
     argts_var: SET[1:?] OF variable_semantics;
DERIVE
     from_table: table_expression :=
          SELF\unary_table_expression.operand;
WHERE
     WR1: SELF.argts_var <= list_to_set(SELF\unary_table_expression.
          operand.its_columns);
END_ENTITY; -- projection_expression
(*
```

Attribute definitions:

argts_var: the **variable_semantics** on which the projection operates.

from_table: the table expression from which the projection is evaluated.

Formal propositions:

WR1: the **argts_var** set shall belong to the list of columns.

8.3.7.18 Natural_join_expression

The **natural_join_expression** carries the semantics of the natural joint operation in relational algebra. The two tables involved in the **natural_join_expression** shall contain one, or several, couples of columns that correspond to the same **variable_semantics**. The **natural_join_expression** evaluates to another table built by a succession of three operations:

— the cartesian product is built, then

— the result is filtered by removing all the lines for which two columns corresponding to the same **variable_semantics** contain different values, then

— for each pair of columns corresponding to the same semantics, the second column is removed from the previous result.

The key of the resulting table is the union of the keys of the two operands.

EXPRESS specification:

```
*)
ENTITY natural_join_expression
SUBTYPE OF(binary_table_expression);
DERIVE
     table_1: table_expression :=
          SELF\binary_table_expression.operands[1];
     table_2: table_expression :=
          SELF\binary_table_expression.operands[2];
WHERE
     WR1: SIZEOF(list_to_set(table_1\table_expression.its_columns) *
          list_to_set(table_2\table_expression.its_columns)) > 0;
END_ENTITY; -- natural_join_expression
(*
```

Attribute definitions:

table1: the **table_expression** that is the first argument of the join operation.

table2: the **table_expression** that is the second argument of the join operation.

Formal propositions:

WR1: there exists at least one common column in the two **table_expression** operands.

8.4 ISO13584_table_resource_schema functions definition

This section describes the functions required to perform type control for table content.

8.4.1 Compatible_column_and_variable function

The function **compatible_column_and_variable** checks if a **generic_variable** (**v**) is type compatible with the type defined by a **column** (**col**). It returns a LOGICAL that is TRUE when they are compatible and FALSE when they are not. It returns UNKNOWN when no **generic_variable** is associated to **v**.

EXPRESS specification:

```
*)
FUNCTION compatible_column_and_variable(col: column;
     v: generic_variable): LOGICAL;

(* The following express statements deal with simple types *)

IF ('ISO13584_EXPRESSIONS_SCHEMA.INT_NUMERIC_VARIABLE' IN TYPEOF(v))
THEN
     RETURN(('ISO13584_TABLE_RESOURCE_SCHEMA.INTEGER_COLUMN'
          IN TYPEOF(col)) OR
          (('ISO13584_TABLE_RESOURCE_SCHEMA.NUMBER_COLUMN'
          IN TYPEOF(col))
          AND NOT('ISO13584_TABLE_RESOURCE_SCHEMA.REAL_COLUMN'
          IN TYPEOF(col))));
END_IF;

IF ('ISO13584_EXPRESSIONS_SCHEMA.REAL_NUMERIC_VARIABLE' IN TYPEOF(v))
THEN
     RETURN(('ISO13584_TABLE_RESOURCE_SCHEMA.REAL_COLUMN'
          IN TYPEOF(col))
          OR (('ISO13584_TABLE_RESOURCE_SCHEMA.NUMBER_COLUMN'
          IN TYPEOF(col))
          AND NOT('ISO13584_TABLE_RESOURCE_SCHEMA.INTEGER_COLUMN'
          IN TYPEOF(col))));
END_IF;

IF ('ISO13584_EXPRESSIONS_SCHEMA.BOOLEAN_VARIABLE' IN TYPEOF(v))
THEN
     RETURN('ISO13584_TABLE_RESOURCE_SCHEMA.BOOLEAN_COLUMN'
     IN TYPEOF(col));
END_IF;

IF ('ISO13584_EXPRESSIONS_SCHEMA.STRING_VARIABLE' IN TYPEOF(v))
THEN
     RETURN('ISO13584_TABLE_RESOURCE_SCHEMA.STRING_COLUMN'
     IN TYPEOF(col));
END_IF;

(* The following express statements deal with complex types *)

IF ('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA.ENTITY_INSTANCE_VARIABLE')
     IN TYPEOF(v)
THEN
     RETURN(('ISO13584_TABLE_RESOURCE_SCHEMA.ENTITY_INSTANCE_COLUMN'
          IN TYPEOF(col))
          AND (v\entity_instance_expression.type_name
                    <= col\entity_instance_column.type_name));
END_IF;

IF ('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA.CLASS_INSTANCE_VARIABLE')
```

```
      IN TYPEOF(v)
THEN
      IF ('ISO13584_TABLE_RESOURCE_SCHEMA.CLASS_INSTANCE_COLUMN'
           IN TYPEOF(col))
      THEN
           RETURN(compatible_class_and_class
                (v\class_instance_expression.expr_type,
                col\class_instance_column.class_ref));
      ELSE
           RETURN(FALSE);
      END_IF;
END_IF;

IF ('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA.LEVEL_SPEC_VARIABLE')
      IN TYPEOF(v) THEN
      IF ('ISO13584_TABLE_RESOURCE_SCHEMA.LEVEL_SPEC_COLUMN'
           IN TYPEOF(col))
      THEN
           IF (list_to_set(v\level_spec_expression.levels)
                = list_to_set(col\level_spec_column.levels))
           THEN
                IF (('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA'
                     +'.INT_LEVEL_SPEC_VARIABLE' IN TYPEOF(v))
                     AND NOT('ISO13584_TABLE_RESOURCE_SCHEMA' +
                     '.INT_LEVEL_SPEC_COLUMN' IN TYPEOF(col)))
                THEN
                     RETURN(FALSE);
                END_IF;
                IF (('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA'
                     +'.REAL_LEVEL_SPEC_VARIABLE' IN TYPEOF(v))
                     AND NOT('ISO13584_TABLE_RESOURCE_SCHEMA' +
                     '.REAL_LEVEL_SPEC_COLUMN' IN TYPEOF(col)))
                THEN
                     RETURN(FALSE);
                END_IF;
                RETURN(TRUE);
           ELSE
                RETURN(FALSE);
           END_IF;
      ELSE
           RETURN(FALSE);
      END_IF;
END_IF;

RETURN(FALSE);

END_FUNCTION; -- compatible_column_and_variable
(*
```


8.4.2 Compatible_column_and_variable_semantics function

The function **compatible_column_and_variable_semantics** function checks if all the **generic_variable**s that are associated with a **variable_semantics** (**sem**) are type compatible with the type defined by a **column** (**col**). It returns a LOGICAL that is TRUE when they are compatible and FALSE when they are not. This function returns UNKNOWN when no **generic_variable** is associated to **sem**.

EXPRESS specification:

```
*)
FUNCTION compatible_column_and_variable_semantics(col: column;
     sem: variable_semantics): LOGICAL;

LOCAL
     va: SET OF generic_variable;
END_LOCAL;

va := syntax_of(sem);
IF SIZEOF(va) = 0
THEN
     RETURN(UNKNOWN);
ELSE
     REPEAT i := LOINDEX(va) TO HIINDEX(va);
          IF (NOT compatible_column_and_variable(col, va[i]))
          THEN
               RETURN(FALSE);
          END_IF;
     END_REPEAT;
     RETURN(TRUE);
END_IF;

END_FUNCTION; -- compatible_column_and_variable_semantics
(*
```

8.4.3 Compatible_list_variable_semantics_and_columns function

The function **compatible_list_variable_semantics_and_columns** checks if a list of **columns** (**col**) is type compatible with a list of **variable_semantics** (**sem**). It checks if the **generic_variable**s that may be associated with these **variable_semantics** are type compatible with the columns. If some **variable_semantics** is not associated with any **generic_variable**, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION compatible_list_variable_semantics_and_columns(
     sem: LIST [0:?] of variable_semantics;
     col: LIST [0:?] of column): LOGICAL;

LOCAL
     res: LOGICAL;
END_LOCAL;
```

```
IF SIZEOF(sem) <> SIZEOF(col)
THEN
     RETURN(FALSE);
END_IF;

res := TRUE;

REPEAT i := LOINDEX(col) TO HIINDEX(col);
     res := res AND
          compatible_column_and_variable_semantics(col[i], sem[i]);
END_REPEAT;

RETURN(res);

END_FUNCTION; -- compatible_list_variable_semantics_and_columns
(*
```

8.4.4 Compatible_variable_semantics_and_expression function

The function **compatible_variable_semantics_and_expression** function checks if all the **variable**s that are associated with a **variable**s_**semantics** (**sem**) are type compatible with the type defined by an **expression** (**expr**). It returns a LOGICAL that is TRUE when they are compatible and FALSE when they are not. This function returns UNKNOWN when no **variable** is associated to **sem**.

EXPRESS specification:

```
*)
FUNCTION compatible_variable_semantics_and_expression(
     sem: variable_semantics; expr: expression): LOGICAL;

LOCAL
     va: SET OF generic_variable;
END_LOCAL;

va := syntax_of(sem);
IF SIZEOF(va) = 0
THEN
     RETURN(UNKNOWN);
ELSE
     REPEAT i := LOINDEX(va) TO HIINDEX(va);
          IF (NOT compatible_variable_and_expression(va[i], expr))
          THEN
               RETURN(FALSE);
          END_IF;
     END_REPEAT;
     RETURN(TRUE);
END_IF;

END_FUNCTION; -- compatible_variable_semantics_and_expression
(*
```


8.4.5 Compatible_list_variable_semantics_and_expressions function

The function **compatible_list_variable_semantics_and_expressions** checks if a list of **variable_semantics** (**sem**) is type compatible with a list of **expression** (**exprs**). It checks if all the **variable**s that may be associated with these **variable_semantics** are type compatible with the corresponding **expression**s. If some **variable_semantics** is not associated with any **variable**, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION compatible_list_variable_semantics_and_expressions(
     sem: LIST [0:?] of variable_semantics;
     exprs: LIST [0:?] of expression): LOGICAL;

LOCAL
     res: LOGICAL;
END_LOCAL;

IF SIZEOF(sem) <> SIZEOF(exprs)
THEN
     RETURN(FALSE);
END_IF;

res := TRUE;

REPEAT i := 1 TO SIZEOF(sem);
     res := res AND
     compatible_variable_semantics_and_expression(sem[i], exprs[i]);
END_REPEAT;

RETURN(res);

END_FUNCTION; -- compatible_list_variable_semantics_and_expressions
(*
```

8.4.6 Collects_columns function

The **collects_columns** function retrieves the list of **variable_semantics** that occur, as columns, in a **table_expression**. It performs a traversal of the tree obtained in the **table_expression** passed parameter.

EXPRESS specification:

```
*)
FUNCTION collects_columns(t: table_expression):
     LIST OF variable_semantics;

LOCAL
     res, tempo: LIST [0:?] OF variable_semantics := [];
     x: BAG [1:1] OF table_specification;
END_LOCAL;
```

```
IF ('ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_VARIABLE'IN TYPEOF(t))
THEN
     RETURN(t\table_variable.structure);
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_LITERAL' IN TYPEOF(t))
THEN
     x := USEDIN(t\table_literal.the_value,
          'ISO13584_TABLE_RESOURCE_SCHEMA'
          + '.TABLE_SPECIFICATION.TABLE_IDENTIFIER');
     RETURN(x[1].column_meaning);
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.NATURAL_JOIN_EXPRESSION'
     IN TYPEOF(t))
THEN
     RETURN(
     diff_columns(collects_columns(
          t\natural_join_expression.table_1),
          collects_columns(
          t\natural_join_expression.table_2)));
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.BINARY_TABLE_EXPRESSION'
     IN TYPEOF(t))
THEN
     RETURN(collects_columns(t\binary_table_expression.operands[1])+
     collects_columns(t\binary_table_expression.operands[2]));
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.MULTIPLE_ARITY_TABLE_EXPRESSION')
     IN TYPEOF(t) THEN
     REPEAT i := 1 TO
               SIZEOF(t\multiple_arity_table_expression.operands);
          res := res +
               collects_columns(t\multiple_arity_table_expression
               .operands[i]);
     END_REPEAT;
     RETURN(res);
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.SELECT_EXPRESSION' IN TYPEOF(t))
THEN
     RETURN(collects_columns(
          t\binary_generic_expression.operands[1]));
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.PROJECTION_EXPRESSION'
     IN TYPEOF(t))
THEN
```



```
      REPEAT i := 1 TO SIZEOF(t\projection_expression.argts_var);
           tempo := tempo +t\projection_expression.argts_var[i];
      END_REPEAT;
      RETURN(tempo);
END_IF;

RETURN([]);

END_FUNCTION; -- collects_columns
(*
```

8.4.7 Diff_columns function

The **diff_columns** function takes two lists l1 and l2 of **variable_semantics** and returns the list l1 of **variable_semantics** augmented by the **variable_semantics** of l2 that do not belong to l1.

EXPRESS specification:

```
*)
FUNCTION diff_columns(l1, l2:LIST [1:?] OF variable_semantics):
      LIST [1:?] OF variable_semantics;

REPEAT i := 1 TO SIZEOF(l2);
      IF NOT(l2[i] IN l1)
      THEN
           l1 := l1 + l2[i];
      END_IF;
END_REPEAT;

RETURN(l1);
END_FUNCTION; -- diff_columns
(*
```

8.4.8 Return_key function

The **return_key** function computes a key of a **table_expression**. It performs a traversal of the tree obtained in the **table_expression** passed parameter. This key is defined by the following rules:

— the key of a **table_variable** is given by the key of the referenced **table_expression**,

— the key of a **table_literal** is given by the **table_identification** that shall be referenced,

— the key of a **select_expression** is given by the key of the table in which the selection is performed,

— the key of a **projection_expression** is computed as follows:

  a) if the projection does not remove any **variable_semantics** belonging to the key, the key is the key of the projected table,

  b) if the projection removes at least one **variable_semantics** of the key of the projected table, the key of the table is defined by all the **variable_semantics** on which the projection is performed,

— the key of a **intersect_table_expression** or a **difference_table_expression** is given by the key of the first table involved in the set expression,

— the key of a **union_table_expression** or a **natural_join_expression** is given by the set-union of the keys of the two tables that are involved in the binary expression,

— the key of a **cartesian_product** expression is given by the sum of all the keys that define the tables that are involved in the product.

EXPRESS specification:

```
*)
FUNCTION return_key(t: table_expression):
     SET [1:?] OF variable_semantics;

LOCAL
     res: SET [0:?] OF variable_semantics := [];
     x: BAG[1:1] OF table_specification;
END_LOCAL;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_VARIABLE'IN TYPEOF(t))
THEN
     RETURN(t\table_variable.its_key);
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_LITERAL' IN TYPEOF(t))
THEN
     x := USEDIN(t\table_literal.the_value,
          'ISO13584_TABLE_RESOURCE_SCHEMA.'
          + 'TABLE_SPECIFICATION.TABLE_IDENTIFIER');
     RETURN(x[1]\table_specification.key);
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.SELECT_EXPRESSION' IN TYPEOF(t))
THEN
     RETURN(return_key(t\binary_generic_expression.operands[1]));
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.PROJECTION_EXPRESSION'
     IN TYPEOF(t))
THEN
     IF QUERY(col <* t\projection_expression.from_table.the_key | NOT
          (col IN t\projection_expression.argts_var)) <> []
     THEN
          RETURN(list_to_set(t\table_expression.its_columns));
     ELSE
          RETURN(t\projection_expression.from_table.the_key);
     END_IF;
END_IF;

IF (('ISO13584_TABLE_RESOURCE_SCHEMA.INTERSECT_TABLE_EXPRESSION'
          IN TYPEOF(t))
```



```
      OR
      ('ISO13584_TABLE_RESOURCE_SCHEMA.DIFFERENCE_TABLE_EXPRESSION'
           IN TYPEOF(t)))
THEN
      RETURN(return_key(t\binary_table_expression.operands[1]));
END_IF;

IF (('ISO13584_TABLE_RESOURCE_SCHEMA.UNION_TABLE_EXPRESSION'
           IN TYPEOF(t))
      OR
      ('ISO13584_TABLE_RESOURCE_SCHEMA.NATURAL_JOIN_EXPRESSION'
           IN TYPEOF(t)))
THEN
      RETURN(return_key(t\binary_table_expression.operands[1]) +
      return_key(t\binary_table_expression.operands[2]));
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.MULTIPLE_ARITY_TABLE_EXPRESSION'
      IN TYPEOF(t))
THEN
      REPEAT i := 1 TO
      SIZEOF(t\multiple_arity_table_expression.operands);
           res := res
           + return_key(t\multiple_arity_table_expression.operands[i]);
      END_REPEAT;
      RETURN(res);
END_IF;

RETURN([]);

END_FUNCTION; -- return_key
(*
```

8.4.9 Is_SQL_mappable_table_expression function

The **is_SQL_mappable_table_expression** function checks if the acyclic graph that represents a **table_expression** only contains elements that are mappable to SQL.

EXPRESS Specification:

```
*)
FUNCTION is_SQL_mappable_table_expression(
      arg: table_expression): LOGICAL;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.SIMPLE_TABLE_EXPRESSION'
      IN TYPEOF(arg))
THEN
      IF ('ISO13584_TABLE_RESOURCE_SCHEMA.RDB_TABLE_VARIABLE'
           IN TYPEOF(arg))
      THEN
           RETURN(TRUE);
      END_IF;
```

```
      IF ('ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_LITERAL'
           IN TYPEOF(arg))
      THEN
           IF (SIZEOF(USEDIN(arg\table_literal.the_value,
                'ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_SPECIFICATION'
                + '.TABLE_IDENTIFIER')) = 1)
           THEN
                RETURN(('ISO13584_TABLE_RESOURCE_SCHEMA'
                     + '.RDB_TABLE_SPECIFICATION')
                     IN TYPEOF(USEDIN(arg\table_literal.the_value,
                     'ISO13584_TABLE_RESOURCE_SCHEMA'
                     + '.TABLE_SPECIFICATION.TABLE_IDENTIFIER')[1]));
           ELSE
                RETURN(UNKNOWN);
           END_IF;
      END_IF;
      RETURN(FALSE); -- table_variable that is not RDB_table_variable
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.UNARY_TABLE_EXPRESSION'
      IN TYPEOF(arg))
THEN
      RETURN(is_SQL_mappable_table_expression(
           arg\unary_table_expression.operand));
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.SELECT_EXPRESSION'
      IN TYPEOF(arg))
THEN
      RETURN(is_SQL_mappable_table_expression(
           arg\select_expression.from_table)
           AND is_SQL_mappable(arg\select_expression.condition));
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.BINARY_TABLE_EXPRESSION'
      IN TYPEOF(arg))
THEN
      RETURN(is_SQL_mappable_table_expression(
           arg\binary_table_expression.operands[1])
           AND Is_SQL_mappable_table_expression(
           arg\binary_table_expression.operands[2]));
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.MULTIPLE_ARITY_TABLE_EXPRESSION'
      IN TYPEOF(arg))
THEN
      REPEAT i := 1 TO SIZEOF
           (arg\multiple_arity_table_expression.operands);
           IF NOT is_SQL_mappable_table_expression
                (arg\multiple_arity_table_expression.operands[i])
           THEN
```



```
                RETURN(FALSE);
            END_IF;
        END_REPEAT;

        RETURN(TRUE);
    END_IF;

    RETURN(UNKNOWN);

    END_FUNCTION; -- is_SQL_mappable_table_expression
    (*
```

8.4.10 Used_table_literals function

The **used_table_literals** function performs a traversal of the generic expression graph and returns the **table_identification**s of all the **table_literal**s used.

EXPRESS Specification:

```
*)
FUNCTION used_table_literals(arg: generic_expression):
     SET OF table_identification;

LOCAL
     result: SET OF table_identification := [];
END_LOCAL;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.TABLE_LITERAL' IN TYPEOF(arg))
THEN
     RETURN([arg\table_literal.the_value]);
END_IF;

IF ('ISO13584_GENERIC_EXPRESSIONS_SCHEMA.UNARY_GENERIC_EXPRESSION')
     IN TYPEOF(arg)
THEN
     RETURN(used_table_literals(
          arg\unary_generic_expression.operand));
END_IF;

IF ('ISO13584_GENERIC_EXPRESSIONS_SCHEMA.BINARY_GENERIC_EXPRESSION'
     IN TYPEOF(arg))
THEN
     RETURN(used_table_literals(arg\binary_generic_expression
          .operands[1]) + used_table_literals(
          arg\binary_generic_expression.operands[2]));
END_IF;

IF ('ISO13584_GENERIC_EXPRESSIONS_SCHEMA.' +
     'MULTIPLE_ARITY_GENERIC_EXPRESSION' IN TYPEOF(arg))
THEN
     REPEAT i := 1 TO
          SIZEOF(arg\multiple_arity_generic_expression.operands);
```



```
            result := result + used_table_literals(
                 arg\multiple_arity_generic_expression.operands[i]);
       END_REPEAT;
       RETURN(result);
  END_IF;

  RETURN([]);

  END_FUNCTION; -- used_table_literals
  (*
```

8.4.11 Check_iterator_context function

The **check_iterator_context** function checks, in the case of a **generic_variable** associated with a **column_traversal_variable_semantics**, that this **column_traversal_variable_semantics** context is equal to the table expression in which the selection is performed.

EXPRESS specification:

```
  *)
  FUNCTION check_iterator_context(expr: select_expression;
       v: generic_variable): LOGICAL;

  IF ('ISO13584_TABLE_RESOURCE_SCHEMA' +
       '.COLUMN_TRAVERSAL_VARIABLE_SEMANTICS' IN
       TYPEOF(v.interpretation.semantics))
  THEN
       IF (v.interpretation.semantics\
            column_traversal_variable_semantics.ctxt:<>: expr.from_table)
       THEN
            RETURN(FALSE);
       ELSE
            RETURN(TRUE);
       END_IF;
  ELSE
       RETURN(UNKNOWN);
  END_IF;

  END_FUNCTION; -- check_iterator_context
  (*
```

8.4.12 Check_iterator_domain_uniqueness function

The **check_iterator_domain_uniqueness** function checks if a **generic_variable**, used in a **select_expression condition** expression, that is associated with a **column_traversal_variable_semantics** corresponds to a unique column of the **select_expression from_table table_expression**.

EXPRESS specification:

```
  *)
  FUNCTION check_iterator_domain_uniqueness(expr: select_expression;
```



```
      v: generic_variable): LOGICAL;

LOCAL
      vars: SET OF generic_variable := [];
      res: SET OF generic_variable := [];
END_LOCAL;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.'
      + 'COLUMN_TRAVERSAL_VARIABLE_SEMANTICS' IN
      TYPEOF(v.interpretation.semantics))
THEN
      vars := used_variables(expr.condition);

      REPEAT i := 1 TO SIZEOF(vars);
           IF ('ISO13584_TABLE_RESOURCE_SCHEMA' +
                '.COLUMN_TRAVERSAL_VARIABLE_SEMANTICS' IN
                TYPEOF(vars[i].interpretation.semantics))
           THEN
                IF (vars[i].interpretation.semantics\
                     column_traversal_variable_semantics.domain
                     :=: v.interpretation.semantics\
                     column_traversal_variable_semantics.domain)
                THEN
                     res := res + vars[i];
                END_IF;
           END_IF;
      END_REPEAT;

      RETURN(SIZEOF(res) = 1);
END_IF;

RETURN(UNKNOWN);

END_FUNCTION; -- check_iterator_domain_uniqueness
(*
```

8.4.13 No_null_values_in_key_columns function

The **no_null_values_in_key_coulumns** function returns TRUE if no null values are defined for each key column. Otherwise, it returns FALSE.

EXPRESS specification:

```
*)
FUNCTION no_null_values_in_key_columns(
      the_columns: LIST[1:?] OF variable_semantics;
      the_key: SET[1:?] OF variable_semantics;
      the_values: LIST[1:?] OF column): BOOLEAN;
LOCAL
      sem: variable_semantics;
      result: BOOLEAN := TRUE;
END_LOCAL;
```



```
REPEAT i := 1 TO SIZEOF(the_columns);
     sem := the_columns[i];
     IF(sem IN the_key) THEN
          REPEAT j := 1 TO SIZEOF(the_values[i].values);
               IF('ISO13584_INSTANCE_RESOURCE_SCHEMA' +
                    '.NULL_VALUE' IN TYPEOF(the_values[i].values[j]))
               THEN
                    result := FALSE;
               END_IF;
          END_REPEAT;
     END_IF;
END_REPEAT;

RETURN(result);

END_FUNCTION; -- no_null_values_in_key_columns
(*
```

8.4.14 Same_translations_for_string_values function

The **same_translations_for_string_values** function returns TRUE if all the **null_or_translatable_string_value**s are translated in the same language. It returns FALSE if the **string_values** aggregate contains translated and not translated strings, or if the translation are not given in the same language. It returns UNKNOWN if the **string_values** are not translated.

EXPRESS specification:

```
*)
FUNCTION same_translations_for_string_values(string_values:
     LIST OF null_or_translatable_string_value): LOGICAL;
LOCAL
     translated_values: SET OF translated_string_value := [];
     not_translated_values: LIST OF string_value := [];
END_LOCAL;

     translated_values := list_to_set(QUERY(val <* string_values |
          'ISO13584_INSTANCE_RESOURCE_SCHEMA.TRANSLATED_STRING_VALUE'
          IN TYPEOF(val)));
     not_translated_values := QUERY(val <* string_values |
          'ISO13584_INSTANCE_RESOURCE_SCHEMA.STRING_VALUE'
          IN TYPEOF(val));

     IF (SIZEOF(translated_values) <> 0)
     THEN
          IF (SIZEOF(not_translated_values) <> 0)
          THEN
               RETURN(FALSE);
          ELSE
               RETURN(same_translations(translated_values));
          END_IF;
     ELSE
```



```
          RETURN(UNKNOWN);
     END_IF;
END_FUNCTION; -- same_translations_for_string_values
(*
```

8.4.15 Same_translations_for_table_extension function

The **same_translations_for_table_extension** returns TRUE if, for each tuple that defines the table **content**, the same translation is used for defining the possible **translated_string_value**s. Otherwise it returns FALSE.

EXPRESS specification:

```
*)
FUNCTION same_translations_for_table_extension(
     content: LIST[1:?] OF column): BOOLEAN;
LOCAL
     translated_values: SET OF translated_string_value := [];
END_LOCAL;

REPEAT i := 1 to SIZEOF(content[1].values);
     translated_values :=
          get_translated_string_values_of_tuple(content, i);
     IF NOT(same_translations(translated_values))
     THEN
          RETURN(FALSE);
     END_IF;
END_REPEAT;

RETURN(TRUE);

END_FUNCTION; -- same_translations_for_table_extension
(*
```

8.4.16 Get_translated_string_values_of_tuple function

The **get_translated_string_values_of_tuple** computes the set **of translated_string_value**s that appear in the **content** list of columns for the tuple located at the **index** position.

EXPRESS specification:

```
*)
FUNCTION get_translated_string_values_of_tuple(
     content: LIST[1:?] OF column; index: INTEGER):
     SET OF translated_string_value;
LOCAL
     translated_values: SET OF translated_string_value := [];
END_LOCAL;

IF (index > SIZEOF(content[1].values))  -- abnormal case
THEN
     RETURN([]);
```

```
    END_IF;

    REPEAT i := 1 TO SIZEOF(content);
         IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.TRANSLATED_STRING_VALUE'
              IN TYPEOF(content[i].values[index]))
         THEN
              translated_values := translated_values +
                   content[i].values[index];
         END_IF;
    END_REPEAT;

    RETURN(translated_values);

    END_FUNCTION; -- get_translated_string_values_of_tuple
    (*

    *)
    END_SCHEMA; --ISO13584_table_resource_schema
    (*
```

9 ISO13584_variable_semantics_schema

This clause defines the requirements for the **ISO13584_variable_semantics_schema**. The following EXPRESS declaration introduces the **ISO13584_variable_semantics_schema** block and identifies the necessary external references.

EXPRESS specification:

```
    *)
    SCHEMA ISO13584_variable_semantics_schema;

    REFERENCE FROM ISO13584_IEC61360_dictionary_schema
         (property_BSU);

    REFERENCE FROM ISO13584_generic_expressions_schema
         (variable_semantics);

    REFERENCE FROM ISO13584_library_expressions_schema
         (compatible_type_and_library_expression,
         syntax_of);

    REFERENCE FROM ISO13584_extended_dictionary_schema
         (applicable_properties,
         data_type_typeof,
         data_type_class_of);

    (*
```

NOTE The schemas referenced above can be found in the following documents:
ISO13584_IEC61360_dictionary_schema IEC 61360-2
(which is duplicated for convenience in informative annex D of ISO 13584-42),
ISO13584_generic_expressions_schema ISO 13584-20,

ISO13584_library_expressions_schema	This part of ISO 13584,
ISO13584_extended_dictionary_schema	This part of ISO 13584.

9.1 Introduction to the ISO13584_variable_semantics_schema

The role of the **ISO13584_variable_semantics_schema** is to provide the resources for referencing the current values of the various items that characterise an instance of a class modelled according to **ISO13584_IEC61360_dictionary_schema**, or its extensions defined in the **ISO13584_extended_dictionary_schema**.

The **ISO13584_variable_semantics_schema** models:

— the **variable_semantics** that provides access to instance characteristics: class names, codes and versions, property names, code, version and value of property types that may be defined using the common **ISO13584_IEC61360_dictionary_schema**;

— the mechanism that provides access from an operation to the properties of an instance the operation applies to;

— the mechanism that provides access from a method to the view created by the method;

— the fact that the type of the variables that substitutes for the values defined by these mechanisms are compatible with the data type of these values.

The **ISO13584_variable_semantics_schema** does not model:

— the use of these **variable_semantics** to model the class instance operations.

9.2 Fundamental concepts and assumptions for the ISO13584_variable_semantics_schema

9.2.1 Instance related operation

In the object oriented framework, instances are modelled through properties and operations.

EXAMPLE 1 In the EXPRESS language, derivation functions are examples of operations that apply to instances.

Operations always apply to one instance, usually called the SELF instance: they access the properties of the SELF instance.

Specific mechanisms are required to represent, within the context of one operation, the properties of the SELF instance.

EXAMPLE 2 In the EXPRESS language, access to the SELF instance is denoted by the "SELF" notation.

9.2.2 Instance structure

An instance may consist of other instances, referred to by means of properties of this instance.

NOTE Such a relationship is called a composition relationship.

It shall be possible, from an operation, to access to the properties of the instances constituting the SELF instance.

EXAMPLE In programming languages, access to the attribute of some attribute is denoted by the "dot" notation.

9.2.3 Context of a method

In ISO 13584, views are created by the models' methods.

Specific mechanisms are required to represent, within the context of a method, the properties of the view created by this method.

9.3 ISO13584_variable_semantics_schema type definition

This clause defines the type resources in the **ISO13584_variable_semantics_schema**.

9.3.1 Property_semantics_or_path

A **property_semantics_or_path** is either a **property_semantics**, or a **sub_property_path**.

EXPRESS specification:

```
*)
TYPE property_semantics_or_path = SELECT(
     property_semantics,
     sub_property_path);
END_TYPE; -- property_semantics_or_path
(*
```

9.4 ISO13584_variable_semantics_schema entity definitions

This clause defines the entity data types in the **ISO13584_variable_semantics_schema**.

9.5 Property_semantics

A **property_semantics** is a **variable_semantics** that allows the reference to a feature of a property of a class instance, or when the value of a property is itself a class instance, the reference to a feature of a property of the value of this property, and so on. The property features that may be referenced are either their value, or any of the pieces of information used to identify the property. When associated with a **generic_variable**, such a **property_semantics** stands for the interpretation function that associates the relevant feature of the property with the **generic_variable**.

EXAMPLE Let x be a particular screw modelled as an instance of a **component_class** that models a family of screws. The threaded diameter of x may be modelled by a **property_semantics** that represents the feature of this property that is its current value. When a **numeric_variable**, associated with this **property_semantics** by an **environment**, is involved as an operand of an **expression**, this **numeric_variable** stands for this current value, is to be used as the value of the operand when the **expression** is evaluated.

NOTE 1 The feature of the threaded diameter represented by the **property_semantics** is characterised by subtyping of the **property_semantics**. The data type of the value of this feature is characterised by subtyping the associated **generic_variable**. Such a data type could be a STRING for a name, or a REAL for a value.

NOTE 2 **expression**, **variable_semantics**, **numeric_variable**, **generic_variable** and **environment** are defined in ISO 13584-20: 1998.

EXPRESS specification:

```
*)
ENTITY property_semantics
ABSTRACT SUPERTYPE OF(ONEOF(
     self_property_semantics,
```



```
      open_view_property_semantics))
SUBTYPE OF(variable_semantics);
      the_property: property_BSU;
      its_own_property: OPTIONAL sub_property_path;
WHERE
      WR1: check_property_semantics(SELF);
END_ENTITY; -- property_semantics
(*
```

Attribute definitions:

the_property: the **property_BSU** that defines the property of the instance that corresponds to the **property_semantics**.

its_own_property: when **the_property** is itself a class instance, the referred property of this instance.

Formal propositions:

WR1: either **its_own_property** does not exist, or **the_property** is a class instance and **its_own_property** is an **applicable_properties** of this class instance.

Informal propositions:

IP1: **the_property** shall be an applicable property for the class to which the instance belongs.

9.6 Sub_property_path

The **sub_property_path** entity provides for referencing recursively a property of a property that is a class instance.

EXAMPLE A bolt + nut assembly may have one property that points to the nut. The **sub_property_path** entity allows to identify any property of this nut, for instance its threaded diameter.

NOTE 1 The role of a **sub_property_path** entity is to give precision to the final instance property referred to by a **property_semantics** entity. Only one of these latter entities may be used within an operation definition. The **sub_property_path** entities enable specification of the "path" that defines the property finally addressed.

NOTE 2 The **sub_property_path** entity enables representing the dot notation used in data specification and in programming languages.

EXPRESS specification:

```
*)
ENTITY sub_property_path;
      the_property: property_BSU;
      its_own_property: OPTIONAL sub_property_path;
WHERE
      WR1: (NOT(EXISTS(SELF.its_own_property)))
           OR (data_type_typeof(SELF.the_property) = [])
           OR
           (('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_INSTANCE_TYPE'
           IN data_type_typeof(SELF.the_property))
           AND applicable_properties(data_type_class_of(
           SELF.the_property)[1],[SELF.its_own_property.the_property]));
```



```
END_ENTITY; -- sub_property_path
(*
```

Attribute definitions:

the_property: the **property_BSU** that defines the property of the class instance that corresponds to the **sub_property_path**.

its_own_property: when **the_property** is itself a class instance, the referred property of this instance.

Formal propositions:

WR1: either **its_own_property** does not exist, or **the_property** is a class instance and **its_own_property** is an **applicable_properties** of this class instance.

9.7 Variable_semantics referring to the SELF entity

The following entities provides for referencing the different features of a class instance from an operation on this instance.

NOTE These **variable_semantic**s correspond to the SELF notation of data specification or programming languages.

9.7.1 Self_variable_semantics

A **self_variable_semantics** is any **variable_semantics** that refers to the current value of the SELF instance that supports an operation. It refers either to a property of the SELF instance, or to the class to which the SELF instance belongs.

EXPRESS specification:

```
*)
ENTITY self_variable_semantics
ABSTRACT SUPERTYPE OF(ONEOF(
     self_property_semantics,
     self_class_variable_semantics))
SUBTYPE OF(variable_semantics);
END_ENTITY; -- self_variable_semantics
(*
```

9.7.2 Self_property_semantics

A **self_property_semantics** is a **variable_semantics** that allows the reference to any feature of a property of the instance that supports an operation, or, when the value of this property is itself a class instance, the reference to any feature of a property of this class instance, and so on. The feature referred to is defined by subtyping.

EXPRESS specification:

```
*)
ENTITY self_property_semantics
ABSTRACT SUPERTYPE OF(ONEOF(
     self_property_value_semantics,
```



```
      self_property_name_semantics))
SUBTYPE OF(self_variable_semantics, property_semantics);
END_ENTITY; -- self_property_semantics
(*
```

9.7.3 Self_property_value_semantics

The **self_property_value_semantics** entity allows the reference to the value of a property of the instance that supports an operation, or when the value of this property is itself a class instance, the reference to the value of a property of this class instance, and so on. When a **generic_variable**, or any of its subtypes, is associated with a **self_property_value_semantics**, the type of this **generic_variable** shall be compatible with the data type defined for the corresponding property. In the operation definition, such a **generic_variable** stands for the current value of the referenced property.

EXPRESS specification:

```
*)
ENTITY self_property_value_semantics
SUBTYPE OF(self_property_semantics);
WHERE
      WR1: QUERY(v <* syntax_of(SELF)
           | NOT compatible_type_and_library_expression(
           BSU_of_property_semantics(SELF), v)) = [];
END_ENTITY; -- self_property_value_semantics
(*
```

Formal propositions:

WR1: all the **generic_variable**s that may be associated with a **self_property_value_semantics** shall be type-compatible with the final property referenced by this **property_semantics**.

9.7.4 Self_property_name_semantics

The **self_property_name_semantics** entity allows reference to the different names of a property of the instance that supports an operation, or when the value of this property is itself a class instance, the reference either to the different names that characterize the class to which the current value of the property belongs or to the different names of a property of this class instance, and so on. The **generic_variable**s that may be associated with such a **variable_semantics** shall be **string_variable**s. In the operation definition, such a **string_variable** stands for the current value of the referenced name.

EXPRESS specification:

```
*)
ENTITY self_property_name_semantics
ABSTRACT SUPERTYPE OF(ONEOF(
      self_property_preferred_name_semantics,
      self_property_short_name_semantics,
      self_property_code_semantics,
      self_property_version_semantics,
      self_property_class_code_semantics,
      self_property_class_supplier_code_semantics,
      self_property_class_version_semantics))
```



```
SUBTYPE OF(self_property_semantics);
WHERE
     WR1: QUERY(v <* syntax_of(SELF)
          | NOT('ISO13584_EXPRESSIONS_SCHEMA.STRING_VARIABLE'
          IN TYPEOF(v))) = [];
END_ENTITY; -- self_property_name_semantics
(*
```

Formal propositions:

WR1: only **string_variable**s may be associated to a **self_property_name_semantics**.

NOTE Following ISO 13584-20, a **variable** is associated to a **variable_semantics** by means of an **environment** entity.

9.7.4.1 Self_property_preferred_name_semantics

The **self_property_preferred_name_semantics** entity allows reference to the possibly translated preferred name of a property of the instance that supports an operation, or when the value of this property is itself a class instance, the reference to the possibly translated preferred name of a property of this class instance, and so on. The final value associated to such a **variable_semantics** is a string that contains the possibly translated preferred name.

NOTE 1 The interpretation function that selects the language in which the value of such a variable is expressed is implementation dependent.

EXPRESS specification:

```
*)
ENTITY self_property_preferred_name_semantics
SUBTYPE OF(self_property_name_semantics);
END_ENTITY; -- self_property_preferred_name_semantics
(*
```

Informal propositions:

IP1: the value associated with a **self_property_preferred_name_semantics** shall be a string, the length of which shall be less than or equal to **preferred_name_len** specified in ISO 13584-42, clause D.3.2.

NOTE 2 The value of **preferred_name_len** is 70 characters.

9.7.4.2 Self_property_short_name_semantics

The **self_property_short_name_semantics** entity allows reference to the possibly translated short name of a property of the instance that supports an operation, or when the value of this property is itself a class instance, the reference to the possibly translated short names of a property of this class instance, and so on. The final value associated to such a **variable_semantics** is a string that contains the possibly translated short name.

NOTE 1 The interpretation function that selects the language in which the value of such a variable is expressed is implementation dependent.

EXPRESS specification:

```
*)
ENTITY self_property_short_name_semantics
SUBTYPE OF(self_property_name_semantics);
END_ENTITY; -- self_property_short_name_semantics
(*
```

Informal propositions:

IP1: the value associated with a **self_property_short_name_semantics** shall be a string the length of which shall be less than or equal to **short_name_len** specified in ISO 13584-42, clause D.3.2.

NOTE 2 The value of **short_name_len** is 15 characters.

9.7.4.3 Self_property_code_semantics

The **self_property_code_semantics** entity allows reference to be made to the code of a property of the instance that supports an operation, or when the value of this property is itself a class instance, the reference to the code of a property of this class instance, and so on.. The final value associated to such a **variable_semantics** is a string that contains the property code.

EXPRESS specification:

```
*)
ENTITY self_property_code_semantics
SUBTYPE OF(self_property_name_semantics);
END_ENTITY; -- self_property_code_semantics
(*
```

Informal propositions:

IP1: the value associated with a **self_property_code_semantics** shall conform to the **property_code_type** specified in ISO 13584-42, clause D.3.8.1.10.

NOTE A **property_code_type** string value is no more than 14 characters and does not include spaces, hyphens or dots.

9.7.4.4 Self_property_version_semantics

The **self_property_version_semantics** entity allows reference to be made to the version of a property of the instance that supports an operation, or when the value of this property is itself a class instance, the reference to the version of a property of this class instance, and so on. The final value associated to such a **variable_semantics** is a string that contains the property version.

EXPRESS specification:

```
*)
ENTITY self_property_version_semantics
SUBTYPE OF(self_property_name_semantics);
END_ENTITY; -- self_property_version_semantics
(*
```


Informal propositions:

IP1: the value associated with a **self_property_version_semantics** shall conform to the **version_type** specified in ISO 13584-42, clause D.3.8.1.18.

NOTE A **version_type** string value is no more than 9 digits.

9.7.4.5 Self_property_class_code_semantics

The **self_property_class_code_semantics** entity shall be used only when the value of a property of the instance that supports an operation is itself a class instance, or when the value of one property of this class instance is also a class instance, and so on. It allows to refer to the code of the class to which the current value of the final property referenced by the **self_property_variable_semantics** belongs. The final value associated to such a **variable_semantics** is a string that contains the class code.

EXPRESS specification:

```
*)
ENTITY self_property_class_code_semantics
SUBTYPE OF(self_property_name_semantics);
END_ENTITY; -- self_property_class_code_semantics
(*
```

Informal propositions:

IP1: the value associated with a **self_property_class_code_semantics** shall conform to the **class_code_type** specified in ISO 13584-42, clause D.3.8.1.1.

NOTE A **class_code_type** string value is no more than 14 characters and does not include spaces, hyphens or dots.

9.7.4.6 Self_property_class_supplier_code_semantics

The **self_property_class_supplier_code_semantics** entity shall be used only when the value of a property of the instance that supports an operation is itself a class instance, or when the value of one property of this class instance is also a class instance, and so on. It allows to refer to the code of the supplier of the class to which the current value of the final property referenced by the **self_property_variable_semantics** belongs. The final value associated to such a **variable_semantics** is a string that contains the supplier code.

EXPRESS specification:

```
*)
ENTITY self_property_class_supplier_code_semantics
SUBTYPE OF(self_property_name_semantics);
END_ENTITY; -- self_property_class_supplier_code_semantics
(*
```

Informal propositions:

IP1: the value associated with a **self_property_class_supplier_code_semantics** shall conform to the **supplier_code_type** specified in ISO 13584-42, clause D.3.8.1.14.

NOTE A **supplier_code_type** string value is no more than 14 characters and does not include spaces, hyphens or dots. Its content is defined by ISO 13584-26.

9.7.4.7 Self_property_class_version_semantics

The **self_property_class_version_semantics** entity shall be used only when the value of a property of the instance that supports an operation is itself a class instance, or when the value of one property of this class instance is also a class instance, and so on. It allows to refer to the version of the class to which the current value of the final property referenced by the **self_property_variable_semantics** belongs. The final value associated to such a **variable_semantics** is a string that contains the class version.

EXPRESS specification:

```
*)
ENTITY self_property_class_version_semantics
SUBTYPE OF(self_property_name_semantics);
END_ENTITY; -- self_property_class_version_semantics
(*
```

Informal propositions:

IP1: the value associated with a **self_property_class_version_semantics** shall conform to the **version_type** specified in ISO 13584-42, clause D.3.8.1.18.

NOTE A **version_type** string value is no more than 9 digits.

9.7.5 Self_class_variable_semantics

A **self_class_variable_semantics** is any **variable_semantics** that refers to a characteristic of the class to which the SELF instance belongs.

EXPRESS specification:

```
*)
ENTITY self_class_variable_semantics
ABSTRACT SUPERTYPE OF(self_class_name_semantics)
SUBTYPE OF(self_variable_semantics);
END_ENTITY; -- self_class_variable_semantics
(*
```

9.7.6 Self_class_name_semantics

A **self_class_name_semantics** is any **variable_semantics** that refer to a name of the class to which the SELF instance belongs

EXPRESS specification:

```
*)
ENTITY self_class_name_semantics
ABSTRACT SUPERTYPE OF(ONEOF(
      self_class_preferred_name_semantics,
      self_class_short_name_semantics,
      self_class_code_semantics,
```



```
      self_class_supplier_code_semantics,
      self_class_version_semantics))
SUBTYPE OF(self_class_variable_semantics);
WHERE
      WR1: SIZEOF(QUERY(v <* syntax_of(SELF) | NOT
           ('ISO13584_EXPRESSIONS_SCHEMA.STRING_VARIABLE'
           IN TYPEOF(v)))) = 0;
END_ENTITY; -- self_class_name_semantics
(*
```

Formal propositions:

WR1: only **string_variable**s may be associated to a **self_class_name_semantics**.

NOTE Following ISO 13584-20, a **variable** is associated to a **variable_semantics** by means of an **environment** entity.

9.7.6.1 Self_class_preferred_name_semantics

The **self_class_preferred_name_semantics** entity allows reference to the possibly translated preferred name of the class of the instance that supports an operation. The final value associated to such a **variable_semantics** is a string that contains the possibly translated preferred name.

NOTE 1 The interpretation function that selects the language in which the value of such a variable is expressed is implementation dependent.

EXPRESS specification:

```
*)
ENTITY self_class_preferred_name_semantics
SUBTYPE OF(self_class_name_semantics);
END_ENTITY; -- self_class_preferred_name_semantics
(*
```

Informal propositions:

IP1: the value associated with a **self_class_preferred_name_semantics** shall be a string the length of which shall be less than or equal to **preferred_name_len** specified in ISO 13584-42, clause D.3.2.

NOTE 2 The value of **preferred_name_len** is 70 characters.

9.7.6.2 Self_class_short_name_semantics

The **self_class_short_name_semantics** entity allows reference to the possibly translated short name of the class of the instance that supports an operation. The final value associated to such a **variable_semantics** is a string that contains the possibly translated short name.

NOTE 1 The interpretation function that selects the language in which the value of such a variable is expressed is implementation dependent.

EXPRESS specification:

```
*)
ENTITY self_class_short_name_semantics
```



```
SUBTYPE OF(self_class_name_semantics);
END_ENTITY; -- self_class_short_name_semantics
(*
```

Informal propositions:

IP1: the value associated with a **self_class_short_name_semantics** shall be a string the length of which shall be less than or equal to **short_name_len** specified in ISO 13584-42, clause D.3.2.

NOTE 2 The value of **short_name_len** is 15 characters.

9.7.6.3 Self_class_code_semantics

The **self_class_code_semantics** entity allows reference to the code of the class of the instance that supports an operation. The final value associated to such a **variable_semantics** is a string that contains the class code.

EXPRESS specification:

```
*)
ENTITY self_class_code_semantics
SUBTYPE OF(self_class_name_semantics);
END_ENTITY; -- self_class_code_semantics
(*
```

Informal propositions:

IP1: the value associated with a **self_class_code_semantics** shall conform to the **class_code_type** specified in ISO 13584-42, clause D.3.8.1.1.

NOTE A **class_code_type** string value is no more than 14 characters and does not include spaces, hyphens or dots.

9.7.6.4 Self_class_supplier_code_semantics

The **self_class_supplier_code_semantics** entity allows the reference to the code of the supplier of the class of the instance that supports an operation.

NOTE 1 Such an operation may be a function, a constraint or a method.

The final value associated to such a **variable_semantics** is a string that contains the supplier code.

EXPRESS specification:

```
*)
ENTITY self_class_supplier_code_semantics
SUBTYPE OF(self_class_name_semantics);
END_ENTITY; -- self_class_supplier_code_semantics
(*
```

Informal propositions:

IP1: the value associated with a **self_class_supplier_code_semantics** shall conform to the **supplier_code_type** specified in ISO 13584-42, clause D.3.8.1.14.

NOTE 2 A **supplier_code_type** string value is no more than 14 characters and does not include spaces, hyphens or dots. Its content is defined by ISO 13584-26.

9.7.6.5 Self_class_version_semantics

The **self_class_version_semantics** entity allows reference to the version of the class of the instance that supports an operation. The final value associated to such a **variable_semantics** is a string that contains the class version.

EXPRESS specification:

```
*)
ENTITY self_class_version_semantics
SUBTYPE OF(self_class_name_semantics);
END_ENTITY; -- self_class_version_semantics
(*
```

Informal propositions:

IP1: the value associated with a **self_class_version variable_semantics** shall conform to the **version_type** specified in ISO 13584-42, clause D.3.8.1.18.

NOTE A **version_type** string value is no more than 9 digits.

9.8 Variables referring to the open view characteristics

The following entities may be used within the context of a method to provide access to a view that is created by the method.

9.8.1 Open_view_variable_semantics

The **open_view_variable_semantics** is a **variable_semantics** that enables access to the different features of a view created by a method.

NOTE Only one subtype of **open_view_variable_semantics** is defined in the **ISO13584_variable_semantics_schema**.

EXPRESS specification:

```
*)
ENTITY open_view_variable_semantics
ABSTRACT SUPERTYPE OF(open_view_property_semantics)
SUBTYPE OF(variable_semantics);
END_ENTITY; -- open_view_variable_semantics
(*
```

9.8.2 Open_view_property_semantics

The **open_view_property_semantics** is a **variable_semantics** that allows the reference to any feature of a property of the view that is created by a method, or, when the value of this property is itself a class instance, the reference to any feature of a property of this class instance, and so on. The feature referred to is defined by subtyping.

EXPRESS specification:

```
*)
ENTITY open_view_property_semantics
ABSTRACT SUPERTYPE OF(open_view_property_value_semantics)
SUBTYPE OF(open_view_variable_semantics, property_semantics);
END_ENTITY; -- open_view_property_semantics
(*
```

9.8.3 Open_view_property_value_semantics

The **open_view_property_value_semantics** entity allows the reference he value of a property of the view that is created by a method, or, when the value of this property is itself a class instance, the reference he value of a property of this class instance, and so on.

EXPRESS specification:

```
*)
ENTITY open_view_property_value_semantics
SUBTYPE OF(open_view_property_semantics);
WHERE
     WR1: SIZEOF(QUERY(v <* syntax_of(SELF)
          | NOT compatible_type_and_library_expression(
          BSU_of_property_semantics(SELF), v))) = 0;
END_ENTITY; -- open_view_property_value_semantics
(*
```

Formal propositions:

WR1: all the **generic_variable**s that may be associated with an **open_view_property_value_semantics** shall be type-compatible with the final property referenced by this **property_semantics**.

9.9 ISO13584_variable_semantics_schema function definitions

9.9.1 BSU_of_property_semantics function

The **BSU_of_property_semantics** function computes the **property_BSU** that defines the final property of a **property_semantics**.

EXPRESS specification:

```
*)
FUNCTION BSU_of_property_semantics(v: property_semantics_or_path):
     property_BSU;

LOCAL
     prop: property_BSU;
END_LOCAL;

prop := v.the_property;

IF EXISTS(v.its_own_property)
```



```
THEN
      RETURN(BSU_of_property_semantics(v.its_own_property));
ELSE
      RETURN(prop);
END_IF;

END_FUNCTION; -- BSU_of_property_semantics
(*
```

9.9.2 Check_property_semantics function

The **check_property_semantics** function checks if either the **property_semantics** (**sem**) **its_own_property** attribute does not exist, or if **the_property** is a class instance, and **its_own_property** is an **applicable_properties** of this class instance.

EXPRESS specification:

```
*)
FUNCTION check_property_semantics(sem: property_semantics): LOGICAL;

LOCAL
      res: LOGICAL;
END_LOCAL;

IF (EXISTS(sem.its_own_property)) AND
      NOT(data_type_typeof(sem.the_property) = [])
THEN
      res := ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_INSTANCE_TYPE'
           IN data_type_typeof(sem.the_property))
           AND applicable_properties(data_type_class_of(
           sem.the_property)[1], [sem.its_own_property.the_property]);

      RETURN(res);
ELSE
      RETURN(UNKNOWN);
END_IF;

END_FUNCTION; -- check_property_semantics
(*

*)
END_SCHEMA; -- ISO13584_variable_semantics_schema
(*
```

10 ISO13584_domain_resource_schema

This clause defines the requirements for the **ISO13584_domain_resource_schema**. The following EXPRESS declaration introduces the **ISO13584_domain_resource_schema** block and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA ISO13584_domain_resource_schema;

REFERENCE FROM ISO13584_IEC61360_dictionary_schema
      (class_BSU,
      is_subclass,
      list_to_set);

REFERENCE FROM ISO13584_generic_expressions_schema
      (generic_variable,
      variable_semantics,
      used_variables);

REFERENCE FROM ISO13584_expressions_schema
      (boolean_expression,
      literal_number,
      numeric_expression);

REFERENCE FROM ISO13584_library_expressions_schema
      (class_instance_expression,
      compatible_variable_and_library_expression,
      library_expression,
      semantics_of,
      syntax_of);

REFERENCE FROM ISO13584_table_resource_schema
      (collects_columns,
      column_traversal_variable_semantics,
      table_expression,
      table_identification,
      used_table_literals);

REFERENCE FROM ISO13584_external_file_schema
      (message);
(*
```

NOTE The schemas referenced above can be found in the following documents:
ISO13584_IEC61360_dictionary_schema IEC 61360-2
(which is duplicated for convenience in informative annex D of ISO 13584-42),
ISO13584_generic_expressions_schema ISO 13584-20,
ISO13584_expressions_schema ISO 13584-20,
ISO13584_instance_resource_schema This part of ISO 13584,
ISO13584_library_expressions_schema This part of ISO 13584,
ISO13584_table_resource_schema This part of ISO 13584,
ISO13584_external_file_schema This part of ISO 13584.

10.1 Introduction to the ISO13584_domain_resource_schema

The role of the **ISO13584_domain_resource_schema** is to provide the resources for modelling the set of allowed values that constitutes the domain of a variable in some context. The variables considered in the **ISO13584_domain_resource_schema** are those that are specified according to the **ISO13584_library_expressions_schema**. They are associated with a **variable_semantics** that defines their roles and meanings.

The domain of such a variable may be independent of any other variables, or it may depend on the values of some other variables. The resources introduced in the **ISO13584_domain_resource_schema** enable the characterisation of both kinds of domains. These resources are generic in nature and can be used for various purposes and in different application contexts. In this part of ISO 13584, these resources are used in the **ISO13584_library_content_schema** to define the extension of a class.

The **ISO13584_domain_resource_schema** models:

— the mechanisms for modelling the domains that constitute the allowed set of values for one, or several variables;

— the mechanisms for modelling how the domain of some variable is dependent on the value of some other variables,

EXAMPLE 1 The resources defined in the **ISO13584_domain_resource_schema** may be used within a screw family to model what diameters are allowed for a screw that has the length defined.

NOTE 1 In the EXPRESS language, such a relationship would be modelled by a constraint.

— the mechanisms to model how the values of one set of variables may be derived from the values of another set of variables, when there exist some functional dependency from the latter to the former.

EXAMPLE 2 The resources defined in the **ISO13584_domain_resource_schema** may be used to model the derivation function that enables the system to compute the value of the area of a rectangle from its length and its width.

NOTE 2 In the EXPRESS language, such a relationship would be modelled by a derivation function.

The **ISO13584_domain_resource_schema** does not model:

— the use of the resources defined in this schema to express relationships between different properties referring to the same product or part.

NOTE 3 The resource constructs defined in the **ISO13584_domain_resource_schema** are used in the **ISO13584_library_content_schema** to model both the allowed sets of values of the properties of a family of parts and the derivation functions that enable the derivation of the values of some properties from the values of the identification characteristics of such a family.

10.2 Fundamental concepts and assumption for the ISO13584_domain_resource_schema

Two different mathematical concepts provide for expressing relationships between the values of two different sets of variables. The more general concept is the one of mathematical relation: the tuple formed by the ordered list of values of the different variables shall belong to a set of tuples that may depend on the values of the other variables. Such a relationships is called a *relational dependency* between both sets of variables.

An important special case of relation corresponds with the concept of function: when the values of the variables belonging to the first set are defined, the values of the variables belonging to the second set are fixed. They may be computed by a system if the specification of the function is available. Such a relationship is usually called a *functional dependency*. A functional dependency is a special case of relational dependency where the allowed set of values degenerates to a singleton and where the relation that shall hold between two sets of variables degenerates to a function.

When the domain of some variable depends on the values of some other variables, both sets of variables need to be characterised differently to capture the role they play in the relation. In particular, when the relation is a function, the variables that define the domain of the function play a different role from the ones that define its range. In the **ISO13584_domain_resource_schema**, the variables that

are supposed to be assigned a value independently of the relation are referenced by the **assumes** attribute. The variables of which the values are intended to be restricted by the relation, or computed by the function, are defined by the **defines** attribute. When all the variables play the same role with respect to the relation, they shall all be referenced by the **defines** attribute, and the **assumes** attribute shall be an empty set.

Various means may be used to express a relation, or a function. In the **ISO13584_domain_resource_schema**, the same mathematical relation may be associated with different information models, each one defining a part of the complete relation. Each of these information models is associated with a predicate, called its **guard** that is either a **boolean_expression** or a specific **others** entity. When a **guard** evaluates to TRUE, the specific information model associated with this **guard** defines the relation. When several **guard**s evaluate to TRUE, any one of the information models associated with one of these **guard**s may be considered as representing the relation, or the function. When no **guard** evaluates to TRUE, the information model associated with **others** defines the relation. The **other** guard shall always exist. This ensures that the relation, or the function, is always defined, as a total function.

In the **ISO13584_domain_resource_schema** the information models used to express a function are the following:

— a table, of which the **assumes** attributes includes the table key,

— an expression, and

— a function that specifies that a variable has no value.

The latter information model may be used to specify that a property, defined as optional in some application context, does not exist. All these specifications are constructive, i.e., from the specification it is possible to compute the result of the function.

When several properties belong to the **defines** set, only tables may be used to define the function: each variable semantics in the **defines** set shall correspond to one column of the corresponding tables and all the properties belonging to the **defines** set shall take a value that corresponds to the same line of the table.

The information models used to express a relation are:

— a table,

— a range,

— a type, in a subclass/superclass network, and

— a predicate defined relation.

The three first specifications are constructive, i.e., from the specification it is possible to compute the domain that results from the relation. The last specification is not constructive. It may be used, to filter some other domain, or to assert that some constraint shall hold.

Only tables and predicate defined relation shall be used when the **defines** set contains several **variable_semantics**.

10.3 ISO13584_domain_resource_schema type definition

This subclause contains the EXPRESS type definition in the **ISO13584_domain_resource_schema**.

10.3.1 Boolean_expression_or_others

A **boolean_expression_or_others** is either a **boolean_expression** or an **others** entity.

EXPRESS specification:

```
*)
TYPE boolean_expression_or_others = SELECT(
     boolean_expression,
     others);
END_TYPE; -- boolean_expression_or_others
(*
```

10.4 ISO13584_domain_resource_schema entity definitions

This subclause contains the EXPRESS entity definition in the **ISO13584_domain_resource_schema**.

10.4.1 Others

An **others** entity specifies the domain that shall be used when no **guard** evaluates to TRUE in a **domain_restriction**.

EXPRESS specification:

```
*)
ENTITY others;
END_ENTITY; -- others
(*
```

10.4.2 Domain_restriction

A **domain_restriction** defines the domain of the variables that may be associated with the **variable_semantics** belonging to the **defines** set, according to the values that are associated with each **variable_semantics** of the **assumes** set. A **domain_restriction** models a relational dependency between two sets of variables. This relational dependency may be a functional dependency modelled as a **functional_domain_restriction** subtype.

A **domain_restriction** may be defined by means of tables. The **table_identifier**s of the tables used in the specification of the domain are collected in the **base_tables** derived attribute.

EXPRESS specification:

```
*)
ENTITY domain_restriction
SUPERTYPE OF(functional_domain_restriction);
     defines: SET[1:?] OF variable_semantics;
     assumes: SET[0:?] OF variable_semantics;
     domains: SET[1:?] OF guarded_simple_domain;
     constraint_description: OPTIONAL message;
DERIVE
     base_tables: SET [0:?] OF table_identification
          := used_tables_in_domain(SELF);
WHERE
     WR1: SIZEOF(QUERY(g <* SELF.domains |
          'ISO13584_DOMAIN_RESOURCE_SCHEMA.OTHERS'
          IN TYPEOF(g.guard))) = 1;
```



```
END_ENTITY; -- domain_restriction
(*
```

Attribute definitions:

defines: the set of **variable_semantics** for which the domain is defined.

assumes: the set of **variable_semantics** whose associated values are required to evaluate the domain.

domains: the set of **guarded_simple_domain**s corresponding to the different parts of the domain according to the values associated with the **assumes variable_semantics**.

constraint_description: a possibly translated message that defines textually the constraint defined by the **domain_restriction**.

NOTE This message is intended to be displayed in some application context when the constraint is only checked by the system after the user selection (see clause 12.6.9 **filters** in **implicit_model_class_extension**).

base_tables: the **table_identifier**s of the tables used in the specification of the domain.

Formal propositions:

WR1: there shall always be one guard defined as **others**.

10.4.3 Guarded_simple_domain

A **guarded_simple_domain** is a domain that is associated with a **guard** that specifies whether this domain is allowed for use.

EXPRESS specification:

```
*)
ENTITY guarded_simple_domain;
      guard: boolean_expression_or_others;
      domain: simple_domain;
INVERSE
      item_of: domain_restriction FOR domains;
WHERE
      WR1: variables_belong_to_assumes(SELF);
END_ENTITY; -- guarded_simple_domain
(*
```

Attribute definitions:

guard: the **boolean_expression** that specifies whether the **domain** applies.

domain: the domain that may be used if the **guard** evaluates to TRUE. In case of several **guard**s evaluating to TRUE, any **simple_domain** of which the **guard** evaluates to TRUE shall define the right domain.

item_of: the **domain_restriction** that contains the **guarded_simple_domain**.

Formal propositions:

WR1: only variables that are associated with the **variable_semantics** of the **assumes** set of a **domain_restriction** may be used to define the **guard**s of its **guarded_simple_domain**s.

10.4.4 Simple_domain

A **simple_domain** entity specifies the domain of values for the set of variables associated with a set of variable semantics.

EXPRESS specification:

```
*)
ENTITY simple_domain
ABSTRACT SUPERTYPE OF(ONEOF(table_defined_domain,
     type_defined_domain,
     subclass_defined_domain,
     constant_range_defined_domain,
     variable_range_defined_domain,
     predicate_defined_domain,
     simple_functional_domain));
INVERSE
     referenced_by: guarded_simple_domain FOR domain;
END_ENTITY; -- simple_domain
(*
```

Attribute definitions:

referenced_by: the **guarded_simple_domain** that references the **simple_domain**.

10.4.5 Table_defined_domain

A **table_defined_domain** entity is a domain specified by means of a **table_expression**. All the **variable_semantics** of the **defines** set shall correspond to one column in the table. This column defines the domain of the corresponding variable, and all the variables shall take their values in the same row of the table. If some **variable_semantics** of the **assumes** set correspond to an entry in the appropriate table column, the rest of this row in the table shall equal the current values of the variables corresponding to these **variable_semantics**.

NOTE The fact that all the **variable_semantics** in the **defines** attribute shall take their values in the same row of the table define a relation between these **variable_semantics**.

EXPRESS specification:

```
*)
ENTITY table_defined_domain
SUBTYPE OF(simple_domain);
     from_table: table_expression;
WHERE
     WR1: SELF\simple_domain.referenced_by.item_of.defines
          <= list_to_set(collects_columns(SELF.from_table));
     WR2: QUERY(sem <* collects_var_sem(
          used_variables(SELF.from_table))
```



```
            | NOT((sem IN
            (SELF\simple_domain.referenced_by.item_of.assumes
            + SELF\simple_domain.referenced_by.item_of.defines))
            OR ((('ISO13584_TABLE_RESOURCE_SCHEMA'
            + '.COLUMN_TRAVERSAL_VARIABLE_SEMANTICS')
            IN TYPEOF(sem))) AND
            (sem\column_traversal_variable_semantics.domain
            IN collects_columns(SELF.from_table)))) = [];
END_ENTITY; -- table_defined_domain
(*
```

Attribute definitions:

from_table: the **table_expression** of which rows define the domain of the **SELF\simple_domain.referenced_by.item_of.defines** variables.

Formal propositions:

WR1: each **SELF\simple_domain.referenced_by.item_of.defines variable_semantics** shall belong to the set of **variable_semantics** that corresponds to the columns of the **table_expression**.

WR2: the **used_variables** in the **table_expression** shall belong to the set of variables associated with the **SELF\simple_domain.referenced_by.item_of.defines variable_semantics** or to the **SELF\simple_domain.referenced_by.item_of.assumes variable_semantics**, or they shall be column iterators associated with **column_traversal_variable_semantics** that reference one of the columns of the tables involved in the **from_table table_expression**.

10.4.6 Type_defined_domain

A **type_defined_domain** entity is a domain that is the complete set of values associated with the data type of the **defines variable_semantics**.

EXPRESS specification:

```
*)
ENTITY type_defined_domain
SUBTYPE OF(simple_domain);
WHERE
     WR1: SIZEOF(SELF\simple_domain.referenced_by
          .item_of.defines) = 1;
END_ENTITY; -- type_defined_domain
(*
```

Formal propositions:

WR1: this entity shall define the domain of only one **variable_semantics**.

10.4.7 Subclass_defined_domain

A **subclass_defined_domain** entity defines the domain of one or several variables, whose data types are **class_instance_type**s as a **class**. This class shall be a subclass of the classes declared as their data types by the variables associated with the **SELF\simple_domain.referenced_by.item_of.defines variables_semantics**.

EXPRESS specification:

```
*)
ENTITY subclass_defined_domain
SUBTYPE OF(simple_domain);
     from_class: class_BSU;
WHERE
     WR1: SIZEOF(SELF\simple_domain.referenced_by.item_of.defines)
          = 1;
     WR2: QUERY(va <* collects_variables(
          SELF\simple_domain.referenced_by.item_of.defines)
          | NOT('ISO13584_LIBRARY_EXPRESSIONS_SCHEMA'
          + '.CLASS_INSTANCE_VARIABLE' IN TYPEOF(va))) = [];
     WR3: QUERY(va <* collects_variables(
          SELF\simple_domain.referenced_by.item_of.defines)
          | NOT is_subclass(SELF.from_class.definition[1],
          va\class_instance_expression.expr_type.definition[1])) = [];
END_ENTITY; -- subclass_defined_domain
(*
```

Attribute definitions:

from_class: the class that defines the domain of the variables associated with the **SELF\simple_domain.referenced_by.item_of.defines variable_semantics**.

Formal propositions:

WR1: this entity shall define the domain of only one **variable_semantics**.

WR2: the variables of which the domain is defined shall be **class_instance_variable**s.

WR3: this class shall be a subclass of the class declared as its data type by the variable associated with the **SELF\simple_domain.referenced_by.item_of.defines variable_semantics**.

10.4.8 Constant_range_defined_domain

A **constant_range_defined_domain** entity is a domain that is defined by its low bound, high bound and interval between allowed values. All of these attributes are expressed as literal numbers. This range includes its bounds if they belong to the data type of the variable.

If the step between allowed values does not exist, the whole set of values belonging to the data type of the variables and that are between the low bound and high bound are allowed.

EXPRESS specification:

```
*)
ENTITY constant_range_defined_domain
SUBTYPE OF(simple_domain);
     minimal: literal_number;
     maximal: literal_number;
     step: OPTIONAL literal_number;
WHERE
```



```
      WR1: SIZEOF(SELF\simple_domain.referenced_by.item_of.defines)
           = 1;
      WR2: minimal.the_value <= maximal.the_value;
      WR3: QUERY(va <* collects_variables(
           SELF\simple_domain.referenced_by.item_of.defines) | NOT
           ('ISO13584_EXPRESSIONS_SCHEMA.NUMERIC_VARIABLE'
           IN TYPEOF(va))) = [];
      WR4: (NOT EXISTS(SELF.step)) OR (SELF.step.the_value > 0);
END_ENTITY; -- constant_range_defined_domain
(*
```

Attribute definitions:

minimal: the value that corresponds to the low bound of the range that defines the domain of the variable associated with the **SELF\simple_domain.referenced_by.item_of.defines variable_semantics**.

maximal: the value that corresponds to the high bound of the range that defines the domain of the variable associated with the **SELF\simple_domain.referenced_by.item_of.defines variable_semantics**.

step: the value that corresponds to the difference between two allowed values of the variable associated with the **SELF\simple_domain.referenced_by.item_of.defines variable_semantics**, the first allowed value being **minimal**.

Formal propositions:

WR1: this entity shall define the domain of only one **variable_semantics**.

WR2: the **minimal** value shall be less or equal to the **maximal** value.

WR3: the **variable** associated with the **SELF\simple_domain.referenced_by.item_of.defines variable_semantics** shall be a **numeric_variable**.

WR4: the **step** value shall be a positive value.

10.4.9 Variable_range_defined_domain

A **variable_range_defined_domain** entity is a domain that is defined by its low bound, high bound and interval between allowed values. All of these attributes are expressed as numeric expressions. This range includes its bounds if they belong to the data type of the variable.

If the step between allowed values does not exist, the whole set of values belonging to the type of the variable and that are between the low bound and high bound are allowed.

EXPRESS specification:

```
*)
ENTITY variable_range_defined_domain
SUBTYPE OF(simple_domain);
      minimal: numeric_expression;
      maximal: numeric_expression;
      step: OPTIONAL numeric_expression;
WHERE
```



```
        WR1: SIZEOF(SELF\simple_domain.referenced_by.item_of.defines)
             = 1;
        WR2: collects_var_sem(used_variables(SELF.minimal)
             + used_variables(SELF.maximal))
             <= SELF\simple_domain.referenced_by.item_of.assumes;
        WR3: NOT(EXISTS(SELF.step)) OR
             (collects_var_sem(used_variables(SELF.step))
             <= SELF\simple_domain.referenced_by.item_of.assumes);
        WR4: QUERY(va <* collects_variables
             (SELF\simple_domain.referenced_by.item_of.defines)
             | NOT('ISO13584_EXPRESSIONS_SCHEMA.NUMERIC_VARIABLE'
             IN TYPEOF(va))) = [];
    END_ENTITY; -- variable_range_defined_domain
    (*
```

Attribute definitions:

minimal: the value that corresponds to the low bound of the range that defines the domain of variables associated with the **SELF\simple_domain.referenced_by.item_of.defines variable_semantics**.

maximal: the value that corresponds to the high bound of the range that defines the domain of the variables associated with the **SELF\simple_domain.referenced_by.item_of.defines variable_semantics**.

step: the value that corresponds to the difference between two allowed values of the variables associated with the **SELF\simple_domain.referenced_by.item_of.defines variable_semantics**, the first allowed value being **minimal**.

Formal propositions:

WR1: this entity shall define the domain of only one **variable_semantics**.

WR2: the **used_variables** in the **numeric_expression**s that define the low bound and the high bound shall belong to the set of variables associated with the **SELF\simple_domain.referenced_by.item_of.assumes variable_semantics**.

WR3: the **used_variables** in the **numeric_expression**s that defines the step, when it exists, shall belong to the set of variables associated with the **SELF\simple_domain.referenced_by.item_of.assumes variable_semantics**.

WR4: the **variable** associated with the **SELF\simple_domain.referenced_by.item_of.defines variable_semantics** shall be a **numeric_variable**.

Informal propositions:

IP1: the **step** expression shall evaluate to a positive value.

IP2: the value of the **minimal** expression shall be less or equal to the value of the **maximal** expression.

10.4.10 Predicate_defined_domain

A **predicate_defined_domain** enables one to specify constraints between a set of variables that shall evaluate to TRUE for any allowed set of values of these variables. Such **simple_domain**s enable further restriction on the domains defined constructively using the previous **simple_domain** subtypes.

EXPRESS specification:

```
*)
ENTITY predicate_defined_domain
SUBTYPE OF(simple_domain);
     constraint: boolean_expression;
WHERE
     WR1: collects_var_sem(used_variables(SELF.constraint))
          <= SELF\simple_domain.referenced_by.item_of.defines +
          SELF\simple_domain.referenced_by.item_of.assumes;
END_ENTITY; -- predicate_defined_domain
(*
```

Attribute definitions:

constraint: the **boolean_expression** that shall evaluate to TRUE for any allowed set of values of the variables associated with the **SELF\simple_domain.referenced_by.item_of.defines** and **SELF\simple_domain.referenced_by.item_of.assumes variable_semantics**.

Formal propositions:

WR1: the variables used in the **constraint** expression shall correspond to **variable**s associated with the **SELF\simple_domain.referenced_by.item_of.defines** and **SELF\simple_domain.referenced_by.item_of.assumes variables_semantics**.

10.4.11 Functional_domain_restriction

A **functional_domain_restriction** is a subtype of a **simple_domain** that defines a domain that degenerates into one singleton. The role of such a domain is to specify that the values of the variables associated with the **defines** attribute of a **domain_restriction** may be automatically computed by a system when the value associated with the **assumes** attribute of this **domain_restriction** are fixed.

When a **functional_domain_restriction** consists of several **guarded_domain**s, each one shall be a **guarded_functional_domain**.

EXPRESS specification:

```
*)
ENTITY functional_domain_restriction
SUBTYPE OF(domain_restriction);
     SELF\domain_restriction.domains:
          SET[1:?] OF guarded_functional_domain;
END_ENTITY; -- functional_domain_restriction
(*
```


Attribute definitions:

domains: the set of **guarded_functional_domain**s that specify the different parts of the domain.

10.4.12 Guarded_functional_domain

A **guarded_functional_domain** is a **guarded_simple_domain** whose **domain** attribute evaluates to a singleton. This singleton is specified by means of a **simple_functional_domain**.

EXPRESS specification:

```
*)
ENTITY guarded_functional_domain
SUBTYPE OF(guarded_simple_domain);
     SELF\guarded_simple_domain.domain: simple_functional_domain;
END_ENTITY; -- guarded_functional_domain
(*
```

Attribute definitions:

SELF\guarded_simple_domain.domain: the function that may be used if the **guard** evaluates to TRUE. In the case of several **guard**s evaluating to TRUE, any **simple_functional_domain** of which the **guard** evaluates to TRUE shall define the domain.

10.4.13 Simple_functional_domain

A **simple_functional_domain** defines a domain for variable(s) as a singleton of which the value is specified as a function. The expression of the function is constructive. Therefore the value of this variable may be computed by some software system as soon as the variables involved in the domain of the function are assigned a value.

NOTE The concept of **simple_functional_domain** enables specification that some functional dependencies exist between two sets of variables. When this resource is used in some information model, it shall specify whether the function shall be evaluated as soon as the variables of its domain are assigned a value.

EXPRESS specification:

```
*)
ENTITY simple_functional_domain
ABSTRACT SUPERTYPE OF(ONEOF(
     library_expression_defined_value,
     table_defined_value,
     null_defined_value))
SUBTYPE OF(simple_domain);
END_ENTITY; -- simple_functional_domain
(*
```

10.4.14 Library_expression_defined_value

A **library_expression_defined_value** is a **simple_functional_domain** that contains one unique value that is defined by a **library_expression**. Such a **simple_functional_domain** may be used to define the derivation function that specifies the value of any variable of which data type belongs to one of the data types defined for properties in the **ISO13584_IEC61360_dictionary_schema**.

EXPRESS specification:

```
*)
ENTITY library_expression_defined_value
SUBTYPE OF(simple_functional_domain);
      its_value: library_expression;
WHERE
      WR1: SIZEOF(SELF\simple_domain.referenced_by.item_of.defines)
           = 1;
      WR2: QUERY(va <* collects_variables
           (SELF\simple_domain.referenced_by.item_of.defines)
           | NOT compatible_variable_and_library_expression
           (va, SELF.its_value)) = [];
      WR3: collects_var_sem(used_variables(SELF.its_value))
           <= SELF\simple_domain.referenced_by.item_of.assumes;
END_ENTITY; -- library_expression_defined_value
(*
```

Attribute definitions:

its_value: the **library_expression** that specifies the unique value of the domain of the variables associated with the **SELF\simple_domain.referenced_by.item_of.defines variable_semantics**.

Formal propositions:

WR1: this entity shall define the value of only one **variable_semantics**.

WR2: the type of the **library_expression** shall be compatible with the type of the **library_variable** associated with the **SELF\simple_domain.referenced_by.item_of.defines variable_semantics**.

WR3: all the **variable_semantics** associated to a variable in the **its_value** expression shall belong to **SELF\simple_domain.referenced_by.item_of.assumes** set.

10.4.15 Table_defined_value

A **table_defined_value** is a **simple_functional_domain** that contains one unique tuple value, a tuple, that is defined by a line of a **table_expression**. The line of the **table_expression** is specified by the values of the **SELF\simple_domain.referenced_by.item_of.assumes** set of **variable_semantics** that shall include the key of the table.

EXPRESS specification:

```
*)
ENTITY table_defined_value
SUBTYPE OF(simple_functional_domain);
      from_table: table_expression;
WHERE
      WR1: SELF\simple_domain.referenced_by.item_of.defines
           <= list_to_set(SELF.from_table\table_expression.its_columns);
      WR2: SELF\simple_domain.referenced_by.item_of.assumes
           >= SELF.from_table\table_expression.the_key;
      WR3: QUERY(sem <* collects_var_sem(
           used_variables(SELF.from_table))
```



```
            | NOT((sem IN
            (SELF\simple_domain.referenced_by.item_of.assumes
            + SELF\simple_domain.referenced_by.item_of.defines))
            OR ((('ISO13584_TABLE_RESOURCE_SCHEMA'
            + '.COLUMN_TRAVERSAL_VARIABLE_SEMANTICS')
            IN TYPEOF(sem))) AND
            (sem\column_traversal_variable_semantics.domain
            IN collects_columns(from_table)))) = [];
END_ENTITY; -- table_defined_value
(*
```

Attribute definitions:

from_table: the **table_expression** that defines the table that specifies the domain.

Formal propositions:

WR1: the **variable_semantics** defined in the **SELF\simple_domain.referenced_by.item_of.defines** attribute shall belong to the columns of the table, i.e., to its **SELF.from_table\table_expression.its_columns** attribute.

WR2: the **variable_semantics** defined in the **SELF\simple_domain.referenced_by.item_of.assumes** attribute shall contain the key of table defined by the **SELF.from_table\table_expression.the_key** attribute.

WR3: the **used_variables** in the **table_expression** shall belong to the set of variables associated with the **SELF\simple_domain.referenced_by.item_of.defines variable_semantics** or to the **SELF\simple_domain.referenced_by.item_of.assumes variable_semantics**, or they shall be column iterators associated with **column_traversal_variable_semantics** that reference one of the columns of the tables involved in the **from_table table_expression**.

10.4.16 Null_defined_value

A **null_defined_value** indicates that no value may be assigned to a variable.

NOTE A **null_defined_value** shall only be used when the value associated with a **variable_semantics** is defined as optional. When a **null_defined_value** defines the domain of a variable, any variable associated with this **variable_semantics** shall not have any value.

EXAMPLE In the **ISO13584_library_content_schema** a property may be defined as optional. If the domain associated with such a property is a **null_defined_value**, the meaning is that the property does not exist.

EXPRESS specification:

```
*)
ENTITY null_defined_value
SUBTYPE OF(simple_functional_domain);
WHERE
     WR1: SIZEOF(SELF\simple_domain.referenced_by.item_of.defines)
          = 1;
END_ENTITY; -- null_defined_value
(*
```


Formal propositions:

WR1: this entity shall define the value of only one **variable_semantics**.

10.5 ISO13584_domain_resource_schema function definitions

This subclause contains the EXPRESS function definitions in the **ISO13584_domain_resource_schema**.

10.5.1 Collects_variables function

The **collects_variables** function collects all the **generic_variables** associated with a list of **variable_semantics**. It uses the defined function **syntax_of** defined in the **ISO13584_library_expressions_schema**.

EXPRESS specification:

```
*)
FUNCTION collects_variables(v_sem: AGGREGATE OF variable_semantics):
     SET OF generic_variable;

LOCAL
     l: SET OF generic_variable := [];
END_LOCAL;

REPEAT i := 1 TO SIZEOF(v_sem);
     l := l + syntax_of(v_sem[i]);
END_REPEAT;

RETURN(l);

END_FUNCTION; -- collects_variables
(*
```

10.5.2 Collects_var_sem function

The **collects_var_sem** function collects all the **variable_semantics** associated with a list of **generic_variable**s. It uses the function **semantics_of** defined in **ISO13584_library_expressions_schema**.

EXPRESS specification:

```
*)
FUNCTION collects_var_sem(va: AGGREGATE OF generic_variable):
     SET OF variable_semantics;

LOCAL
     l: SET OF variable_semantics := [];
END_LOCAL;

REPEAT i := 1 TO SIZEOF(va);
     l := l + semantics_of(va[i]);
END_REPEAT;
```



```
RETURN(l);

END_FUNCTION; -- collects_var_sem
(*
```

10.5.3 Used_tables_in_domain function

The **used_tables_in_domain** function performs a traversal of the whole expression graph and returns the **table_identification**s of all the **table_literal** used.

EXPRESS Specification:

```
*)
FUNCTION used_tables_in_domain(arg: domain_restriction):
     SET OF table_identification;

LOCAL
     result: SET OF table_identification := [];
END_LOCAL;

REPEAT i := 1 TO SIZEOF(arg.domains);
     IF ('ISO13584_EXPRESSIONS_SCHEMA.BOOLEAN_EXPRESSION'
          IN TYPEOF(arg.domains[i].guard))
     THEN
          result := result + used_table_literals
               (arg.domains[i].guard);
     END_IF;

     IF ('ISO13584_DOMAIN_RESOURCE_SCHEMA.TABLE_DEFINED_DOMAIN'
          IN TYPEOF(arg.domains[i].domain))
     THEN
          result := result + used_table_literals
               (arg.domains[i].domain\table_defined_domain.from_table);
     END_IF;

     IF ('ISO13584_DOMAIN_RESOURCE_SCHEMA.PREDICATE_DEFINED_DOMAIN'
          IN TYPEOF(arg.domains[i].domain))
     THEN
          result := result + used_table_literals(
          arg.domains[i].domain\predicate_defined_domain.constraint);
     END_IF;

     IF ('ISO13584_DOMAIN_RESOURCE_SCHEMA'
          +'.LIBRARY_EXPRESSION_DEFINED_VALUE' IN TYPEOF(
          arg.domains[i].domain))
     THEN
          result := result + used_table_literals(
               arg.domains[i].domain\library_expression_defined_value
               .its_value);
     END_IF;
```



```
      IF ('ISO13584_DOMAIN_RESOURCE_SCHEMA.TABLE_DEFINED_VALUE'
           IN TYPEOF(arg.domains[i].domain))
      THEN
           result := result + used_table_literals
                (arg.domains[i].domain\table_defined_value.from_table);
      END_IF;
 END_REPEAT;

 RETURN(result);

 END_FUNCTION; -- used_tables_in_domain
 (*
```

10.5.4 Used_variables_in_domain function

The **used_variables_in_domain** performs a traversal of the whole expression graph and returns the **generic_variable**s of all the **domain_restriction**s used.

EXPRESS Specification:

```
 *)
 FUNCTION used_variables_in_domain(arg: domain_restriction):
      SET OF generic_variable;

 LOCAL
      result: SET OF generic_variable := [];
 END_LOCAL;

 REPEAT i := 1 TO SIZEOF(arg.domains);
      IF ('ISO13584_EXPRESSIONS_SCHEMA.BOOLEAN_EXPRESSION'
           IN TYPEOF(arg.domains [i].guard))
      THEN
           result := result + used_variables(arg.domains [i].guard);
      END_IF;

      IF ('ISO13584_DOMAIN_RESOURCE_SCHEMA.TABLE_DEFINED_DOMAIN'
           IN TYPEOF(arg.domains [i].domain))
      THEN
           result := result + used_variables(
                arg.domains [i].domain\table_defined_domain
                .from_table);
      END_IF;

      IF ('ISO13584_DOMAIN_RESOURCE_SCHEMA.PREDICATE_DEFINED_DOMAIN'
           IN TYPEOF(arg.domains [i].domain))
      THEN
           result := result + used_variables(
                arg.domains[i].domain\predicate_defined_domain
                .constraint);
      END_IF;
```

```
        IF ('ISO13584_DOMAIN_RESOURCE_SCHEMA' +
                                '.LIBRARY_EXPRESSION_DEFINED_VALUE'
             IN TYPEOF(arg.domains [i].domain))
        THEN
             result := result + used_variables(
                  arg.domains[i].domain\
                  library_expression_defined_value.its_value);
        END_IF;

        IF ('ISO13584_DOMAIN_RESOURCE_SCHEMA.TABLE_DEFINED_VALUE'
             IN TYPEOF(arg.domains [i].domain))
        THEN
             result := result + used_variables(
                  arg.domains [i].domain\table_defined_value.from_table);
        END_IF;
   END_REPEAT;

   RETURN(result);

   END_FUNCTION; -- used_variables_in_domain
   (*
```

10.5.5 Variables_belong_to_assumes function

The **variables_belong_to_assumes** function checks that only variables that are associated with the **variable_semantics** of the **assumes** set of a **domain_restriction** may be used to define the **guard**s of its **guarded_simple_domain**s.

EXPRESS Specification:

```
   *)
   FUNCTION variables_belong_to_assumes(gsd: guarded_simple_domain):
        LOGICAL;

   IF ('ISO13584_EXPRESSIONS_SCHEMA.BOOLEAN_EXPRESSION')
        IN TYPEOF(gsd)
   THEN
        RETURN(collects_var_sem(used_variables(gsd.guard))
             <= gsd.item_of.assumes);
   ELSE
        RETURN(TRUE);
   END_IF;

   END_FUNCTION; -- variables_belong_to_assumes
   (*

   *)
   END_SCHEMA; -- ISO13584_domain_resource_schema
   (*
```


11 ISO13584_extended_dictionary_schema

This clause defines the requirement for the **ISO13584_extended_dictionary_schema**. The following EXPRESS declaration introduces the **ISO13584_extended_dictionary_schema** block and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA ISO13584_extended_dictionary_schema;

REFERENCE FROM ISO13584_IEC61360_dictionary_schema
     (all_class_descriptions_reachable,
     basic_semantic_unit,
     class,
     class_BSU,
     class_BSU_relationship,
     class_instance_type,
     class_related_BSU,
     code_type,
     component_class,
     compute_known_visible_data_types,
     compute_known_visible_properties,
     content_item,
     data_type,
     data_type_BSU,
     data_type_element,
     definition_available_implies,
     definition_type,
     dictionary_element,
     document,
     entity_instance_type,
     graphics,
     item_class,
     item_names,
     level,
     level_type,
     list_to_set,
     material_class,
     named_type,
     non_quantitative_code_type,
     non_quantitative_int_type,
     note_type,
     property_BSU,
     property_DET,
     remark_type,
     revision_type,
     sep_cv,
     sep_id,
     supplier_BSU,
     supplier_BSU_relationship,
     supplier_element,
```



```
      supplier_related_BSU,
      value_domain,
      version_type);

  REFERENCE FROM ISO13584_IEC61360_language_resource_schema
      (present_translations);

  REFERENCE FROM ISO13584_generic_expressions_schema
      (variable_semantics);

  REFERENCE FROM ISO13584_instance_resource_schema
      (compatible_level_type_and_instance,
      property_or_data_type_BSU);

  REFERENCE FROM ISO13584_table_resource_schema
      (class_instance_column,
      column,
      entity_instance_column,
      RDB_table_extension,
      RDB_table_specification,
      table_extension,
      table_identification,
      table_specification);

  REFERENCE FROM ISO13584_variable_semantics_schema
      (property_semantics,
      self_property_semantics);

  REFERENCE FROM ISO13584_external_file_schema
      (external_file_protocol,
      simple_program_protocol);

  REFERENCE FROM date_time_schema
      (year_number);

  REFERENCE FROM support_resource_schema
      (identifier,
      label);

  REFERENCE FROM person_organization_schema
      (organization ,
      person);

  REFERENCE FROM application_context_schema
      (application_protocol_definition);
  (*
```

NOTE The schemas referenced above can be found in the following documents:
ISO13584_IEC61360_dictionary_schema IEC 61360-2
(which is duplicated for convenience in informative annex D of ISO 13584-42:1998),
ISO13584_IEC61360_language_resource_schema IEC 61360-2
(which is duplicated for convenience in informative annex D of ISO 13584-42),
ISO13584_generic_expressions_schema ISO 13584-20

ISO13584_instance_resource_schema	This part of ISO 13584,
ISO13584_table_resource_schema	This part of ISO 13584,
ISO13584_variable_semantics_schema	This part of ISO 13584,
ISO13584_external_file_schema	This part of ISO 13584,
date_time_schema	ISO 10303-41,
support_resource_schema	ISO 10303-41,
person_organization_schema	ISO 10303-41,
application_context_schema	ISO 10303-41.

11.1 Introduction to the ISO13584_extended_dictionary_schema

The **ISO13584_extended_dictionary_schema** defines the extensions to the ISO/IEC common dictionary schema that enable the exchange of ISO 13584 dictionary data.

It extends the the ISO/IEC common dictionary schema in the following four ways:

a) The ISO/IEC common dictionary schema models hierarchies of classes of items, components and materials. The extended dictionary schema also provides a specialisation for modelling features. This specialisation of **item_class** is represented as a **feature_class**.

EXAMPLE 1 A form feature may be represented as an instance of a **feature_class**. It is associated with dimensioning properties and captures one aspect of the shape of a component.

EXAMPLE 2 In a piping component, an outlet is associated with properties (for instance, its name, its role) and captures one aspect of the definition of a component. It may be represented as an instance of a **feature_class**.

b) The ISO/IEC common dictionary schema provided for modelling **item_class**. The extended dictionary schema defines two new subtypes of class: **functional_model_class** and **functional_view_class**. A functional view is the representation of an item in product data. A functional model is the representation of an item in a library.

c) The ISO/IEC common dictionary schema defined three kinds of properties that might be used to describe parts or items: **condition_DET**, **non_dependent_P_DET** and **dependent_P_DET**. The extended dictionary schema introduces a new subtype of **property_DET** called **representation_P_DET** to represent properties defined in functional models and functional views.

NOTE 1 It is necessary to introduce **representation_P_DET** because such a property is not associated with a part, or an item, but with a representation of a part or item. Therefore, the life cycle of a **representation_P_DET** value is independent from the life cycle of any part or item.

d) The ISO/IEC common dictionary schema only associated properties and data types with classes. The extended dictionary schema defines tables and documents that can be associated with classes.

The **ISO13584_extended_dictionary_schema** models:

— the total structure of the elements that belong to an ISO 13584-conformant dictionary; this overall structure specifies the different elements that shall be processed to be stored in an existing user dictionary;

— the seven kinds of elements whose references and descriptions are intended to be stored in a user dictionary; these seven kinds of elements are: supplier, program library, class, property, data type, table and document;

— those properties that are used in classes of functional models and classes of functional views for the computation or representation of a functional view of some item;

— the content of the tables referenced in a dictionary.

The **ISO13584_extended_dictionary_schema** does not model:

— the set of allowed instances of a class (class extension) of which reference and description are intended to be stored in a user dictionary;

— the content of program libraries and documents of which references and descriptions are intended to be stored in a user dictionary but of which contents are represented as external files.

NOTE 2 The representations of the external files that may be associated with an ISO 13584-conformant library are defined in the **ISO13584_external_file_schema**.

11.2 Fundamental concepts and assumptions for the ISO13584_extended_dictionary_schema

11.2.1 Dictionary structure

The **ISO13584_extended_dictionary_schema** describes hierarchies of classes with simple inheritance intended to be stored in a user dictionary. Three kinds of classes are defined:

a) an **item_class** is used to define stand-alone and identifiable kinds of objects. **item_class** has three subtypes: **component_class**, **material_class** and **feature_class**.

NOTE **Component_class** and **material_class** are defined in the ISO/IEC common dictionary schema in ISO 103584-42. **Feature_class** is defined in the **ISO13584_extended_dictionary_schema** in clause 11.18 of this part of ISO 13584.

b) a **functional_model_class** is used to record a mechanism that may generate representations of these identifiable kinds of objects

EXAMPLE A parametric program is a mechanism that may generate geometric representations. A parametric program may be recorded in a functional model class.

c) a **functional_view_class** is used to characterize each kind of possible representation of an **item_class**, whatever be the **item_class**.

11.2.2 Class related elements

Four categories of elements are associated with classes:

a) properties, of which the values are used to characterise the instances;

b) tables, that describe relations between properties;

c) documents, that provide human readable information about classes and properties;

d) named data types that may be used to specify the domain of different properties.

These four categories of elements are denoted class related elements.

11.2.3 Supplier related elements

One category of elements is associated with a library data supplier: a **program_library** is a set of programs delivered by this library data supplier. Once processed by a LMS, such a library may be referenced by programs provided as library external files, as a **linked_interface_program_protocol**.

EXAMPLE ISO 13584-31 defines a programming interface (API) for parametric geometry. An upward compatibility layer that enables processing programs based on another parametric geometry API may be defined as a **program_library**. Referencing a **linked_interface_program_protocol** that includes this **program_library** allows the processing of programs based on this other parametric geometry API.

11.2.4 Three-fold description of dictionary elements

The **ISO13584_extended_dictionary_schema** dictionary conforms to the three levels architecture defined in ISO 13584-42 clause D.3.3.2.

NOTE In the **ISO13584_IEC61360_dictionary_schema**, each concept is modelled by three entities:

— its **basic_semantic_unit** that identifies the element, and enables reference to it, whether or not its description and content are available in the same exchange context, and

— its **dictionary_element** that provides a human-readable and computer-sensible definition of the element, and

— its **content_item** that contains the (possible) additional computer-data that represent the element in a data processing universe.

11.2.5 Unique identification of dictionary elements

Each library data supplier has a unique identification as defined in ISO 13584-26.

Each **class** or **program_library** shall be assigned a unique code by the library data supplier who defines it.

To provide for unique identification of any other dictionary element, its **basic_semantic_unit** contains, as a **name_scope** attribute, the root of the class tree where it is visible. For each category of dictionary elements, a unique identification may therefore be derived from:

— its code,

— the code of the class that corresponds to its name scope, and,

— the code of the library data supplier that defined this class.

This mechanism is only an identification mechanism. It does not define which elements are applicable to a particular class.

11.2.6 Applicable elements

Applicability is inherited. That is, if a **class_related_dictionary_element** is applicable to a class C, then it is also applicable to every subclass of C.

NOTE To ensure a full compatibility between the data based on the **ISO13584_IEC61360_dictionary_schema** and those that are based on the ISO 13584 extensions, two different mechanisms are used to specify the class related elements applicable to a class. Applicable properties and types are specified by class attributes. Applicable tables and documents are specified using the entity relationship approach. In the receiving system, a different implementation may be used to improve performance.

Visibility rules and importation rules are defined to specify which elements are permitted for explicit reference in a class description.

11.2.7 Visibility rule

Visibility is inherited. A class related element is visible from a class if this class corresponds to the **name_scope** of the class related element or if this class is a subclass of the class that corresponds to the **name_scope** of the class related element.

Any visible element may be used to describe a class and therefore may become an element applicable to the class, and to all of its subclasses.

11.2.8 Semantic relationships between classes

In the object oriented approach, the definition of a class C1 may depend on the definition of another class C2. When the definition of C2 is not available, C1 is not completely defined. Such relationships are called semantic relationships between classes.

Beside the inheritance relationship (C1 is a subclass of C2) and the aggregation relationship (C2 is the data type of a property defined in C1), two other semantics relationships are defined by ISO 13584.

a) The *is-case-of* relationship enables to specify that some of the properties that apply to C2 apply also to C1, although they do not belong to the same inheritance tree;

NOTE 1 ISO 13584 supports only simple inheritance. The *is-case-of* relationship enables for instance a particular supplier to specify that one of his/her class, belonging to his/her particular inheritance tree, is also *case-of* class(es) defined by some standardisation organisation(s).

b) The *is-view-of* relationship enables to specify that a class C1 contains method(s) able to create representation(s) of the instances of a class C2 when the values of some properties of these instances are provided as input parameter of this (these) method(s).

This part of ISO 13584 contains two different approaches for modelling such relationships.

a) In the a priori approach, a class C1 may be defined as having a particular semantic relationship with a class C2. In this case, it may import properties and data types that are visible or applicable for C2. This explicit importation makes these properties applicable to C1. But, if class C2 is not delivered together with class C1, and if it is not already available on the receiving system, class C1 cannot be processed.

NOTE 2 The library data supplier responsible for class C1 may decide to provide together with class C1, those information elements that describe class C2 and/or properties, documents, data types and tables associated with class C2. These information elements reference, as their **supplier_bsu**, their own library data supplier, and cannot be guaranted by the library data supplier of class C1. These information elements are useless when class C2 is already available in the user library.

EXAMPLE Assume that a particular component supplier provides resistors that comply with the concept of a resistor as defined in IEC 61630-4. When describing its resistor family in an ISO 13584-conformant library, this supplier may decide to assume that the receiving user library should contain all the classes and properties defined by IEC 61630-4. In this case, the resistor family is defined as an **item_class_case_of**. This class imports the needed properties (through their **property_BSU**) without having to describe them by means of a **property_DET** dictionary element. But, if the user library does not contain all the classes and properties defined in IEC 61630-4, and if their dictionary definitions are not provided in the library of the component supplier, the referenced properties may be unknown by the receiving system. In this case, the supplier class would not be completely defined. According to the logical description of the compiling process of ISO 13584-conformant dictionaries and libraries, defined in annex O, the compiling of this class should fail.

b) In the a posteriori approach, the C1 class does not include any reference to C2 in its description. C1 duplicates the complete description (i.e., a different **property_BSU** and a **property_DET**) for each of the properties that correspond to a property defined in C2. The semantic relationship is defined by a third entity, an **a_posteriori_semantic_relationship**. This entity references C1 and C2 and describes the set of couples of properties that correspond to each other, one being applicable to C1 and the other being applicable to C2. The overhead is to re-describe properties that might be already available. The advantage is that C1 can always be processed on the receiving system. Only the compiling of the **a_posteriori_semantic_relationship** fails if class C2 is not available on the receiving system.

NOTE 3 A posteriori relationships may also be introduced on the library end-user site, as part of the user customisation of supplier libraries.

11.2.9 A priori semantic relationships and importation rule

A class defined through an a priori semantic relationship with another class may refer, in its

description, to the properties, data types, tables or documents that are visible or applicable to this other class. This reference imports these properties, data types, tables or documents into the referencing class and makes them applicable to the class that imports them and usable for describing the class instances.

This importation shall always be explicit, i.e., done through an attribute, or a relation, that specifies which properties, data types, tables or documents are imported.

The following a priori semantic relationships are specified:

— is-case-of relationship for an **item_class**: when a class C is defined as **item_class_case_of** referring to a class D, C may import properties, data types, tables and documents from D;

— is-case-of relationship for a **functional_model_class**: a functional model class may refer, through its **case_of** attributes to other functional model classes. In this case it may import properties, data types, tables and documents from these functional model classes;

— when a class C is a **functional_model_class** that creates a functional view V, C may import properties, data types, tables and documents from V;

— is-view-of relationship: when a class C is a **fm_class_view_of** referring to an **item_class** D, C may import properties, data types, tables and documents from D.

11.2.10 Type checking for the tables referenced in the dictionary

In the **ISO13584_table_resource_schema**, type checking for a table is achieved by asserting that the type of a **column** shall be type-compatible with any **library_variable** associated with the corresponding **variable_semantic**s.

A table referenced in a dictionary contains only columns that refer to **self_property_semantic**s of which the data types are explicitly declared in the corresponding **property_DET**s. Type checking is performed:

— by asserting that any **library_variable** associated with a **property_semantic**s shall be type-compatible with the data type declared for this property (this is already documented in the **ISO13584_variable_semantics_schema**), and

— by asserting that, if the **table_extension** of a table is available, the data type of each column shall be type compatible with the declared data type of the corresponding **self_property_semantics** (this is documented in the **table_content** entity).

11.3 ISO13584_extended_dictionary_schema constant definitions

11.3.1 Element_code_len

element_code_len is the maximum length for the code associated with a subtype of **dictionary_element** defined in the **ISO13584_extended_dictionary_schema**. This length allows the building of an identifier conformant with ISO 9075: SQL.

NOTE This identifier is the concatenation of the **dictionary_element** code and version.

EXPRESS specification:

```
*)
CONSTANT
      element_code_len: INTEGER := 14;
(*
```


11.3.2 Dictionary_code_len

dictionary_code_len is the maximum length for the code that identifies a dictionary.

NOTE This identifier is the concatenation of the **dictionary_element** code and version.

EXPRESS specification:

```
*)
     dictionary_code_len: INTEGER := 80;
END_CONSTANT;
(*
```

11.4 ISO13584_extended_dictionary_schema type definitions

This clause introduces the type definitions in the **ISO13584_extended_dictionary_schema**.

11.4.1 Document_code_type

A **document_code_type** is a code associated with a document.

EXPRESS specification:

```
*)
TYPE document_code_type = code_type;
WHERE
     WR1: LENGTH(SELF) <= element_code_len;
END_TYPE; -- document_code_type
(*
```

Formal propositions:

WR1: the length of the code shall be less or equal to the value of **element_code_len**.

11.4.2 Program_library_code_type

A **program_library_code_type** is a code associated with a program library.

EXPRESS specification:

```
*)
TYPE program_library_code_type = code_type;
WHERE
     WR1: LENGTH(SELF) <= element_code_len;
END_TYPE; -- program_library_code_type
(*
```

Formal propositions:

WR1: the length of the code shall be less or equal to the value of **element_code_len**.

11.4.3 Table_code_type

A **table_code_type** is a code associated with a table.

EXPRESS specification:

```
*)
TYPE table_code_type = code_type;
WHERE
     WR1: LENGTH(SELF) <= element_code_len;
END_TYPE; -- table_code_type
(*
```

Formal propositions:

WR1: the length of the code shall be less or equal to the value of **element_code_len**.

11.4.4 Absolute_URL_type

An **absolute_URL_type** is a string, the structure of which is specified by IAB RFC 1739 and that identifies the absolute location where a resource may be found on the Internet, and the protocol that shall be used to access resources.

EXPRESS specification:

```
*)
TYPE absolute_URL_type = identifier;
WHERE
     WR1: SELF LIKE '*://*';
END_TYPE; -- absolute_URL_type
(*
```

Formal propositions:

WR1: an **absolute_URL_type** shall contain a semi-colon followed by the string '//'.

NOTE 1 The Internet protocol shall be specified in an **absolute_URL_type**, and the address of the server shall be part of an **absolute_URL_type**.

NOTE 2 Parties to agreements based on this part of ISO 13584 are encouraged to investigate the possibility of applying the most recent IAB RFC belonging to the Standards Track RFC and that updates IAB RFC 1739.

11.4.5 Dictionary_code_type

A **dictionary_code_type** is a code associated with a dictionary.

EXPRESS specification:

```
*)
TYPE dictionary_code_type = code_type;
WHERE
     WR1: LENGTH(SELF) <= dictionary_code_len;
```



```
END_TYPE; -- dictionary_code_type
(*
```

Formal propositions:

WR1: the length of the code shall be less or equal to the value of **dictionary_code_len**.

11.5 ISO13584_extended_dictionary_schema identification of a dictionary

A **dictionary_identification** entity allows to identify unambiguously a particular version of a particular dictionary of a particular supplier, whether information on this dictionary is available in some exchange context or not. It contains a **code** defined by the library data supplier which defines the dictionary, and a **version** version number and a **revision** revision number that characterize a particular state of this dictionary.

The **version** shall be incremented whenever the **version** of any class described in this dictionary is incremented. When the **version** is incremented, the **revision** of the corresponding **dictionary_identification** shall be reset to '000'.

NOTE 1 The **version** of a class is incremented when any change occurs in the dictionary definition of this class that influence its use, and when the class has a content and this content changes. These cases are specified in clause 8.3 of ISO 13584-42 and in clause 12.2.4 of this part of ISO 13584.

The **revision** revision number shall be increased when ever:

— the revision number of some classes described in this dictionary is increased and no class version number is increased, and

— any other piece of information represented in the dictionary is changed, but the revision numbers and version numbers of the classes described in this dictionary.

NOTE 2 A new revision number of a class is defined whenever a change in the attributes that describe this class influences neither its meaning nor its use. These cases are specified in clause 8.3 of ISO 13584-42.

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;
INVERSE
     definition: SET [0:1] of dictionary FOR identified_by;
UNIQUE
     UR1: absolute_id;
END_ENTITY;
(*
```


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.

definition: a possible dictionary entity that describes whole or part of the dictionary.

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 Representation of the standard numbers of standard documents is specified in clauses 5.1 and 5.2 of ISO 13584-26.

11.6 ISO13584_extended_dictionary_schema overall architecture of a dictionary

A **dictionary** entity gives the summary information about a dictionary. This **dictionary** entity may either describe whole or part of the dictionary identified by a **dictionary_identification** entity, or it may describe a dictionary that is not associated with any particular **dictionary_identification** entity. When identified by a **dictionary_identification** entity, the **dictionary** entity specifies, through its **is_complete** Boolean attribute, whether it describes the whole **dictionary_identification** dictionary, or it describes only part of it. When the **dictionary** entity only describes part of the whole **dictionary_identification** dictionary, it may define the dictionary that is required to be available on the receiving system to be able to create the complete content of the **dictionary_identification** dictionary from the content of the **dictionary** entity. The dictionary required to be available on the receiving system is specified by the **updates** attribute. A **dictionary** entity may also specify particular rules for combining the entities referenced by the **dictionary** entity with the content of the preexisting **updates** dictionary to build the **dictionary_identification** dictionary. This is done by the **update_agreement** attribute.

When **updates** exists and **update_agreement** does not exists, this means that all the **basic_semantic_unit**s, **dictionary_element**s or **content_item**s that belongs to the **identified_by** dictionary and that are not provided as part of the content of the **dictionary** entity, already belonged, in the same version, to the **updates** dictionary. Thus, the **identified_by** dictionary may be built from the **updates** dictionary and from the up-dating entities provided as part of the context of the **dictionary** entity.

The **dictionary** entity may also specify, by means of its **referenced_dictionaries** attributes, whether it references some other dictionaries that are identified by **dictionary_identification** entities.

NOTE 1 Reference to **dictionary_identification** entities allows to specify particular versions and/or revisions of referenced dictionaries. When this information is not needed, a dictionary is not required to reference any **dictionary_identification** entity.

A **dictionary** entity specifies the precise information model of the library delivery file in which a **dictionary** entity instance is included by means of two attributes. The **library_structure** attribute

references the library integrated information model of which the library delivery file contains a population. This reference is done by means of an **library_iim_identification** entity instance that allows to exactly specify a version of the corresponding EXPRESS schema. The **supported_vep** specifies which view exchange protocols are referenced by the library delivery file. All the constraints defined in these view exchange protocols shall be fulfilled by the library delivery file content.

NOTE 2 A view exchange protocol may specify how it should be referenced from a library delivery file by means of an EXPRESS schema that consists only of constraints. These constraints are fulfilled by any library delivery file that references this view exchange protocol in any of its conformance class.

EXAMPLE 1 Annex C specifies how to build the complete information model of a library delivery file that reference **ISO13584_g_m_iim_schema** and a view exchange protocol "V1".

A **dictionary** entity contains one particular **supplier_BSU** that identifies the library data supplier who is responsible for the dictionary content. Only the **dictionary_element**s and the **content_item**s associated with basic semantic units defined by this library data supplier are guaranteed by this library data supplier. Other **dictionary_element**s or **content_item**s, if any, are only provided for convenience and should not be recorded in the user integrated library.

NOTE 3 This part of ISO 13584 does not forbid to deliver **dictionary_element**s or **content_item**s defined by supplier B within a **dictionary** of which supplier A is responsible. But this part of ISO 13584 does not contain any provision to ensure correctness or updating correctness of such pieces of information.

NOTE 4 **basic_semantic_unit, dictionary_element**, **content_item** and the **defined_by** attribute of **class_BSU** are defined in ISO 13584-42.

NOTE 5 A **dictionary** entity, or one of its subtype, is required to be instantiated in a library delivery file for all the conformance classes of the library integrated information models defined in this part of ISO 13584, but the conformance class 0 intended to be compatible with IEC 61360-2.

NOTE 6 Conformance requirements are defined in clauses 15, 16, 17 and 18 of this part of ISO 13584.

A dictionary also contains a set of **supplier_BSU**s that are the other library data suppliers referred to in this dictionary. It contains a list of **class_BSU**s that specifies the ordered list of classes referred to, or contained. The order of the list shall be such that when a **class_BSU** is associated with a **class**, the **class_BSU**s referred to either directly or indirectly by this **class** shall always precede the referring **class_BSU** in the **contained_classes** list.

NOTE 7 This order is only intended to facilitate the compiling process of a dictionary if the implementation uses a procedural compiling process such as the one defined in the informative annex O. In this case, it avoids any forward references. This order doesn't need to be followed provide that the result of the compiling process is the same that the one that would result from the process defined in the informative annex O.

A **dictionary** finally defines the set of protocols that are referenced by the elements in this dictionary, the set of possible view exchange protocols that specify the library external files, the set of possible **a_posteriori_semantic_relationship**s between library classes. A **dictionary** also contains various informative items.

A library exchange context conformant to one of the LIIM specified in this part of ISO 13584 shall contain only one **dictionary**.

A **class_BSU** is referred to directly by a **class** through the following attributes:

— **class.its_superclass**: *is-a* relationship;

and, if the class is defined by means of an **a_priori_semantic_relationship**:

— **class\a_priori_semantic_relationship.referenced_classes**.

A **class_BSU** is referred to indirectly by a **class**:

— either when a **property_BSU** or a **data_type_BSU** referenced by this **class** references itself directly or indirectly by this **class_BSU**, or

— when a **class_BSU_relationship** referring to this **class** by its **relating_class** attribute refers to the **class_BSU** by its **related_tokens** attributes.

A **property_BSU** or a **data_type_BSU** refers to a **class_BSU** when:

— its **name_scope** is this **class_BSU**, or

— its definition **dictionary_element** is provided and refers, as its domain, to a **class_BSU** (*is-part-of* relationship), or

— its definition **dictionary_element** is provided and refers, as its domain, to another **data_type_BSU** that refers to a **class_BSU** (recursive definition);

Such references are checked by the **makes_reference_outside** function.

In the **ISO13584_extended_dictionary_schema**, a **class_BSU** makes an indirect forward reference to a **class_BSU** with respect to the class list order in five cases.

a) when its **described_by** attribute contains a **property_BSU** that makes a **makes_reference_outside**;

b) or when the class is defined by means of an **a_priori_semantic_relationship**, and its imported properties defined by the **referenced_properties** inherited attribute contain a **property_BSU** that makes a **makes_reference_outside**;

c) or when its **defined_types** attribute contains a **data_type_BSU** that makes a **makes_reference_outside**;

d) or when the class is defined by means of an **a_priori_semantic_relationship**, and its imported data types defined by the **referenced_data_types** inherited attribute contain a **data_type_BSU** that makes a **makes_reference_outside**;

e) or when its **associated_items** attribute contains a **class_BSU_relationship** whose **related_tokens** makes a **makes_reference_outside**;

   In the context of the **ISO13584_extended_dictionary_schema**, this situation may happen in two cases:

   1) the **class_BSU_relationship** is a **class_table_relationship** that contains a **table_BSU** that makes through its **name_scope** attribute a **makes_reference_outside**, or

   2) the **class_BSU_relationship** is a **class_document_relationship** that contains a **document_BSU** that makes through its **name_scope** attributes a **makes_reference_outside**

The **base_protocols** attribute enables the library data supplier to specify which external file protocols shall be supported by the user LMS to be able to process the library external files associated with the library delivery file. In a **dictionary** entity, the **external_file_protocol**s referenced by the **base_protocols** attribute may be any protocol. This entity may therefore be used for an exchange between a library data supplier and an end-user who agree on some proprietary protocols. The **dictionary_in_standard_format**, subtype of **dictionary**, only permits those **external_file_protocol**s that are defined in the referenced library integrated information model, and in the referenced view exchange protocols.

EXAMPLE 2 A library data supplier and an end-user may agree to use a private encoding for the content of the documents referenced in a dictionary. Such a **dictionary** cannot be exchanged as a **dictionary_in_standard_format**.

EXPRESS specification:

```
*)
ENTITY dictionary
SUPERTYPE OF(dictionary_in_standard_format);
      identified_by: OPTIONAL dictionary_identification;
      is_complete: OPTIONAL BOOLEAN;
      updates: OPTIONAL dictionary_identification;
      update_agreement: OPTIONAL identifier;
      referenced_dictionaries: SET [0:?] OF dictionary_identification;
      responsible_supplier: supplier_BSU;
      library_structure: library_iim_identification;
      base_protocols: SET [0:?] OF external_file_protocol;
      supported_vep: SET [0:?] OF
           view_exchange_protocol_identification;
      referred_suppliers: SET [1:?] OF supplier_BSU;
      contained_classes: LIST [0:?] OF UNIQUE class_BSU;
      a_posteriori_semantic_relationships: SET [0:?] OF
           a_posteriori_semantic_relationship;
      names: item_names;
      note: OPTIONAL note_type;
      remark: OPTIONAL remark_type;
WHERE
      WR1: prefix_ordered_class_list(SELF.contained_classes);
      WR2: (EXISTS(identified_by) AND EXISTS(is_complete))
           OR (NOT(EXISTS(identified_by)) AND NOT(EXISTS(is_complete)));
      WR3: NOT(EXISTS(identified_by)) OR
           (SELF.identified_by.defined_by = SELF.responsible_supplier);
      WR4: NOT(EXISTS(updates)) OR
           (EXISTS(identified_by) AND (is_complete = FALSE));
      WR5: NOT(EXISTS(update_agreement)) OR EXISTS(updates);
      WR6: NOT(EXISTS(updates)) OR
           ((updates.code = identified_by.code)
           AND (updates.defined_by = identified_by.defined_by)
           AND (updates.version <= identified_by.version)
           AND (NOT(updates.version = identified_by.version) OR
           (updates.revision < identified_by.revision)));
END_ENTITY; -- dictionary
(*
```

Attribute definitions:

identified_by: the **dictionary_identification**, if any, identifying the dictionary to which the classes defined in the **dictionary** entity belong.

is_complete: a Boolean attribute that specifies whether the **dictionary** entity describes all the classes belonging to the dictionary identified by the **identified_by dictionary_identification** entity.

NOTE 8 **is_complete** only exists when **identified_by** exists.

updates: the **dictionary_identification**, if any,of the dictionary that should be already available on the receiving system to be able to create the complete content of the **dictionary_identification** dictionary from the content of the **dictionary** entity.

NOTE 9 **updates** may only exist when **identified_by** exists and **is_complete** is equal to false.

update_agreement: the **identifier**, if any, that identifies the process to be used for creating the **identified_by** dictionary on the receiving system from the **updates** dictionary and the content of the **dictionary** entity.

NOTE 10 **update_agreement** may only exist when **updates** exists.

NOTE 11 Values to be used for this attribute may be either specified by private agreement between the sender and the receiver, or may be specified within a library integrated information model.

EXAMPLE 3 A particular value of the **update_agreement** attribute might mean that only those **dictionary_element**s of which some attributes are explicitly modified when moving from the **updates** dictionary to the **identified_by** dictionary shall be provided as part of the content of the **dictionary** entity.

NOTE 12 When **updates** exists and **update_agreement** does not exist, the process to be used to restore the **identified_by** dictionary is the one described in Annex O: all the **dictionary_element**s or the **content_item**s of the **basic_semantic_unit**s defined in the **identified_by** dictionary either are provided as part of the content of the **dictionary** entity, or they already belong to the **updates** dictionary.

referenced_dictionaries: the **dictionary_identification**s, if any, identifying the other dictionaries of which some classes are referenced in the **dictionary** entity.

NOTE 13 Neither the described dictionary, nor the referenced dictionaries need to be identified by **dictionary_identification** entities.

responsible_supplier: the library data supplier responsible for the **dictionary_element**s.

library_structure: the library integrated information model that the library delivery containing this dictionary realizes.

base_protocols: the set of referenced **external_file_protocol**s.

NOTE 14 Only the **external_file_protocol**s of which the support is required to successfully compile the dictionary belong to the **base_protocols** attribute. The possible **external_file_protocol**s referenced by a **document** content do not belong to the **base_protocols** attribute (see annex O).

supported_vep: the set of view exchange protocols supported by the dictionary.

referred_suppliers: the list of suppliers referred to or defined.

contained_classes: the list of the classes contained, whichever the **supplier_bsu** they reference.

a_posteriori_semantic_relationships: the list of **a_posteriori_semantic_relationship**s that are specified.

names: the names of the dictionary.

note: the note that describes the content of the dictionary.

remark: the remarks associated with the current delivery of the described dictionary.

Formal propositions:

WR1: the classes may be compiled without forward references in the order defined by the **contained_classes** list.

WR2: when the **dictionary** entity describes a dictionary identified by a **dictionary_identification** entity, it shall specify whether it describes whole or part of this dictionary.

WR3: when the **dictionary** entity describes a dictionary identified by a **dictionary_identification** entity, the supplier referenced by the **defined_by** attribute of the latter shall be the same as the supplier referenced by the **responsible_supplier** attribute of the former.

WR4: **updates** shall not exists when **identified_by** does not exist or **is_complete** equals TRUE.

WR5: **update_agreement** shall not exist when **updates** does not exist.

WR6: if both **identified_by** and **updates** exist, the **identified_by** dictionary shall have the same code and the same supplier as the **updates** dictionary, and a version/revision greater than the **updates** dictionary.

11.7 Dictionary_in_standard_format

A **dictionary_in_standard_format** entity is a **dictionary** that only references in its **base_protocols** attribute those external file protocols that are allowed either by the library integrated information model indicated by the **library_structure** attribute or the view exchange protocols referenced in the **supported_vep** attribute.

NOTE 1 Both **library_iim_identification**, and **view_exchange_protocol_identification** are meta data that describes respectively library integrated information models and a view exchange protocols. Standard values of these entities are specified within the standard documents that specify library integrated information models and view exchange protocols. Each one of these entities specifies which external file protocol(s) may be used for exchanging information between conformant implementations.

EXAMPLE By agreement between the sender and the receiver, a **dictionary** might reference a **program_protocol** that corresponds to CATIA® FORTRAN programs. Such a proprietary external file protocol is allowed neither by a library integrated information model, nor by an ISO 13584 view exchange protocol. Thus it shall not be referenced by a **dictionary_in_standard_format** entity, but by a **dictionary** entity.

Only a user LMS supporting the same conformance classes as the ones referenced in the **library_structure** and in the **supported_vep** attribute of a **dictionary** shall be able to compile this **dictionary**.

NOTE 2 Both library integrated information models and view exchange protocols define various sets of options that may be selected by a conformant implementation. These sets of options are termed conformance classes.

EXPRESS specification:

```
*)
ENTITY dictionary_in_standard_format
SUBTYPE OF(dictionary);
WHERE
     WR1: QUERY(int <* SELF\dictionary.base_protocols
          | ((SIZEOF(QUERY(vep <* SELF\dictionary.supported_vep
          | int IN vep\data_exchange_specification_identification
          .external_file_protocols)) = 0) AND NOT(int IN
          SELF\dictionary.library_structure
```



```
          .external_file_protocols))) = [];
END_ENTITY; -- dictionary_in_standard_format
(*
```

Formal propositions:

WR1: the set of **external_file_protocol**s in the **base_protocols** attribute shall be a subset of the union of the **external_file_protocol**s referenced in the **library_structure** attribute and the **external_file_protocol**s referenced in the members of the **supported_vep** attribute.

11.8 Data_exchange_specification_identification

A **data_exchange_specification_identification** identifies a data specification published in one part of ISO 13584 that contributes to the specification of a library exchange context. A **data_exchange_specification_identification** may be either a **library_iim_identification**, or a **view_exchange_protocol_identification**.

NOTE The specification of a library exchange context consists of:

a) a library integrated information model,

b) 0 to n view exchange protocol(s), and

c) an implementation method.

EXPRESS specification:

```
*)
ENTITY data_exchange_specification_identification
ABSTRACT SUPERTYPE OF(ONEOF(library_iim_identification,
     view_exchange_protocol_identification));
     source_document_identifier: OPTIONAL identifier;
     status: label;
     name: identifier;
     date: year_number;
     application: OPTIONAL identifier;
     level: OPTIONAL identifier;
     external_file_protocols: SET [0:?] OF external_file_protocol;
END_ENTITY; -- data_exchange_specification_identification
(*
```

Attribute definitions:

source_document_identifier: the identifier of the document that contains the data specification. For those documents issued by ISO TC184/SC4/WG2 this identifier shall be the integer part of the N number.

status: a classification of the data specification with respect to its acceptance by the approving body of this International Standard, possibly followed by an integer version. It may take the values: 'WD', 'CD', 'DIS', 'FDIS', 'IS', 'TS', 'PAS', 'ITA'.

name: the **identifier** of the data specification as defined in the corresponding part of ISO 13584.

date: the year when the corresponding part of ISO 13584 reached its status.

application: an **identifier** possibly defined in the corresponding part of ISO 13584 to characterise an allowed functional subset of the complete data specification.

level: an **identifier** possibly defined in the corresponding part of ISO 13584 that further characterises an allowed subset of the application subset.

external_file_protocols: the list of **external_file_protocol**s of which the use is allowed by the view exchange protocol for the **application** functional subset.

11.9 Library_iim_identification

A **library_iim_identification** identifies a library integrated information model that includes the definition of a library delivery file information model.

NOTE This part of ISO 13584 includes three library integrated information models, which are defined in clauses 16, 17 and 18.

EXAMPLE The library integrated information model LIIM 24-1 includes the **ISO13584_g_m_iim_schema** that is the information model of a library that consists only of general model classes. It also includes the **ISO13584_g_m_iim_conformance_schema** documented in annex E.

EXPRESS specification:

```
*)
ENTITY library_iim_identification
SUBTYPE OF(data_exchange_specification_identification);
END_ENTITY; -- library_iim_identification
(*
```

11.10 View_exchange_protocol_identification

A **view_exchange_protocol_identification** identifies a data specification defined in a part of the view exchange protocol series of parts of ISO 13584. References to such view exchange protocols define which library external files are used within a library exchange context, which dictionary entries shall be recognized by a receiving system that claims conformance to this view exchange protocol, and which additional constraints are fulfilled by a library delivery file.

NOTE 1 A view exchange protocol may specify which standard instance of a **view_exchange_protocol_identification** entity data type is to be used to reference it. This is done by means of an EXPRESS schema that consists only of constraints. These constraints are fulfilled by any library delivery file that references this view exchange protocol in any of its conformance class. This means that only this standard instance may be used to reference this view exchange protocol.

NOTE 2 The various kinds of library external files that are allowed for use by a view exchange protocol are those external files protocols that are referenced by the **external_file_protocols** inherited attribute of the **view_exchange_protocol_identification** entity instance. Reference to such an instance by a **dictionary** entity adds these **external_file_protocol**s to those that were allowed by the library integrated model and by the other view exchange protocols.

When a **view_exchange_protocol_identification** references an **application_protocol_definition**, this means that the corresponding view exchange protocol supports the use of library external files conformant with one Application Protocol of ISO 10303.

EXPRESS specification:

```
*)
ENTITY view_exchange_protocol_identification
```



```
SUBTYPE OF(data_exchange_specification_identification);
     referenced_ISO10303_AP: OPTIONAL application_protocol_definition;
END_ENTITY; -- view_exchange_protocol_identification
(*
```

Attribute definitions:

referenced_ISO10303_AP: the ISO 10303 Application Protocol to which library external files shall conform. The **referenced_ISO10303_AP** need not to be specified for a particular **view_exchange_protocol_id**.

11.11 ISO13584_extended_dictionary_schema entity definitions: additional entity instance types

This clause specifies some subtypes of **entity_instance_type** that are needed for the extended dictionary schema.

NOTE 1 **Entity_instance_type** is defined in the ISO/IEC common dictionary schema.

NOTE 2 Instances of these subtypes of **entity_instance_type** might also be represented as instances of **entity_instance_type**. These subtypes are defined to enable an EXPRESS schema to use explicitlly some or all of these subtypes without using the **entity_instance_type** supertype.

11.11.1 Representation_type

A **representation_type** is the type of an instance of an ISO 10303-43 **representation**. View exchange protocols shall specify the use of this resource.

NOTE According to ISO 10303-42, an instance of **placement** may only exist if it is related to a **geometric_representation_context** in some **representation**. Therefore, if some class properties are **placement**s, this class shall contain a property whose data type is a **geometric_representation_context_type** (that defines the context of these **placement**s) and a property whose data type is a **representation_type** (that gathers these **placement**s with their context).

EXPRESS specification:

```
*)
ENTITY representation_type
     SUBTYPE OF(entity_instance_type);
WHERE
     WR1:'REPRESENTATION_SCHEMA.REPRESENTATION'
          IN SELF\entity_instance_type.type_name;
END_ENTITY; -- representation_type
(*
```

Formal propositions:

WR1: the string 'REPRESENTATION_SCHEMA.REPRESENTATION' shall be contained in the attribute **SELF\entity_instance_type.type_name** that is a SET of STRINGs.

11.11.2 Geometric_representation_context_type

A **geometric_representation_context_type** is the type of an instance of an ISO 10303-42 **geometric_representation_context**. View exchange protocols shall specify the use of this resource.

EXPRESS specification:

```
*)
ENTITY geometric_representation_context_type
     SUBTYPE OF(entity_instance_type);
WHERE
     WR1:'GEOMETRY_SCHEMA.GEOMETRIC_REPRESENTATION_CONTEXT'
          IN SELF\entity_instance_type.type_name;
END_ENTITY; -- geometric_representation_context_type
(*
```

Formal propositions:

WR1: the string 'GEOMETRY_SCHEMA.REPRESENTATION_CONTEXT' shall be contained in the attribute **SELF\entity_instance_type.type_name** that is a SET of STRINGs.

11.11.3 Representation_reference_type

A **representation_reference_type** is the type of an instance of **representation_reference** (see clause 13.7.2).

EXAMPLE An instance of **representation_reference** may be used in a library exchange context to refer to a library external file that contains an instance of an ISO 10303-43 **representation**. Such a value might be the one of a functional model property that describes the content of some functional view the functional model may generate.

NOTE View exchange protocols shall specify the use of this resource.

EXPRESS specification:

```
*)
ENTITY representation_reference_type
     SUBTYPE OF(entity_instance_type);
WHERE
     WR1:'ISO13584_EXTERNAL_FILE_SCHEMA.REPRESENTATION_REFERENCE'
          IN SELF\entity_instance_type.type_name;
END_ENTITY; -- representation_reference_type
(*
```

Formal propositions:

WR1: the string 'ISO13584_EXTERNAL_FILE_SCHEMA.REPRESENTATION_REFERENCE' shall be contained in the attribute **SELF\entity_instance_type.type_name** that is a SET of STRINGs.

11.11.4 Program_reference_type

A **program_reference_type** is the type of an external reference, from a library exchange context, to an external file that contains a program. For example, this external file may be an ISO 13584-31 conforming program that may generate, according to the values of some parameters, different instances of ISO 10303 representation data types. Such an entity instance may be used, in a library exchange context, to describe the content of a method associated with a functional model class.

NOTE View exchange protocols specify the use of this resource.

EXPRESS specification:

```
*)
ENTITY program_reference_type
     SUBTYPE OF(entity_instance_type);
WHERE
     WR1: 'ISO13584_EXTERNAL_FILE_SCHEMA.PROGRAM_REFERENCE'
          IN SELF\entity_instance_type.type_name;
END_ENTITY; -- program_reference_type
(*
```

Formal propositions:

WR1: the string 'ISO13584_EXTERNAL_FILE_SCHEMA.PROGRAM_REFERENCE' must be contained in the attribute **SELF\entity_instance_type.type_name** that is a SET of STRINGs.

11.12 ISO13584_extended_dictionary_schema entity definitions: additional basic semantic units

This clause introduces the different basic semantic units that are used to represent the supplier related elements and class related elements that are specific to the **ISO13584_extended_dictionary_schema**. They are obtained by subtyping the **supplier_related_BSU** and **class_related_BSU** described in the **ISO13584_IEC61360_dictionary_schema**.

11.12.1 Program_library_BSU

A **program_library_BSU** is a basic semantic unit associated with a program library.

EXPRESS specification:

```
*)
ENTITY program_library_BSU
SUBTYPE OF(supplier_related_BSU);
     defined_by: supplier_BSU;
     SELF\basic_semantic_unit.code: program_library_code_type;
DERIVE
     absolute_id: identifier := defined_by.dic_identifier
          + sep_id + dic_identifier;
INVERSE
     associated_to_supplier: supplier_BSU_relationship
          FOR related_tokens;
UNIQUE
     UR1: absolute_id;
WHERE
     WR1: defined_by :=: associated_to_supplier.
          relating_supplier\dictionary_element.identified_by;
END_ENTITY; -- program_library_BSU
(*
```

Attribute definitions:

defined_by: the supplier who defined the referenced program library.

absolute_id: the **identifier** associated with the referenced program library.

associated_to_supplier: the **supplier_BSU_relationship** referring to this **program_library_BSU**.

Formal propositions:

UR1: the **absolute_id** attribute is unique.

WR1: through the **supplier_BSU_relationship**, the referenced program library is associated with the same supplier as the supplier who defined it.

Informal propositions:

IP1: when the version of a **program_library_BSU** is increased, the new program library shall include one, possibly new, release of each program contained in the previous program library version.

When the **version** is incremented, the **revision** of the corresponding **dictionary_element** and the **content_revision** of the possible **content_item** shall both be reset to '000'.

NOTE This informal proposition ensures the feasibility of upward compatible evolution of program libraries: any program that makes references to one program library in some version may be run (with a difference considered by the supplier as acceptable) using any new version.

11.12.2 Table_BSU

A **table_BSU** entity is a **basic_semantic_unit** that constitutes the **table_identification** of a table referenced in a ISO 13584-conformant dictionary.

The following gives an overview of how updating operations have effects on the version number (**V**) of a **table_BSU,** on the **table_element** revision number (**DR**) or on the **table_content** revision of content number (**CR**) concept for the given class, or are forbidden at all (**X**).— Change of key columns[1]: **X**

— Suppression of non-key columns: **X**

— Addition of non-key columns: **V**

— Change of number of lines: **V**

— Other changes in value content[2]: **V**

— Change of names, definition, note, and remark: **DR**

— Change of version of referenced property[3]: **DR/V**

— Correction of value errors or imprecision **CR**

[1] Other than change of version of the referenced properties

[2] Other than typo correction or value accuracy correction

[3] **DR** if values are unchanged, **V** if values are changed

Moreover, a table contributes to the definition of the class where it is applicable. Thus, a change of version of a table implies a change of version of the classes where it is applicable.

EXPRESS specification:

```
*)
ENTITY table_BSU
SUBTYPE OF(class_related_BSU, table_identification);
      name_scope: class_BSU;
      SELF\basic_semantic_unit.code: table_code_type;
DERIVE
      absolute_id: identifier:= name_scope.
           defined_by.dic_identifier + sep_id
           + name_scope.dic_identifier + sep_id
           + dic_identifier;
UNIQUE
      UR1: absolute_id;
END_ENTITY; -- table_BSU
(*
```

Attribute definitions:

name_scope: the root class where the referenced table is visible.

absolute_id: the **identifier** associated with the referenced table.

Formal propositions:

UR1: the **absolute_id** attribute is unique.

Informal propositions:

IP1: when the version of a **table_BSU** is increased, the new table shall:

— have the same **property_BSU**s as a key, and

— contain, as its first **column**s, **column**s referring to the same **property_BSU**s as the previous table version. It may contain additional **column**s and/or different values in the column.

NOTE This informal proposition ensures the feasibility of upward compatible evolution of tables: any query or program that makes reference to one table may be run (with a difference considered by the supplier as acceptable) using any new version.

IP2: when the **version** is incremented, the **revision** of the corresponding **dictionary_element** and the **revision_of_content** of the possible **table_content** shall both be reset to '000'.

11.12.3 Document_BSU

A **document_BSU** is a basic semantic unit associated with a document.

EXPRESS specification:

```
*)
ENTITY document_BSU
SUBTYPE OF(class_related_BSU);
      name_scope: class_BSU;
      SELF\basic_semantic_unit.code: document_code_type;
DERIVE
      absolute_id: identifier :=
           name_scope.defined_by.dic_identifier + sep_id
           + name_scope.dic_identifier + sep_id
           + dic_identifier;
INVERSE
      associate_to_class: class_BSU_relationship FOR related_tokens;
UNIQUE
      UR1: absolute_id;
END_ENTITY; -- document_BSU
(*
```

Attribute definitions:

name_scope: the root class where the referenced document is visible.

absolute_id: the **identifier** associated with the referenced document.

associate_to_class: the **class_BSU_relationship** referring to this **document_BSU**.

Formal propositions:

UR1: the **absolute_id** attribute is unique.

Informal propositions:

IP1: when the version of a **document_BSU** is increased, the new document shall have the same scope as the previous one but may have a different content or use a different **protocol**.

When the **version** is incremented, the **revision** of the corresponding **dictionary_element** and the **content_revision** of the possible **content_item** shall both be reset to '000'.

NOTE This informal proposition ensures the feasibility of upward compatible evolution of document usage. When, in any program, a request to display some document takes place. Displaying a new version of the document provides the same kind of information to the user.

11.13 ISO13584_extended_dictionary_schema entity definitions: supplier BSU relationship

11.13.1 Supplier_program_library_relationship

A **supplier_program_library_relationship** is a link between program libraries and the supplier that provides them. This link is modelled through the entity relationship paradigm.

EXPRESS specification:

```
*)
ENTITY supplier_program_library_relationship
SUBTYPE OF(supplier_BSU_relationship);
     SELF\supplier_BSU_relationship.related_tokens:
          SET [1:?] OF program_library_BSU;
END_ENTITY; -- supplier_program_library_relationship
(*
```

Attribute definitions:

related_tokens: a set of pogram library units that are provided by the supplier described.

11.14 ISO13584_extended_dictionary_schema entity definitions: class BSU relationships

This clause introduces the different relationships between a class and class related elements.

11.14.1 Class_table_relationship

A **class_table_relationship** is a link between tables and the root class(es) where they are applicable.

EXPRESS specification:

```
*)
ENTITY class_table_relationship
SUBTYPE OF(class_BSU_relationship);
     SELF\class_BSU_relationship.related_tokens:
          SET [1:?] OF table_BSU;
WHERE
     WR1: applicable_properties_for_applicable_tables(SELF);
     WR2: visible_tables(SELF\class_BSU_relationship.relating_class.
          identified_by, SELF.related_tokens);
END_ENTITY; -- class_table_relationship
(*
```

Attribute definitions:

related_tokens: the set of tables related to the class.

Formal propositions:

WR1: each table associated with a class shall contain columns that refer to applicable properties for this class.

WR2: all the tables associated with a class shall be visible from the class, i.e., the tree whose root is their **name_scope** shall contain the class.

11.14.2 Class_document_relationship

A **class_document_relationship** is a link between documents and the root class(es) where they are applicable.

EXPRESS specification:

```
*)
ENTITY class_document_relationship
SUBTYPE OF(class_BSU_relationship);
     SELF\class_BSU_relationship.related_tokens:
          SET [1:?] OF document_BSU;
WHERE
     WR1: visible_documents(
          SELF\class_BSU_relationship.relating_class.identified_by,
          SELF.related_tokens);
END_ENTITY; -- class_document_relationship
(*
```

Attribute definitions:

related_tokens: the set of **document_BSU** that are related to the class.

Formal propositions:

WR1: the documents associated with a class shall be visible from the class, i.e., the tree whose root is their **name_scope** shall contain the class.

11.15 ISO13584_extended_dictionary_schema entity definitions: properties of functional models and functional views

The ISO/IEC common dictionary schema makes provision for three kinds of properties that may be used to describe the library items represented by general model classes:

— context parameters (represented as a **condition_DET property_DET**),

— item characteristics (represented as a **non_dependent_P_DET property_DET**),

— context-dependent characteristics (represented as a **dependent_P_DET property_DET**).

This clause provides the resource to represent model-defined or view-defined properties.

11.15.1 Representation_P_DET

A **representation_P_DET** entity represents the properties that are defined in classes of functional models and classes of functional views.

EXAMPLE This subtype of **property_DET** shall be used to represent the properties that characterise the insertion context of the representation of a library item, the internal variables used in a method, and the properties that constitute a representation of a library item.

EXPRESS specification:

```
*)
ENTITY representation_P_DET
SUBTYPE OF(property_DET);
WHERE
     WR1: NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA.CONDITION_DET'
```



```
            IN TYPEOF(SELF)) AND
            NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA.DEPENDENT_P_DET'
            IN TYPEOF(SELF)) AND
            NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
            '.NON_DEPENDENT_P_DET' IN TYPEOF(SELF));
END_ENTITY; -- representation_P_DET
(*
```

Formal propositions:

WR1: a **representation_P_DET** cannot be a **non_dependent_P_DET**, a **dependent_P_DET**, or a **condition_DET**.

11.16 ISO13584_extended_dictionary_schema entity definitions: specific dictionary elements

This clause defines the dictionary elements specific to ISO 13584. They are divided into two categories: one for suppliers and another for classes.

11.16.1 Supplier_related_dictionary_element

A **supplier_related_dictionary_element** is a **dictionary_element** that contains information that may be used to process any library provided by a supplier.

EXPRESS specification:

```
*)
ENTITY supplier_related_dictionary_element
ABSTRACT SUPERTYPE OF(program_library_element)
SUBTYPE OF(dictionary_element);
     SELF\dictionary_element.identified_by: supplier_related_BSU;
     names: item_names;
     definition: definition_type;
     note: OPTIONAL note_type;
     remark: OPTIONAL remark_type;
END_ENTITY; -- supplier_related_dictionary_element
(*
```

Attribute definitions:

identified_by: the basic semantic unit that identifies the **dictionary_element**.

names: the names of the **dictionary_element** described.

definition: the definition of the **dictionary_element** described.

note: the notes associated with the **dictionary_element** described.

remark: the remark associated with the **dictionary_element** described.

11.16.2 Class_related_dictionary_element

A **class_related_dictionary_element** is a **dictionary_element** that contributes to the definition of a class.

EXPRESS specification:

```
*)
ENTITY class_related_dictionary_element
ABSTRACT SUPERTYPE OF(ONEOF(table_element, document_element))
SUBTYPE OF(dictionary_element);
     SELF\dictionary_element.identified_by: class_related_BSU;
     names: item_names;
     definition: definition_type;
     note: OPTIONAL note_type;
     remark: OPTIONAL remark_type;
END_ENTITY; -- class_related_dictionary_element
(*
```

Attribute definitions:

identified_by: the basic semantic unit that identifies the **dictionary_element**.

names: the names of the **dictionary_element** described.

definition: the definition of the **dictionary_element** described.

note: the notes associated with the **dictionary_element** described.

remark: the remarks associated with the **dictionary_element** described.

11.16.3 Program_library_element

A **program_library_element** is a description of a program library.

EXPRESS specification:

```
*)
ENTITY program_library_element
SUBTYPE OF(supplier_related_dictionary_element);
     SELF\dictionary_element.identified_by: program_library_BSU;
END_ENTITY; -- program_library_element
(*
```

Attribute definitions:

identified_by: the basic semantic unit that identifies the program library described.

11.17 ISO13584_extended_dictionary_schema entity definitions: class related elements

These clauses introduce the class related elements specific to ISO 13584.

11.17.1 Table_element

A **table_element** is the **table_specification** of a table associated with a class in an ISO 13584 conformant dictionary. The **variable_semantics** that define the meaning of its **columns** shall be

self_property_semantics that refer to properties that are either visible for the class that constitutes the **name_scope** of the table or imported by this class or any of its superclass.

NOTE 1 If a **property_semantics**, defined in functional view classes, needs to be referenced as their meaning by **column**s of a **table_element** associated with a functional model class in an ISO 13584 conformant dictionary, the functional model class can import the **property_semantics** defined in functional view classes through the **imported_properties_from_view** attribute of the **abstract_functional_model_class** entity.

EXPRESS specification:

```
*)
ENTITY table_element
SUBTYPE OF(class_related_dictionary_element, table_specification);
     SELF\dictionary_element.identified_by: table_BSU;
     SELF\table_specification.column_meaning:
          LIST [1:?] OF UNIQUE self_property_semantics;
     SELF\table_specification.key:
          SET [1:?] OF self_property_semantics;
DERIVE
     SELF\table_specification.table_identifier: table_BSU
          := SELF.identified_by;
WHERE
     WR1: QUERY (temp <* SELF.column_meaning
          | NOT visible_properties(
          SELF\dictionary_element.identified_by.name_scope,
          get_property_BSU_from_property_semantics([temp]))
          AND NOT applicable_properties(
          SELF\dictionary_element.identified_by.name_scope,
          list_to_set(get_property_BSU_from_property_semantics(
          [temp])))) = [];
END_ENTITY; -- table_element
(*
```

Attribute definitions:

identified_by: the basic semantic unit that identifies the table.

column_meaning: the list of **self_property_semantics** that represents the meaning of the different **column**s contained in the table.

key: the set of **property_semantics** that are the key of the table described.

table_identifier: the **table_identification** that is the **table_BSU** that identifies the table.

Formal propositions:

WR1: the **variable_semantics** that define the meaning of the **columns** of the table shall be **self_property_semantics** that refer to properties that are visible for the class that constitutes the **name_scope** of the table or imported by this class, or any of its superclasses.

Informal propositions:

IP1: the different revision of the **dictionary_element** that describes the same version of a **table_BSU** may only differ by their informal attributes: names, definition, note and remark.

NOTE 2 This informal proposition states that when updating a dictionary an old revision may be replaced by a new one without any consequence on the functional behaviour of the integrated library.

11.17.2 RDB_table_element

A **RDB_table_element** is a **table_element** that contains columns of simple types.

EXPRESS specification:

```
*)
ENTITY RDB_table_element
SUBTYPE OF(table_element, RDB_table_specification);
WHERE
     WR1: QUERY(temp <* SELF.column_meaning
          | simple_type_data_type(temp\property_semantics.the_property)
          = FALSE) = [];
END_ENTITY; -- RDB_table_element
(*
```

Formal propositions:

WR1: all the types of the properties in the **column_meaning** list shall be simple types.

11.17.3 Document_element

A **document_element** is a specification of a document.

EXPRESS specification:

```
*)
ENTITY document_element
SUPERTYPE OF(ONEOF(
     document_element_with_http_access,
     document_element_with_translated_http_access))
SUBTYPE OF(class_related_dictionary_element);
     SELF\dictionary_element.identified_by: document_BSU;
     authors: OPTIONAL LIST [1:?] OF person;
     publishing_organisation: organization;
END_ENTITY; -- document_element
(*
```

Attribute definitions:

identified_by: the basic semantic unit that identifies the document.

authors: the author(s) of the document.

publishing_organisation: the organisation that publishes the document.

Informal propositions:

IP1: the different revision of the **dictionary_element** that describes the same version of a **document_BSU** may only differ by their informal attributes: names, definition, note and remark.

NOTE This informal proposition states that when updating a dictionary an old revision may be replaced by a new one without any consequence on the functional behaviour of the integrated library.

11.17.4 Document_element_with_http_access

A **document_element_with_http_access** is a document whose content may be accessed using the http Internet transfer protocol at some absolute URL. The document is in the language defined by the unique **global_language_assignment** associated with the dictionary.

NOTE 1 The **global_language_assignment** entity is defined in the **ISO13584_IEC61360_language_resource_schema** documented in ISO 13584-42: 1998.

NOTE 2 The http protocol is specified in IAB RFC 2068.

NOTE 3 Parties to agreements based on this part of ISO 13584 are encouraged to investigate the possibility of applying the most recent IAB RFC belonging to the Standards Track RFC and that updates IAB RFC 2068.

EXPRESS specification:

```
*)
ENTITY document_element_with_http_access
SUBTYPE OF(document_element);
      remote_location: absolute_URL_type;
END_ENTITY; -- document_element_with_http_access
(*
```

Attribute definitions:

remote_location: the absolute URL that specifies the document locator.

Informal propositions:

IP1: the document format shall correspond to a standard MIME format registered by IAB.

IP2: the language of the document shall correspond to the language defined by the unique **global_language_assignment**.

11.17.5 Document_element_with_translated_http_access

A **document_element_with_translated_http_access** is a document whose content may be accessed in different languages using the http Internet transfer protocol at several absolute URL. The translation languages are defined by a **present_translations** entity, and an URL is given for each corresponding language.

NOTE 1 The **present_translations** entity is defined in the **ISO13584_IEC61360_language_resource_schema** documented in ISO 13584-42.

NOTE 2 The http protocol is specified in IAB RFC 2068.

NOTE 3 Parties to agreements based on this part of ISO 13584 are encouraged to investigate the possibility of applying the most recent IAB RFC belonging to the Standards Track RFC and that updates IAB RFC 2068.

EXPRESS specification:

```
*)
ENTITY document_element_with_translated_http_access
SUBTYPE OF(document_element);
      remote_locations: LIST [1:?] OF absolute_URL_type;
      languages: present_translations;
WHERE
      WR1: SIZEOF(remote_locations) = SIZEOF(languages.language_codes);
END_ENTITY; -- document_element_with_translated_http_access
(*
```

Attribute definitions:

remote_locations: the list of absolute URLs that specify the locators where the document may be found in the different languages specified by the **languages** attribute.

languages: the list of languages in which the document is translated.

Formal propositions:

WR1: the number of URL contained in the **remote_locations** list shall be equal to the number of **languages** defined in the **languages.language_codes** attribute.

Informal propositions:

IP1: the possibly different document formats shall correspond to standard MIME registered by the IAB.

IP2: the document located at the **remote_locations**[i] URL shall be in the language identified by **languages.language_codes**[i].

11.17.6 Referenced_document

A **referenced_document** is a specialisation of a **document** defined in the ISO/IEC dictionary schema that enables to reference a document that is identified by a **document_BSU**.

EXPRESS specification:

```
*)
ENTITY referenced_document
SUBTYPE OF(document);
      document_reference: document_BSU;
WHERE
      WR1: NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
           '.IDENTIFIED_DOCUMENT' IN TYPEOF(SELF));
END_ENTITY; -- referenced_document
(*
```


Attribute definitions:

document_reference: the basic semantic unit associated with the document.

Formal propositions:

WR1: a **referenced_document** shall not be an **identified_document**.

11.17.7 Referenced_graphics

A **referenced_graphics** is a specialisation of a **graphics** defined in the ISO/IEC dictionary schema that enables to reference a document that is identified by a **document_BSU** and which contains the graphic image.

NOTE A **referenced_graphics** is a specialisation of a **graphics** entity defined in ISO 13584-42. Another specialisation of a **graphics** entity is an **illustration** defined in the **ISO13584_external_file_schema**. The **illustration_is_not_a_referenced_graphics_rule** ensures that both subtypes are incompatible. This rule is defined in the **ISO13584_external_file_schema**

EXPRESS specification:

```
*)
ENTITY referenced_graphics
SUBTYPE OF(graphics);
     graphics_reference: document_BSU;
END_ENTITY; -- referenced_graphics
(*
```

Attribute definitions:

graphics_reference: the basic semantic unit associated with the document that provides the graphic image.

11.18 ISO13584_extended_dictionary_schema entity definitions: feature class

A **feature_class** captures the dictionary description of items that represent one aspect of another item and that are themselves associated with properties.

EXAMPLE Such an aspect may be a form feature or an outlet of an electronic component.

A property whose data type is defined by a **feature_class** captures that some aspect of an item is defined by an instance of the **feature_class**.

EXPRESS specification:

```
*)
ENTITY feature_class
SUBTYPE OF(item_class);
WHERE
     WR1: NOT(('ISO13584_IEC61360_DICTIONARY_SCHEMA.'
          + 'COMPONENT_CLASS') IN TYPEOF(SELF));
     WR2: NOT(('ISO13584_IEC61360_DICTIONARY_SCHEMA.'
          + 'MATERIAL_CLASS') IN TYPEOF(SELF));
END_ENTITY; -- feature_class
```



```
(*
```

Formal propositions:

WR1: a **feature_class** shall not be a **component_class**.

WR2: a **feature_class** shall not be a **material_class**.

11.19 ISO13584_extended_dictionary_schema entity definitions: a priori semantic relationship

An **a_priori_semantic_relationship** is a kind of class that is defined on the basis of other classes, and that can import properties, data types, tables and documents contained in these classes. The properties, data types, tables or documents whose definitions are imported through the **a_priori_semantic_relationship** entity became applicable to the class that imports them, but not visible for this class. This means that they are not considered as visible by the **visible_properties** function.

NOTE 1 The inheritance relationship is a well known example of semantic relationship between classes modelled according to the object oriented paradigm. All the properties and other features defined in a class usually apply implicitly to all its subclasses. This relationship is used in ISO 13584 where all the properties, data types, tables or documents visible (respectively applicable) to some classes are implicitly visible (respectively applicable) to all its subclasses. As usual, in ISO 13584 this inheritance is implicit (i.e., not declared by means of an **a_priori_semantic_relationship**) and global (i.e., all the properties and data types are inherited by all its subclasses). An **a_priori_semantic_relationship** enables to define other semantic relationships that are particular to the parts library application domain.

NOTE 2 In ISO 13584, besides the usual inheritance (is-a) and aggregation (is-part-of) relationships, two additional semantic relationship are defined: is-view-of and is-case-of (see clause 3). These two relationships are captured by means of the **a_priori_semantic_relationship** entity.

NOTE 3 The properties, data types, tables or documents whose definitions are imported through the **a_priori_semantic_relationship** entity are explicitly defined. They may consist of a subset of all the properties, data types, tables or documents defined (i.e., visible or applicable) for the **referenced_classes** class(es).

NOTE 4 The properties, data types, tables or documents whose definitions are imported through the **a_priori_semantic_relationship** entity will be visible or already imported through another **a_priori_semantic_relationship** for the **referenced_classes** class(es). This is asserted by a global rule of the **ISO13584_extended_dictionary_schema**.

EXPRESS specification:

```
*)
ENTITY a_priori_semantic_relationship
ABSTRACT SUPERTYPE OF(ONEOF(
      item_class_case_of,
      fm_class_view_of,
      functional_model_class))
SUBTYPE OF(class);
      referenced_classes: SET [1:?] OF class_BSU;
      referenced_properties: LIST [0:?] OF property_BSU;
      referenced_data_types: SET [0:?] OF data_type_BSU;
      referenced_tables: SET [0:?] OF table_BSU;
      referenced_documents: SET [0:?] OF document_BSU;
END_ENTITY; -- a_priori_semantic_relationship
(*
```


Attribute definitions:

referenced_classes: the class(es) from where the properties, data types, tables or documents are imported.

NOTE 5 The class from which properties, data types, tables or documents are imported cannot be deduced from the identification of the imported properties, data types, tables or documents because they may be imported from a class where they are inherited. For instance, in IEC 61360-4, "input-voltage" is a property visible at the root level of the IEC classification. If a supplier class imports the "input-voltage" property from the IEC "transistor" class, this means that (1) the supplier class defines a transistor, and (2) these transistors are described by means of an "input voltage" property.

referenced_properties: the properties whose definitions are imported through the **a_priori_semantic_relationship** entity.

NOTE 6 The list order defines the default order for displaying imported properties during user access to the various subtypes of **a_priori_semantic_relationship**.

referenced_data_types: the data types whose definitions are imported through the **a_priori_semantic_relationship** entity.

referenced_tables: the tables whose definitions are imported through the **a_priori_semantic_relationship** entity.

referenced_documents: the documents whose definitions are imported through the **a_priori_semantic_relationship** entity.

11.20 ISO13584_extended_dictionary_schema entity definitions: functional model class

An **abstract_functional_model_class** entity, is the supertype of the various kinds of functional model classes. A functional model classes describe a set of representations that may be associated with a set of items.

NOTE 1 The items to which representations may be associated need not to be specified in a functional model class.

EXAMPLE 1 Prices and simulation models are examples of representations that are usually associated with a set of items.

EXAMPLE 2 A schematic representation of a screw is a representation that may defined independently of any screw.

The representation category provided by a functional model class is specified by the functional view that is referenced, and, in the case of an instantiable functional view class, that may be created by the functional model class.

EXAMPLE 3 Geometry view and schematic view are categories of representation that may be created in product data by a functional model class, for instance through a geometric application programming interface.

The levels of representation provided are specified by the **view_control_variable_range**.

EXAMPLE 4 The level of representation of a geometry view could be the detail-level of the view: simplified to extended.

NOTE 2 The standard functional views that may be created by an ISO 13584 conforming library are specified in the view exchange protocol series of parts of ISO 13584. Non standard functional views may also be defined by private agreement between the sender and the receiver. The definitions of such non standard functional view classes may be exchanged using the library integrated information model LIIM 24-3 defined in clause 18 of this part of ISO 13584.

The distinction established for general model classes between parts characteristics, context parameters and context-dependent characteristics are no longer significant for functional model classes. The shape representation of the end of a threaded hole, for example, may be considered either as an intrinsic characteristics, or as a context-dependent characteristics, depending upon the level of detail required of the representation. Therefore, all the properties defined within a functional model class, whether they are intended to play a role of a context parameter, a characteristics of the representation or a variable internal to the methods, shall be defined as **representation_P_DET**s. These properties are specified by the **SELF\class.described_by** inherited attribute.

An abstract functional model class may be associated with an item class in two different ways.

a) The first way, captured as a **functional_model_class** entity, is to describe the functional model class without referring to any item class. The association between this functional model class and an item class is possibly done afterwards, either by the supplier or by the end-user. It is called the a posteriori schema. In this approach, the properties (defined by the **described_by** inherited attribute), that are intended to match properties defined in the item classes are defined as **representation_P_DET**s. According to their role with respect to some item class, they will be matched with **condition_DET**, **non_dependent_P_DET**, **dependent_P_DET** in the target item classes.

EXAMPLE 5 The geometry of an H-screw may be described without referring explicitly to any specific component class. The same geometry applies to some ISO standard component classes, to some DIN standard component classes, or to various supplier component classes. The relationship between such a functional model class and one specific component class may be specified outside the functional model class either by the supplier (see: **a_posteriori_view_of** entity) or by the end-user. In both cases, when such a relationship between two classes is specified, the mapping from the component properties onto the representation properties shall be specified. For example, one property of this functional model class defined as **representation_P_DET** would correspond to the threaded diameter. It would be matched with a **non_dependent_P_DET** of the component class.

b) The second way, captured as a **fm_class_view_of** entity, is to directly define the functional model class by reference to the properties defined in some item class. It is called the a priori schema. In this case the properties that describe the functional model class consist of both the properties defined by the **described_by** inherited attribute and the properties imported from the item class (**imported_properties_from_item**).

EXAMPLE 6 Consider the functional model of supplying. It describes the price, quantity of order and so on, of some well defined component class. It does not apply to any other component class, and it is useless without this component class. When describing this functional model class, the library data supplier directly refers to the component class the functional model class is-view-of. This allows the direct reference (through their **property_BSU**s) to the properties defined within this component class.

A functional model class is always involved in an **a_priori_semantic_relationship** with the functional view class that specifies the user perspective addressed by the functional model class, and, in the case of an instanciable functional view class, the created representations. In the a priori schema, it is also involved in an **a_priori_semantic_relationship** with the item class to which it refers.

EXAMPLE 7 Procurement and design are user perspectives that may be associated to a functional view class.

11.20.1 Abstract_functional_model_class

An **abstract_functional_model_class** entity captures the dictionary definition of a functional model class, whether its relationship with an **item_class** is part of its definition, or is intended to be defined afterwards. An **abstract_functional_model_class** may reference other functional model classes by its **case_of** attribute.

EXPRESS specification:

```
*)
ENTITY abstract_functional_model_class
ABSTRACT SUPERTYPE OF(ONEOF(functional_model_class,
     fm_class_view_of))
SUBTYPE OF(class);
     created_view: class_BSU;
     v_c_v_range: SET [0:?] OF view_control_variable_range;
     imported_properties_from_view: LIST [0:?] OF property_BSU;
     imported_types_from_view: SET [0:?] OF data_type_BSU;
     imported_tables_from_view: SET [0:?] OF table_BSU;
     imported_documents_from_view: SET [0:?] OF document_BSU;
     case_of: SET [0:?] OF class_BSU;
     imported_properties_from_models: LIST [0:?] OF property_BSU;
     imported_types_from_models: SET [0:?] OF data_type_BSU;
     imported_tables_from_models: SET [0:?] OF table_BSU;
     imported_documents_from_models: SET [0:?] OF document_BSU;
WHERE
     WR1: created_view_is_functional_view(SELF.created_view);
     WR2: QUERY(v_c_v <* SELF.v_c_v_range
          | NOT applicable_properties(SELF.created_view,
          [v_c_v.parameter_type])) = [];
     WR3: QUERY(v_c_v <* SELF.v_c_v_range
          | NOT(v_c_v.parameter_type
          IN SELF.imported_properties_from_view)) = [];
     WR4: NOT EXISTS(SELF\class.its_superclass)
          OR ('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
          +'.ABSTRACT_FUNCTIONAL_MODEL_CLASS'
          IN TYPEOF(SELF\class.its_superclass));
     WR5: QUERY(v_c_v <* SELF.v_c_v_range
          | SIZEOF(QUERY(v_c_v_2 <* SELF.v_c_v_range
          | v_c_v.parameter_type = v_c_v_2.parameter_type)) <> 1) = [];
     WR6: QUERY(prop <* SELF\class.described_by
          | definition_available_implies(prop,
          ('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
          +'.REPRESENTATION_P_DET') IN TYPEOF(prop.definition[1])))
          = SELF\class.described_by;
     WR7: QUERY(cl <* SELF.case_of
          | definition_available_implies(cl,
          ('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
          +'.ABSTRACT_FUNCTIONAL_MODEL_CLASS')
          IN TYPEOF(cl.definition[1])))
          = SELF.case_of;
END_ENTITY; -- abstract_functional_model_class
(*
```

Attribute definitions:

created_view: the **functional_view_class** that characterises the user perspective addressed by the functional model class, and, in the case of an instanciable functional view class, the views that may be generated by the **functional_model_class**.

v_c_v_range: the list of the view control variable range that specifies the various views the **functional_model_class** is able to create.

NOTE 1 When a view control variable of the functional view defined by the **created_view** attribute is not represented in the **v_c_v_range** attribute, its range is its complete value domain.

NOTE 2 The **declared_created_views_are_created_rule** and **all_views_available_for_each_component_rule** rules defined in clause 12 ensures that the set of functional views that may be created either by a **functional_model_class_extension** or by an **explicit_functional_model_class_extension** includes the set of functional views that its corresponding **functional_model_class** declares to be able to create by means of its **created_view** and **v_c_v_range** inherited attributes.

NOTE 3 The **all_views_available_for_each_component_rule** rule defined in clause 12 ensures that the set of functional views that may be created by an **explicit_functional_model_class_extension** includes the set of functional views that its corresponding **functional_model_class** declares to be able to create.

imported_properties_from_view: the list of properties that are imported from the described view.

imported_types_from_view: the list of types that are imported from the described view.

imported_tables_from_view: the set of tables that are imported from the described view.

imported_documents_from_view: the set of documents that are imported from the described view.

case_of: the other **abstract_functional_model_class**es the current **abstract_functional_model_class** is **case_of**.

imported_properties_from_models: the list of properties that are imported from the **case_of abstract_functional_model_class**es.

imported_types_from_models: the list of types that are imported from the **case_of abstract_functional_model_class**es.

imported_tables_from_models: the set of tables that are imported from the described **case_of abstract_functional_model_class**es.

imported_documents_from_models: the set of documents that are imported from the **case_of abstract_functional_model_class**es.

Formal propositions:

WR1: the attribute **created_view** shall correspond to a functional view class.

WR2: each view control variable shall correspond to a view control variable of the functional view that is referenced by the **created_view** attribute.

WR3: each view control variable shall belong to the list of imported properties contained in the **imported_properties_from_view**.

WR4: either the class has no superclass, or the superclass shall be of **abstract_functional_model_class** entity data type.

WR5: each view control variable contained in the attribute **v_c_v_range** shall be unique.

WR6: if data are available, then **IP1** holds.

WR7: the class referenced through the **case_of** attribute shall correspond to functional model classes.

Informal propositions:

IP1: all the properties that are defined in the functional model class shall be defined as **representation_P_DET**.

11.20.2 Functional_model_class

A **functional_model_class** entity captures the dictionary definition of a functional model class that is described without referring to any **item_class**.

NOTE The association between a **functional_model_class** and an **item_class** is intended to be done afterwards, either by the supplier or by the end-user.

EXPRESS specification:

```
*)
ENTITY functional_model_class
SUBTYPE OF(a_priori_semantic_relationship,
     abstract_functional_model_class);
DERIVE
     SELF\a_priori_semantic_relationship.referenced_classes:
          SET [1:?] OF class_BSU :=
          [SELF\abstract_functional_model_class.created_view]
          + SELF\abstract_functional_model_class.case_of;
     SELF\a_priori_semantic_relationship.referenced_properties:
          LIST [0:?] OF property_BSU :=
          SELF\abstract_functional_model_class
               .imported_properties_from_view
          + SELF\abstract_functional_model_class
               .imported_properties_from_models;
     SELF\a_priori_semantic_relationship.referenced_data_types:
          SET [0:?] OF data_type_BSU :=
          SELF\abstract_functional_model_class
               .imported_types_from_view
          + SELF\abstract_functional_model_class
               .imported_types_from_models;
     SELF\a_priori_semantic_relationship.referenced_tables:
          SET [0:?] OF table_BSU :=
          SELF\abstract_functional_model_class
               .imported_tables_from_view
          + SELF\abstract_functional_model_class
               .imported_tables_from_models;
     SELF\a_priori_semantic_relationship.referenced_documents:
          SET [0:?] OF document_BSU :=
          SELF\abstract_functional_model_class
               .imported_documents_from_view
          + SELF\abstract_functional_model_class
               .imported_documents_from_models;
END_ENTITY; -- functional_model_class
(*
```

11.20.3 Fm_class_view_of

A **fm_class_view_of** entity captures the dictionary definition of a functional model class that is defined by reference to an **item_class** to which it applies.

NOTE This kind of association between a **functional_model_class** and an **item_class** is an a priori semantic relationship. The **functional_model_class** may import properties, data types, tables and documents from this **item_class**.

EXPRESS specification:

```
*)
ENTITY fm_class_view_of
SUBTYPE OF(a_priori_semantic_relationship,
      abstract_functional_model_class);
      view_of: class_BSU;
      imported_properties_from_item: LIST [0:?] OF property_BSU;
      imported_types_from_item: SET [0:?] OF data_type_BSU;
      imported_tables_from_item: SET [0:?] OF table_BSU;
      imported_documents_from_item: SET [0:?] OF document_BSU;
DERIVE
      SELF\a_priori_semantic_relationship.referenced_classes:
            SET [2:2] OF class_BSU :=
            [SELF\abstract_functional_model_class.created_view,
            SELF.view_of] + SELF\abstract_functional_model_class.case_of;
      SELF\a_priori_semantic_relationship.referenced_properties:
            LIST [0:?] OF property_BSU :=
            SELF\abstract_functional_model_class
            .imported_properties_from_view
            + SELF\abstract_functional_model_class
            .imported_properties_from_models
            + SELF.imported_properties_from_item;
      SELF\a_priori_semantic_relationship.referenced_data_types:
            SET [0:?] OF data_type_BSU :=
            SELF\abstract_functional_model_class
            .imported_types_from_view
            + SELF\abstract_functional_model_class
            .imported_types_from_models
            + SELF.imported_types_from_item;
      SELF\a_priori_semantic_relationship.referenced_tables:
            SET [0:?] OF table_BSU :=
            SELF\abstract_functional_model_class
            .imported_tables_from_view
            + SELF\abstract_functional_model_class
            .imported_tables_from_models
            + SELF.imported_tables_from_item;
      SELF\a_priori_semantic_relationship.referenced_documents:
            SET [0:?] OF document_BSU :=
            SELF\abstract_functional_model_class
            .imported_documents_from_view
            + SELF\abstract_functional_model_class
            .imported_documents_from_models
```



```
          + SELF.imported_documents_from_item;
WHERE
     WR1: check_view_of_instance_datatype(SELF);
END_ENTITY; -- fm_class_view_of
(*
```

Attribute definitions:

view_of: the **item_class** of which the described **functional_model_class** is able to generate view.

imported_properties_from_item: the properties that are imported from the **item_class** for which the present class is able to generate the view.

imported_types_from_item: the types that are imported from the **item_class** for which the present class is able to generate the view.

imported_tables_from_item: the tables that are imported from the **item_class** for which the present class is able to generate the view.

imported_documents_from_item: the documents that are imported from the **item_class** for which the present class is able to generate the view.

Formal propositions:

WR1: the **view_of** attribute shall refer to an **item_class**.

11.21 ISO13584_extended_dictionary_schema entity definitions: functional view class

A functional view class specifies one representation category that may be represented in a **dictionary** and/or generated by a **library** by means of functional model classes associated with item classes.

EXAMPLE 1 Such representation categories may be geometry, kinematics, or procurement.

If a functional view class is instantiable in a user modelling system, this functional view class shall defines the structure of the representation category it specifies in a modelling system whatever item it refers to.

EXAMPLE 2 Geometry, kinematics and schematics are representation categories.

This is done by referencing an ISO 10303-43 **representation**. The **view_control_variable**s precise the representation category. Their values are to be provided by the user to specify which precise representation is required. The **view_properties** define the content of the view. Their values shall be computed by the user library management system. Both kinds of property shall be represented as **representation_P_DET**s.

A **functional_view_class** is a subtype of ISO 10303-43 **representation**. Therefore, each functional view class instance contains implicitly both a **representation_context** and a set of **representation_item**s referred to by the **items** inherited attributes. When a functional view class is a subtype of a subtype of ISO 10303-43 **representation**, this latter subtype is specified, in the same format as the result of the EXPRESS TYPEOF function, by the **representation_type** attribute of the **functional_view_class** instance that represents the functional view class.

EXAMPLE 3 One specific class of functional view of geometry may be specified by a set of view control variables and by one view property, called **insertion_placements**, that is a SET of placements: simple **placement**, **axis1_placement**, and/or **axis2_placement** intended to be computed by functional model classes and to play a specific role in the representation. As a subtype of **representation** it also contains a

representation_context and a set of **geometric_representation_items** (see ISO 10303-42 for details) referred to by the **items** attributes.

A functional view class that is not instantiable in a user modelling system shall not contain any view property. Functional model classes referring to such a functional view class only contains data intended to be used during the user selection process. Such a functional view class allows the specification of which user perspective these data are intended to be used.

EXAMPLE 4 Inventory management and procurement may be functional views that are not intended to be instanciated.

11.21.1 Functional_view_class

A **functional_view_class** entity is a subtype of a **class** entity. The content of an instance of a functional view class is a subtype of an ISO 10303-43 **representation**. Therefore, it contains two inherited attributes:

— a **context_of_items** attribute that contains the **representation_context** that defines the context of the representation elements that constitute the view;

— an **items** attribute that contains the set of **representation_item**s belonging to the content of the view.

When the content of an instance of a functional view class shall be of a particular subtype of **representation**, this subtype may be defined through the **representation_type** attribute, which contains its fully qualified name.

A **functional_view_class** may also contain **view_control_variable**s which allow to distinguish several **representation**s of the same item in the same representation category, and **view_properties** defining properties that shall be contained in any instance of the functional view class.

EXAMPLE ISO 13584-101 defines the *basic_geometry* functional view class that captures the generic concept of the shape of a part. Instances of this functional view class are instances of STEP **representation**. Five view control variables provide for specifying precisely the various *basic_geometry* representations that may be associated with a library item: 'geometry_level', 'detail_level', 'side', 'variant' and 'unreg_variant'. ISO 13584-101 also specify how to exchange functional model classes that contains FORTRAN parametric programs able to create *basic_geometry* **representation**s.

NOTE 1 The standardised functional view classes are defined by the view exchange protocol series of parts of ISO 13584. Each view exchange protocol specifies the standard data that represents the functional view class, the **view_control_variable**s, and **view_properties** of the view, and how to exchange functional model classes able to create instances of the the functional view class.

EXPRESS specification:

```
*)
ENTITY functional_view_class
SUPERTYPE OF(non_instantiable_functional_view_class)
SUBTYPE OF(class);
     representation_type: OPTIONAL STRING;
     view_control_variables: LIST [0:?] OF UNIQUE property_BSU;
     view_properties: LIST [0:?] OF UNIQUE property_BSU;
DERIVE
     SELF\class.described_by: LIST [0:?] OF UNIQUE property_BSU
          := SELF.view_control_variables + SELF.view_properties;
WHERE
     WR1: QUERY(v_c_v <* SELF.view_control_variables
```



```
            | NOT((data_type_typeof(v_c_v) = [])
            OR (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
            + '.NON_QUANTITATIVE_INT_TYPE' IN data_type_typeof(v_c_v))
            AND ordered_index_value(
            data_type_non_quantitative_int_type(v_c_v)[1].domain))))
            = [];
      WR2: NOT EXISTS(SELF.representation_type)
            OR ('_SCHEMA.' LIKE SELF.representation_type);
      WR3: NOT EXISTS(SELF\class.its_superclass)
            OR (('ISO13584_EXTENDED_DICTIONARY_SCHEMA.'
            + 'FUNCTIONAL_VIEW_CLASS')
            IN TYPEOF(SELF\class.its_superclass));
      WR4: QUERY(prop <* SELF\class.described_by
            | definition_available_implies(prop,
            ('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
            +'.REPRESENTATION_P_DET') IN TYPEOF(prop.definition[1])))
            = SELF\class.described_by;
END_ENTITY; -- functional_view_class
(*
```

Attribute definitions:

representation_type: the specification, in the format of a STRING, of the ISO 10303 **representation** subtype the functional view content is a subtype of.

view_control_variable: the list of properties that are view control variables of the functional view class.

view_properties: the list of properties that are contained in the view.

described_by: the whole set of properties of a functional view class. It is a list that contains the **view_control_variable** list and the **view_properties** list.

Formal propositions:

WR1: the **parameter_type** attribute of all the view control variables shall be a **non_quantitative_int_type** whose values are successive integers.

WR2: the **representation_type** shall contains the string "_SCHEMA.".

NOTE 2 The **representation_type** conatins a string corresponding to an EXPRESS fully qualified name.

WR3: either the class has no superclass, or if it has, the superclass shall be a **functional_view_class**.

WR4: if data are available, then **IP1** holds.

Informal propositions:

IP1: all the properties that are defined in the functional view class shall be defined as **representation_P_DET**.

IP2: the **representation_type** shall contain the fully qualified name, in the same format as the EXPRESS TYPEOF function, of a subtype of ISO 10303-43 **representation**.

11.21.2 Non_instantiable_functional_view_class

A **non_instantiable_functional_view_class** specifies a perspective that may be adopted by the user in a part selection process. No instances of such views shall be created in the user modelling system.

EXPRESS specification:

```
*)
ENTITY non_instantiable_functional_view_class
SUBTYPE OF(functional_view_class);
DERIVE
     SELF\functional_view_class.view_properties
          : LIST [0:?] OF UNIQUE property_BSU := [];
WHERE
     WR1: NOT EXISTS(SELF\functional_view_class.representation_type);
END_ENTITY; -- non_instantiable_functional_view_class
(*
```

Attribute definitions:

SELF\functional_view_class.view_properties: a **non_instantiable_functional_view_class** is not associated with any view properties.

Formal propositions:

WR1: the **representation_type** shall not exist.

11.21.3 Specification of the range of a view control variable

A **view_control_variable_range** specifies a subset of the domain of a view control variable as a range defined by its low bound and its high bound. This range includes its bounds.

NOTE 1 The domain of a view control variable, represented by a **non_quantitative_int_type** entity, is an enumeration type where elements of the enumeration are represented as a set of successive integers. The range consists of all the values whose associated integers are greater or equal to the integer defined as the range low bound and less or equal to the integer defined as the range high bound.

NOTE 2 **non_quantitative_int_type** is defined in the **ISO13584_IEC61360_dictionary_schema** specified in IEC 61360-2, and duplicated for convenience in informative annex D of ISO 13584-42. WR1 rule, in the definition of the **functional_view_class** entity as specified in 11.21.1, ensures that the integers associated with the view control variable values constitutes as a set of successive integers.

EXPRESS specification:

```
*)
ENTITY view_control_variable_range;
     parameter_type: property_BSU;
     range_lobound: INTEGER;
     range_hibound: INTEGER;
WHERE
     WR1: (data_type_typeof(SELF.parameter_type) = [])
          OR ('ISO13584_IEC61360_DICTIONARY_SCHEMA'
          + '.NON_QUANTITATIVE_INT_TYPE'
```



```
          IN data_type_typeof(SELF.parameter_type));
     WR2: SELF.range_lobound <= SELF.range_hibound;
     WR3: view_control_variables_attributes_belong_to_domain(SELF);
END_ENTITY; -- view_control_variable_range
(*
```

Attribute definitions:

parameter_type: the view control variable for which the range is described.

range_lobound: the integer that describes the low bound of the range.

range_hibound: the integer that describes the high bound of the range.

Formal propositions:

WR1: the **parameter_type** attribute shall be a **non_quantitative_int_type**.

WR2: the low bound of the range shall be less or equal to the high bound of the range.

WR3: the attributes **range_lobound** and **range_hibound** shall belong to the domain of a view control variable.

11.22 ISO13584_extended_dictionary_schema entity definitions: item class a priori case of

11.22.1 Item_class_case_of

An **item_class_case_of** is the description of an item class that is defined as a is-case-of of some other item class(es).

NOTE An **item_class_case_of** defines an a priori relationship.

EXPRESS specification:

```
*)
ENTITY item_class_case_of
SUPERTYPE OF(ONEOF(component_class_case_of,
     material_class_case_of,
     feature_class_case_of))
SUBTYPE OF(item_class, a_priori_semantic_relationship);
     is_case_of: SET [1:?] OF class_BSU;
     imported_properties: LIST [0:?] OF property_BSU;
     imported_types: SET [0:?]OF data_type_BSU;
     imported_tables: SET [0:?] OF table_BSU;
     imported_documents: SET [0:?] OF document_BSU;
DERIVE
     SELF\a_priori_semantic_relationship.referenced_classes:
          SET [1:?] OF class_BSU
          := SELF.is_case_of;
     SELF\a_priori_semantic_relationship.referenced_properties:
          LIST [0:?] OF property_BSU := SELF.imported_properties;
     SELF\a_priori_semantic_relationship.referenced_data_types:
```



```
            SET [0:?] OF data_type_BSU := SELF.imported_types;
      SELF\a_priori_semantic_relationship.referenced_tables:
            SET [0:?] OF table_BSU := SELF.imported_tables;
      SELF\a_priori_semantic_relationship.referenced_documents:
            SET [0:?] OF document_BSU := SELF.imported_documents;
WHERE
      WR1: superclass_of_item_is_item(SELF);
      WR2: check_is_case_of_referenced_classes_definition(SELF);
END_ENTITY; -- item_class_case_of
(*
```

Attribute definitions:

is_case_of: the **item_class**(es) of which the present **item_class** is-case-of.

imported_properties: the list of properties that are imported from the **item_class**(es) the defined **item_class** is-case-of.

imported_types: the set of data types that are imported from the **item_class**(es) the defined **item_class** is-case-of.

Imported_tables: the set of **table_BSU**s that are imported from the **item_class**(es) the defined **item_class** is-case-of.

Imported_documents: the set of **document_BSU**s that are imported from the **item_class**(es) the defined **item_class** is-case-of.

Formal propositions:

WR1: the superclass of an **item_class_case_of** shall be an **item_class**.

WR2: an **item_class_case_of** shall be case-of **item_class**(es).

11.22.2 Component_class_case_of

A **component_class_case_of** is the description of a component class that is defined as a case of other component class(es).

NOTE A **component_class_case_of** defines an a priori relationship.

EXPRESS specification:

```
*)
ENTITY component_class_case_of
SUBTYPE OF(item_class_case_of, component_class);
WHERE
      WR1: check_is_case_of_referenced_classes_definition(SELF);
END_ENTITY; -- component_class_case_of
(*
```


Formal propositions:

WR1: a **component_class_case_of** shall be is-case-of **component_class**(es).

11.22.3 Material_class_case_of

A **material_class_case_of** is the description of a material class that is defined as a case of other material class(es).

NOTE A **material_class_case_of** defines an a priori relationship.

EXPRESS specification:

```
*)
ENTITY material_class_case_of
SUBTYPE OF(item_class_case_of, material_class);
WHERE
     WR1: check_is_case_of_referenced_classes_definition(SELF);
END_ENTITY; -- material_class_case_of
(*
```

Formal propositions:

WR1: a **material_class_case_of** shall be is-case-of **material_class**(es).

11.22.4 Feature_class_case_of

A **feature_class_case_of** is the description of a feature class that is defined as a case of other feature class(es).

NOTE A **feature_class_case_of** defines an a priori relationship.

EXPRESS specification:

```
*)
ENTITY feature_class_case_of
SUBTYPE OF(item_class_case_of, feature_class);
WHERE
     WR1: check_is_case_of_referenced_classes_definition(SELF);
END_ENTITY; -- feature_class_case_of
(*
```

Formal propositions:

WR1: a **feature_class_case_of** shall be is-case-of **feature_class**(es).

11.23 ISO13584_extended_dictionary_schema entity definitions: a posteriori semantic relationships

In an a posteriori definition of semantic relationships between classes, each class defines all its properties. Therefore, it is self sufficient even if the referenced class is not available.

This mechanism also allows suppliers or end-users to define their own semantic relationships.

11.23.1 A_posteriori_semantic_relationship

An **a_posteriori_semantic_relationship** is a relationship between two classes.

EXPRESS specification:

```
*)
ENTITY a_posteriori_semantic_relationship
ABSTRACT SUPERTYPE OF(ONEOF(a_posteriori_case_of,
     a_posteriori_view_of));
END_ENTITY; -- a_posteriori_semantic_relationship
(*
```

11.23.2 A_posteriori_case_of

An **a_posteriori_case_of** specifies a is-case-of relationship between two classes.

EXPRESS specification:

```
*)
ENTITY a_posteriori_case_of
SUBTYPE OF(a_posteriori_semantic_relationship);
     source: class_BSU;
     is_case_of: class_BSU;
     corresponding_properties: SET [0:?] OF
          LIST [2:2] OF property_BSU;
WHERE
     WR1: definition_available_implies(SELF.source,
          ('ISO13584_IEC61360_DICTIONARY_SCHEMA.ITEM_CLASS'
          IN TYPEOF(SELF.source.definition[1]))
          OR (('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
           + '.ABSTRACT_FUNCTIONAL_MODEL_CLASS')
          IN TYPEOF(SELF.source.definition[1])));
     WR2: definition_available_implies(SELF.source,
          (('ISO13584_IEC61360_DICTIONARY_SCHEMA.ITEM_CLASS'
          IN TYPEOF(SELF.source.definition[1]))
          AND (definition_available_implies(SELF.is_case_of,
          ('ISO13584_IEC61360_DICTIONARY_SCHEMA.ITEM_CLASS'
          IN TYPEOF(SELF.source.definition[1])))))
          OR (('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
           + '.ABSTRACT_FUNCTIONAL_MODEL_CLASS'
          IN TYPEOF(SELF.source.definition[1]))
          AND (definition_available_implies(SELF.is_case_of,
          ('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
           + '.ABSTRACT_FUNCTIONAL_MODEL_CLASS'
          IN TYPEOF(SELF.source.definition[1]))))));
     WR3: QUERY(couple <* SELF.corresponding_properties
          | NOT compatible_types(couple[1], couple[2])) = [];
     WR4: QUERY(couple <* SELF.corresponding_properties
          | (NOT applicable_properties(SELF.source,[couple[1]])
          OR NOT applicable_properties(SELF.is_case_of,[couple [2]])))
```



```
          = [];
END_ENTITY; -- a_posteriori_case_of
(*
```

Attribute definitions:

source: the class that is is-case-of of the **is_case_of** attribute.

is_case_of: the class to which the **source** attribute is is-case-of.

corresponding_properties: the set of pairs of properties that correspond to each other in the is-case-of relationship, the first property of each pair belonging to the **source** class and the second to the **is_case_of** class.

Formal propositions:

WR1: if data are available, then **IP1** holds.

WR2: if data are available, then **IP2** holds.

WR3: all the corresponding properties of the list of **corresponding_properties** shall be type compatible.

WR4: each property involved in each pair of the **corresponding_properties** list shall be applicable properties for one of the two **class** involved in the relationship.

Informal propositions:

IP1: the **source** attribute shall refer either to an **item_class** or to an **abstract_functioinal_model_class**.

IP2: the **source** and **is_case_of** attribute shall either be both **item_class**es or be both **abstract_functional_model_class**es.

11.23.3 A_posteriori_view_of

An **a_posteriori_view_of** entity is a relationship between a functional model class and an item class.

EXPRESS specification:

```
*)
ENTITY a_posteriori_view_of
SUBTYPE OF(a_posteriori_semantic_relationship);
      functional_model: class_BSU;
      is_view_of: class_BSU;
      corresponding_properties: SET [0:?] OF
           LIST [2:2] OF property_BSU;
WHERE
      WR1: definition_available_implies(SELF.functional_model,
           ('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
           +'.FUNCTIONAL_MODEL_CLASS' IN TYPEOF(
           SELF.functional_model.definition[1])));
      WR2: definition_available_implies(SELF.is_view_of,
```



```
            ('ISO13584_IEC61360_DICTIONARY_SCHEMA.ITEM_CLASS'
            IN TYPEOF(SELF.is_view_of.definition[1])));
       WR3: QUERY(couple <* SELF.corresponding_properties
            | NOT compatible_types(couple [1], couple [2])) = [];
       WR4: QUERY(couple <* SELF.corresponding_properties
            | (NOT applicable_properties(
            SELF.functional_model,[couple[1]]) OR NOT
            applicable_properties(SELF.is_view_of,[couple [2]]))) = [];
END_ENTITY; -- a_posteriori_view_of
(*
```

Attribute definitions:

functional_model: the functional model class that is is-view-of of the **is_view_of** class.

is_view_of: the **item_class** for which the **functional_model** class is is-view-of.

corresponding_properties: a set of pairs of properties belonging respectively to the **functional_model** class and to the is-view-of **item_clas**s class.

Formal propositions:

WR1: if data are available, then **IP1** holds.

WR2: if data are available, then **IP2** holds.

WR3: all the corresponding properties of the list of **corresponding_properties** shall be type compatible.

WR4: the properties defined in each pair of the **corresponding_properties** list shall be applicable properties for their corresponding class.

Informal propositions:

IP1: the **functional_model** attribute shall refer to a **functional_model_class**.

IP2: the **is_view_of** attribute shall refer to an **item_class**.

11.24 ISO13584_extended_dictionary_schema entity definitions: table contents

11.24.1 Table_content

A **table_content** entity is the **table_extension** of a table associated with a class in an ISO 13584-conformant dictionary. The **columns** that constitute the **content** of the **table_content** shall be type-compatible with the **self_property_semantics** that define the meaning of these **columns**.

A **table_content** inherits a **revision_of_content** attribute, that specifies the revision number of the set of values contained in the table, and a **content_revision_date** attribute that specifies at what time this set of values has been defined by the supplier. A revision shall neither change the set of key values in the table, nor change the list of **columns**. It may only change values of non-key **column**s.

NOTE A **table_BSU** has also a **version** number that characterises the set of key values contained in the table and the list of columns that constitute its **content**.

EXPRESS specification:

```
*)
ENTITY table_content
SUBTYPE OF(content_item, table_extension);
     SELF\content_item.dictionary_definition: table_BSU;
DERIVE
     SELF\table_extension.table_identifier: table_bsu
          := SELF\content_item.dictionary_definition;
WHERE
     WR1: definition_available_implies(
          SELF\content_item.dictionary_definition,
          'ISO13584_EXTENDED_DICTIONARY_SCHEMA.TABLE_ELEMENT'
          IN TYPEOF(SELF\content_item.dictionary_definition.
          definition[1]));
     WR2: compatible_content_and_specification(SELF);
END_ENTITY; -- table_content
(*
```

Attribute definitions:

dictionary_definition: inherited attribute specialised so as to refer to the table for which the current entity describes the content.

Formal propositions:

WR1: if data are available, then **IP1** holds.

WR2: the number of **columns** is correct with respect to the table **table_element** description, and the types and values of each **column** are compatible and correspond to the type defined in the **data_type** specification of the corresponding **property_semantics**.

Informal propositions:

IP1: the **dictionary_element** related to the **identified_by** attribute is a **table_element**.

IP2: all the revisions that correspond to the same **table_BSU version** have the same set of key values and the same **content** LIST. The only possible change from revision to revision is correcting typos in values or providing more accurate values

NOTE 1 This informal proposition only specifies the allowed changes from revision to revision. However it is up to the supplier, when making such a change, to decide whether it is a revision that does not affect the **table_BSU** identifier of the table or whether it is a new version of the table that changes its identification **version** number.

NOTE 2 This informal proposition states that when updating a dictionary, an old revision may be replaced by a new one without any consequence on the functional behavioir of the integrated library.

11.24.2 RDB_table_content

A **RDB_table_content** is a structure that restricts a **table_content** to tables that are relational database compatible.

EXPRESS specification:

```
*)
ENTITY RDB_table_content
SUBTYPE OF(table_content, RDB_table_extension);
WHERE
     WR1: definition_available_implies(
          SELF\content_item.dictionary_definition,
          'ISO13584_EXTENDED_DICTIONARY_SCHEMA.RDB_TABLE_ELEMENT'
          IN TYPEOF(SELF\content_item.dictionary_definition.
          definition[1]));
END_ENTITY; -- RDB_table_content
(*
```

Formal propositions:

WR1: if data are available, then **IP1** holds.

Informal propositions:

IP1: the **dictionary_element** related to the **identified_by** attribute is a **RDB_table_element**.

IP2: all the revisions that correspond to the same **table_BSU version** have the same set of key values and the same **rdb_class_extension** content LIST. The only possible change from revision to revision is changing non-key values.

11.25 ISO13584_extended_dictionary_schema: RULE definitions

This section presents the different EXPRESS rules that are used in this part of the ISO 13584 standard. These rules are related to:

— the dictionary element that describes a property defined in a model class,

— the declaration of a property in a class,

— the declaration of a type in a class,

— the usage of the named types,

— the program libraries that are associated with suppliers,

— the documents and tables that are associated with classes,

— the properties, data types, documents and tables that are imported by means of a semantic relationship,

— the levels that are used in the view control variables.

11.25.1 Representation_properties_for_model_and_view_rule rule

The **representation_property_for_model_and_view_rule** rule states that a property defined in a **functional_model_class** or a **functional_view_class** shall be described as a **representation_P_DET**. Only an **item_class** may describe properties as **condition_DET**, **dependent_P_DET** or **non_dependent_P_DET**.

EXPRESS specification:

```
*)
RULE representation_properties_for_model_and_view_rule
     FOR(property_DET);
WHERE
     WR1: QUERY(prop <* property_DET
          | (SIZEOF(prop.identified_by.name_scope.definition) = 1)
          AND (('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
          +'.FUNCTIONAL_MODEL_CLASS' IN TYPEOF(
          prop.identified_by.name_scope.definition))
          OR ('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
          +'.FUNCTIONAL_VIEW_CLASS' IN TYPEOF(
          prop.identified_by.name_scope.definition)))
          AND NOT('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
          +'.REPRESENTATION_P_DET' IN TYPEOF(prop))
          ) = [];
END_RULE; -- representation_properties_for_model_and_view_rule
(*
```

Formal propositions:

WR1: a property defined in a **functional_model_class** or a **functional_view_class** shall be described as a **representation_P_DET**.

11.25.2 Allowed_named_type_usage_rule rule

The **allowed_named_type_usage_rule** rule is related to the usage of a named type. It states that only types that are applicable to a class may be used to specify the domain of the properties declared by a class, through its **described_by** attribute.

EXPRESS specification:

```
*)
RULE allowed_named_type_usage_rule FOR(class);
LOCAL
     named_type_usage_allowed: LOGICAL := TRUE;
     is_app: LOGICAL;
     prop: property_bsu;
     cl: class;
     dtnt: SET[0:1] OF data_type_bsu := [];
END_LOCAL;

REPEAT i := 1 TO SIZEOF(class);
     cl := class[i];
     REPEAT j := 1 TO SIZEOF(class[i].described_by);
          prop := cl.described_by[j];
          dtnt := data_type_named_type(prop);

          IF (SIZEOF(dtnt) = 1) THEN
               is_app := applicable_types(cl.identified_by, dtnt);
               IF (NOT is_app) THEN
```

```
                    named_type_usage_allowed := FALSE;
                END_IF;
            END_IF;
        END_REPEAT;
    END_REPEAT;

    WHERE
        WR1: named_type_usage_allowed;
    END_RULE; -- allowed_named_type_usage_rule
    (*
```

Formal propositions:

WR1: only types that are applicable to a **class** may be used to specify the domain of the properties declared by a **class**, through its **described_by** attribute.

11.25.3 Assert_oneof_for_table_rule rule

The **assert_oneof_for_table_rule** rule states that a **table_element** that is not a **RDB_table_element** cannot be a **RDB_table_specification**.

EXPRESS specification:

```
    *)
    RULE assert_oneof_for_table_rule FOR(table_element);
    WHERE
        WR1: QUERY(temp <* table_element |
            NOT('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
            + '.RDB_TABLE_ELEMENT' IN TYPEOF(temp))
            AND ('ISO13584_TABLE_RESOURCE_SCHEMA.RDB_TABLE_SPECIFICATION'
            IN TYPEOF(temp))) = [];
    END_RULE; -- assert_oneof_for_table_rule
    (*
```

Formal propositions:

WR1: a **table** that is not a **RDB_table_element** cannot be a **RDB_table_specification**.

11.25.4 Assert_oneof_for_class_rule rule

The **assert_oneof_for_class_rule** rule states that a **class** shall not be at the same time an **item_class** and/or a **functional_view_class** and/or an **abstract_functional_model_class**.

EXPRESS specification:

```
    *)
    RULE assert_oneof_for_class_rule FOR(class);
    WHERE
        WR1: QUERY(cl <* class |
            ('ISO13584_IEC61360_DICTIONARY_SCHEMA.ITEM_CLASS'
            IN TYPEOF(cl))
```



```
            AND (('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
            +'.ABSTRACT_FUNCTIONAL_MODEL_CLASS') IN TYPEOF(cl))) = [];
      WR2: QUERY(cl <* class |
            ('ISO13584_IEC61360_DICTIONARY_SCHEMA.ITEM_CLASS'
            IN TYPEOF(cl))
            AND (('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
            + '.FUNCTIONAL_VIEW_CLASS') IN TYPEOF(cl))) = [];
      WR3: QUERY(cl <* class |
            (('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
            + '.ABSTRACT_FUNCTIONAL_MODEL_CLASS')
            IN TYPEOF(cl))
            AND (('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
            +'.FUNCTIONAL_VIEW_CLASS') IN TYPEOF(cl))) = [];
END_RULE; -- assert_oneof_for_class_rule
(*
```

Formal propositions:

WR1: a **class** shall not be at the same time an **item_class** and an **abstract_functional_model_class**.

WR2: a **class** shall not be at the same time an **item_class** and a **functional_view_class**.

WR3: a **class** shall not be at the same time a **functional_view_class** and an **abstract_functional_model_class**.

11.25.5 No_forward_reference_from_table_rule rule

The **no_forward_reference_from_table_rule** rule checks that when a PLIB conformant library exchange context includes **table_element**s, these **table_element**s do not refer through their **column_meaning** inherited attribute to a **property_BSU** that makes a forward reference to a **class_BSU** with respect of the **class_BSU** list order defined by the **contained_classes** attribute of any **dictionary** entity included in the library exchange context.

NOTE The library integrated information model defined in this part of ISO 13584 only permits one **dictionary** entity within a library exchange context.

EXPRESS specification:

```
*)
RULE no_forward_reference_from_table_rule FOR(
          dictionary, table_element);
WHERE
      WR1: QUERY(dic <* dictionary | QUERY(tab <* table_element
          | makes_reference_outside(
          get_property_BSU_from_property_semantics(
          tab\table_specification.column_meaning),
          sub_list_until(dic.contained_classes,
          tab\table_specification.table_identifier\
          table_BSU.name_scope))) <> []) = [];
END_RULE; -- no_forward_reference_from_table_rule
(*
```

Formal propositions:

WR1: there is no forward references from any **table_element** to a **property_bsu** with respect of the **class_BSU** list order defined by the **contained_classes** attribute of any **dictionary** entity included in the library exchange context.

11.25.6 Imported_properties_are_visible_or_applicable_rule rule

The **imported_properties_are_visible_or_applicable_rule** rule checks that when a property is imported by a class by means of an **a_priori_semantic_relationship**, this property is visible or applicable for the class it is imported from.

NOTE Applicable properties include the properties imported through a semantic relationship. This rule enables to import properties from a class where they were already imported.

EXPRESS specification:

```
*)
RULE imported_properties_are_visible_or_applicable_rule FOR(
     a_priori_semantic_relationship, property_DET);
WHERE
     WR1: QUERY(rel <* a_priori_semantic_relationship
          | QUERY(prop <* rel.referenced_properties
          | QUERY(cl <* rel.referenced_classes
          | NOT visible_properties(cl, [prop])
          AND NOT applicable_properties(cl, [prop]))
          = rel.referenced_classes) = [])
          = a_priori_semantic_relationship;
END_RULE; -- imported_properties_are_visible_or_applicable_rule
(*
```

Formal propositions:

WR1: the imported properties defined by the **referenced_properties** attribute of any **a_priori_semantic_relationship** shall be visible or applicable for one of the classes belonging to the **referenced_classes** set of this **a_priori_semantic_relationship**.

11.25.7 Imported_data_types_are_visible_or_applicable_rule rule

The **imported_data_types_are_visible_or_applicable_rule** rule checks that when a data type is imported by a class by means of an **a_priori_semantic_relationship**, this data type is visible or applicable for the class it is imported from.

NOTE Applicable data types include the data types imported through a semantic relationship. This rule enables to import data types from a class where they were already imported.

EXPRESS specification:

```
*)
RULE imported_data_types_are_visible_or_applicable_rule FOR(
     a_priori_semantic_relationship, data_type_element);
WHERE
     WR1: QUERY(rel <* a_priori_semantic_relationship
```



```
          | QUERY(typ <* rel.referenced_data_types
          | QUERY(cl <* rel.referenced_classes
          | NOT visible_types(cl, [typ])
          AND NOT applicable_types(cl, [typ]))
          = rel.referenced_classes) = [])
          = a_priori_semantic_relationship;
END_RULE; -- imported_data_types_are_visible_or_applicable_rule
(*
```

Formal propositions:

WR1: the imported types defined by the **referenced_data_types** attribute of any **a_priori_semantic_relationship** shall be visible or applicable for one of the classes belonging to the **referenced_classes** set of this **a_priori_semantic_relationship**.

11.25.8 Imported_tables_are_visible_or_applicable_rule rule

The **imported_tables_are_visible_or_applicable_rule** rule checks that when a table is imported by a class by means of an **a_priori_semantic_relationship**, this table is visible or applicable for the class it is imported from.

NOTE Applicable tables include the tables imported through a semantic relationship. This rule enables to import tables from a class where they were already imported.

EXPRESS specification:

```
*)
RULE imported_tables_are_visible_or_applicable_rule FOR(
     a_priori_semantic_relationship, table_element);
WHERE
     WR1: QUERY(rel <* a_priori_semantic_relationship
          | QUERY(tab <* rel.referenced_tables
          | QUERY(cl <* rel.referenced_classes
          | NOT visible_tables(cl, [tab])
          AND NOT applicable_tables(cl, [tab]))
          = rel.referenced_classes) = [])
          = a_priori_semantic_relationship;
END_RULE; -- imported_tables_are_visible_or_applicable_rule
(*
```

Formal propositions:

WR1: the imported tables defined by the **referenced_tables** attribute of any **a_priori_semantic_relationship** shall be visible or applicable for one of the classes belonging to the **referenced_classes** set of this **a_priori_semantic_relationship**.

11.25.9 Imported_documents_are_visible_or_applicable_rule rule

The **imported_documents_are_visible_or_applicable_rule** rule checks that when a document is imported by a class by means of an **a_priori_semantic_relationship**, this document is visible or applicable for the class it is imported from.

NOTE Applicable documents include the documents imported through a semantic relationship. This rule enables to import documents from a class where they were already imported.

EXPRESS specification:

```
*)
RULE imported_documents_are_visible_or_applicable_rule FOR(
     a_priori_semantic_relationship, document_element);
WHERE
     WR1: QUERY(rel <* a_priori_semantic_relationship
          | QUERY(doc <* rel.referenced_documents
          | QUERY(cl <* rel.referenced_classes
          | NOT visible_documents(cl, [doc])
          AND NOT applicable_documents(cl, [doc]))
          = rel.referenced_classes) = [])
          = a_priori_semantic_relationship;
END_RULE; -- imported_documents_are_visible_or_applicable_rule
(*
```

Formal propositions:

WR1: the imported documents defined by the **referenced_documents** attribute of any **a_priori_semantic_relationship** shall be visible or applicable for one of the classes belonging to the **referenced_classes** set of this **a_priori_semantic_relationship**.

11.26 ISO13584_extended_dictionary_schema: function definitions

This section gathers the set of functions that are needed for the specification of the constraints on the entities defined in the **ISO13584_extended_dictionary_schema**.

11.26.1 Visible_properties function

The **visible_properties** function checks that the properties corresponding to **prop** are visible from the class identified by the **cl** parameter. A property is visible from a class if its DET refers to the BSU of this class or any of its superclass.

If the known visible properties are possibly incomplete because a superclass of **cl** is not available in the current exchange context, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION visible_properties(cl: class_BSU;
     prop: AGGREGATE OF property_BSU): LOGICAL;

LOCAL
     ok: BOOLEAN := TRUE;
END_LOCAL;

IF NOT EXISTS(cl)
THEN
     RETURN(UNKNOWN);
END_IF;

REPEAT i := 1 to SIZEOF(prop);
```



```
      IF NOT(prop[i] IN compute_known_visible_properties(cl))
           THEN ok := FALSE;
      END_IF;
END_REPEAT;

IF NOT(ok) AND NOT(all_class_descriptions_reachable(cl))
      THEN RETURN(UNKNOWN);
END_IF;

IF NOT(ok) AND all_class_descriptions_reachable(cl)
      THEN RETURN(FALSE);
END_IF;

RETURN(TRUE);

END_FUNCTION; -- visible_properties
(*
```

11.26.2 Visible_types function

The **visible_types** function checks that the types corresponding to **typ** are visible from the class identified by the **cl** parameter. A type is visible from a class if its DET refers to the BSU of this class or any of its superclass.

If the known visible data types are possibly incomplete because a superclass of **cl** is not available in the current exchange context, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION visible_types(cl: class_BSU;
      typ: AGGREGATE OF data_type_BSU): LOGICAL;

LOCAL
      ok: BOOLEAN := TRUE;
END_LOCAL;

IF NOT EXISTS(cl)
THEN
      RETURN(UNKNOWN);
END_IF;

REPEAT i := 1 to SIZEOF(typ);
IF NOT(typ[i] IN compute_known_visible_data_types(cl))
      THEN ok := FALSE;
END_IF;
END_REPEAT;

IF NOT(ok) AND NOT(all_class_descriptions_reachable(cl))
      THEN RETURN(UNKNOWN);
END_IF;

IF NOT(ok) AND all_class_descriptions_reachable(cl)
```



```
     THEN RETURN(FALSE);
END_IF;

RETURN(TRUE);

END_FUNCTION; -- visible_types
(*
```

11.26.3 Visible_tables function

The **visible_table**s function checks that the tables corresponding to **tab** are visible from the class identified by the **cl** parameter. A table is visible from a class if its DET refers to the BSU of this class or any of its superclass.

If a BSU definition is not available to compute the whole set of visible tables and if a **table_BSU** from **tab** has not been proved to be visible, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION visible_tables(cl: class_BSU;
     tab: AGGREGATE OF table_BSU): LOGICAL;

IF NOT EXISTS(cl)
THEN
     RETURN(UNKNOWN);
END_IF;

REPEAT i := SIZEOF(tab) TO 1 BY -1;
     IF tab[i].name_scope = cl
     THEN
          tab := tab - tab[i];
     END_IF;
END_REPEAT;

IF SIZEOF(tab) = 0
THEN
     RETURN(TRUE);
END_IF;

IF SIZEOF(cl.definition) = 0
THEN
     RETURN(UNKNOWN);
END_IF;

IF EXISTS(cl.definition[1]\class.its_superclass)
THEN
     RETURN(visible_tables(cl.definition[1]
          \class.its_superclass, tab));
ELSE
     RETURN(FALSE);
END_IF;
```



```
END_FUNCTION; -- visible_tables
(*
```

11.26.4 Visible_documents function

The **visible_documents** function checks that the documents corresponding to **doc** are visible from the class identified by the **cl** parameter. A document is visible from a class if its DET refers to the BSU of this class or any of its superclass.

If a BSU definition is not available to compute the whole set of visible documents and if a **document_BSU** from **doc** has not been proved to be visible, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION visible_documents(cl: class_BSU;
      doc: AGGREGATE OF document_BSU): LOGICAL;

IF NOT EXISTS(cl)
THEN
      RETURN(UNKNOWN);
END_IF;

REPEAT i := SIZEOF(doc) TO 1 BY -1;
      IF doc[i].name_scope = cl
      THEN
           doc := doc - doc[i];
      END_IF;
END_REPEAT;

IF SIZEOF(doc) = 0
THEN
      RETURN(TRUE);
END_IF;

IF SIZEOF(cl.definition) = 0
THEN
      RETURN(UNKNOWN);
END_IF;

IF EXISTS(cl.definition[1]\class.its_superclass)
THEN
      RETURN(visible_documents(cl.definition[1]
           \class.its_superclass, doc));
ELSE
      RETURN(FALSE);
END_IF;

END_FUNCTION; -- visible_documents
(*
```

11.26.5 Applicable_properties function

The **applicable_properties** function checks that the properties corresponding to **prop** are applicable to the class identified by the **cl** parameter. A property is applicable to a class if its **property_BSU** belongs to the **described_by** attribute of this class or any of its super-classes, or if its **property_BSU** is imported by the class, or by any of its super-classes.

Note that, in particular, all the properties belonging to the **known_applicable_properties** attribute of a class are applicable to this class.

If a **dictionary_element** is not available to compute the whole set of applicable properties and if a **property_BSU** from **prop** has not been proved to be applicable, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION applicable_properties(cl: class_BSU;
     prop: AGGREGATE OF property_BSU): LOGICAL;

IF SIZEOF(prop) = 0
THEN
     RETURN(TRUE);
END_IF;

IF NOT EXISTS(cl)
THEN
     RETURN(UNKNOWN);
END_IF;

IF SIZEOF(cl.definition) = 0
THEN
     RETURN(UNKNOWN);
END_IF;

prop := prop - list_to_set(cl.definition[1]\class.described_by);

IF 'ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
     '.A_PRIORI_SEMANTIC_RELATIONSHIP' IN TYPEOF(cl.definition[1])
THEN
     prop := prop - list_to_set(cl.definition[1]
          \a_priori_semantic_relationship.referenced_properties);
END_IF;

IF SIZEOF(prop) = 0
THEN
     RETURN(TRUE);
ELSE
     IF EXISTS(cl.definition[1]\class.its_superclass)
     THEN
          RETURN(applicable_properties(cl.definition[1]
               \class.its_superclass, prop));
     ELSE
```



```
          RETURN(FALSE);
     END_IF;
END_IF;

END_FUNCTION; -- applicable_properties
(*
```

11.26.6 Applicable_types function

The **applicable_types** function checks that the types corresponding to **typ** are applicable to the class identified by the **cl** parameter. A type is applicable to a class if its **data_type_BSU** belongs to the **defined_types** attribute of this class or any of its super-classes, or if its **data_type_BSU** is imported by the class, or any of its super-classes.

Note that, in particular, all the data types belonging to the **known_applicable_data_types** attribute of a class are applicable to this class.

If a **dictionary_element** is not available to compute the whole set of applicable types and if a **data_type_BSU** from **typ** has not been proved to be applicable, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION applicable_types(cl: class_BSU;
     typ: AGGREGATE OF data_type_BSU): LOGICAL;

IF SIZEOF(typ) = 0
THEN
     RETURN(TRUE);
END_IF;

IF NOT EXISTS(cl)
THEN
     RETURN(UNKNOWN);
END_IF;

IF SIZEOF(cl.definition) = 0
THEN
     RETURN(UNKNOWN);
END_IF;

typ := typ - cl.definition[1]\class.defined_types;

IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
     '.A_PRIORI_SEMANTIC_RELATIONSHIP' IN TYPEOF(cl.definition[1]))
THEN
     typ := typ -
          cl.definition[1]\a_priori_semantic_relationship
          .referenced_data_types;
END_IF;

IF SIZEOF(typ) = 0
THEN
```

```
      RETURN(TRUE);
ELSE
      IF EXISTS(cl.definition[1]\class.its_superclass)
      THEN
           RETURN(applicable_types(cl.definition[1]
                \class.its_superclass, typ));
      ELSE
           RETURN(FALSE);
      END_IF;
END_IF;

END_FUNCTION; -- applicable_types
(*
```

11.26.7 Applicable_tables function

The **applicable_tables** function checks that the tables corresponding to **tab** are applicable to the class identified by the **cl** parameter. A table is applicable to a class if a **class_table_relationship** contains the corresponding **table_BSU** in its **related_tokens** list and refers to the class, or to any of its super-classes, as its **relating_class** attributes. It is also applicable if the corresponding **table_BSU** is imported by the class, or any of its super-classes.

If a **dictionary_element** is not available to compute the whole set of applicable tables and if a **table_BSU**s from **tab** has not been proved to be applicable, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION applicable_tables(cl: class_BSU;
      tab: AGGREGATE OF table_identification): LOGICAL;

IF SIZEOF(tab) = 0
THEN
      RETURN(TRUE);
END_IF;

IF NOT EXISTS(cl)
THEN
      RETURN(UNKNOWN);
END_IF;

IF SIZEOF(cl.definition) = 0
THEN
      RETURN(UNKNOWN);
END_IF;

tab := tab - retrieve_tables(cl);

IF 'ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
      '.A_PRIORI_SEMANTIC_RELATIONSHIP' IN TYPEOF(cl.definition[1])
THEN
      tab := tab - cl.definition[1]\a_priori_semantic_relationship
```



```
          .referenced_tables;
END_IF;

IF SIZEOF(tab) = 0
THEN
     RETURN(TRUE);
ELSE
     IF EXISTS(cl.definition[1]\class.its_superclass)
     THEN
     RETURN(
          applicable_tables(cl.definition[1]\class.its_superclass,
               tab));
     ELSE
          RETURN(FALSE);
     END_IF;
END_IF;

END_FUNCTION; -- applicable_tables
(*
```

11.26.8 Retrieve_tables function

The **retrieve_tables** function collects the set of tables directly associated with a class through a **class_table_relationship**.

EXPRESS specification:

```
*)
FUNCTION retrieve_tables(cl: class_BSU): SET[0:?] OF table_BSU;
-- requires: { SIZEOF(cl.definition) <> 0 }

LOCAL
     s: SET[0:?] OF table_BSU := [];
END_LOCAL;

REPEAT i := 1 TO SIZEOF(cl.definition[1]\class.associated_items);
     IF 'ISO13584_EXTENDED_DICTIONARY_SCHEMA'
          +'.CLASS_TABLE_RELATIONSHIP'
          IN TYPEOF(cl.definition[1]\class.associated_items[i])
     THEN
          s := s + cl.definition[1]\class.associated_items[i]
               .related_tokens;
     END_IF;
END_REPEAT;
RETURN(s);

END_FUNCTION; -- retrieve_tables
(*
```

11.26.9 Applicable_documents function

The **applicable_documents** function checks that the documents corresponding to **doc** are applicable to the class identified by the **cl** parameter. A document is applicable to a class if a

class_document_relationship contains the corresponding **document_BSU** in its **related_tokens** list, and refers to the class, or to any of its super-classes, as its **relating_class** attributes. It is also applicable if the corresponding **document_BSU** is imported by the class, or any of its super-classes.

If a **dictionary_element** is not available to compute the whole set of applicable documents and if a **document_BSU** from **doc** has not been proved to be applicable, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION applicable_documents(cl: class_BSU;
     doc: AGGREGATE OF document_BSU): LOGICAL;

IF SIZEOF(doc) = 0
THEN
     RETURN(TRUE);
END_IF;

IF NOT EXISTS(cl)
THEN
     RETURN(UNKNOWN);
END_IF;

IF SIZEOF(cl.definition) = 0
THEN
     RETURN(UNKNOWN);
END_IF;

doc := doc - retrieve_documents(cl);

IF 'ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
     '.A_PRIORI_SEMANTIC_RELATIONSHIP' IN TYPEOF(cl.definition[1])
THEN
     Doc := doc - cl.definition[1]\a_priori_semantic_relationship.
          referenced_documents;
END_IF;

IF SIZEOF(doc) = 0
THEN
     RETURN(TRUE);
ELSE
     IF EXISTS(cl.definition[1]\class.its_superclass)
     THEN
          RETURN(applicable_documents(cl.definition[1]
               \class.its_superclass, doc));
     ELSE
          RETURN(FALSE);
     END_IF;
END_IF;

END_FUNCTION; -- applicable_documents
```



```
(*
```

11.26.10 Retrieve_documents function

The **retrieve_documents** function collects the set of documents directly associated with a class through a **class_document_relationship**.

EXPRESS specification:

```
*)
FUNCTION retrieve_documents(cl: class_BSU): SET[0:?] OF document_BSU;
-- requires: { SIZEOF(cl.definition) <> 0 }

LOCAL
     s: SET[0:?] OF document_BSU := [];
END_LOCAL;

REPEAT i := 1 TO SIZEOF(cl.definition[1]\class.associated_items);
     IF 'ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
          '.CLASS_DOCUMENT_RELATIONSHIP'
          IN TYPEOF(cl.definition[1]\class.associated_items[i])
     THEN
          s := s+ cl.definition[1]\class.associated_items[i]
               \class_document_relationship.related_tokens;
     END_IF;
END_REPEAT;

RETURN(s);

END_FUNCTION; -- retrieve_documents
(*
```

11.26.11 Makes_reference_outside function

The function **makes_reference_outside** checks if an aggregate of **property_or_data_type_BSU** makes references to **class_BSU**s that do not belong to the **l** parameter. It returns FALSE if all the references are in the **l** parameter and TRUE if some references are outside the **l** parameter.

A **property_BSU** or a **data_type_BSU** refers to a **class_BSU** when:

— its **name_scope** is this **class_BSU**, or

— its definition **dictionary_element** is provided and refers, as its domain, to a **class_BSU** (*is-part-of* relationship), or

— its definition **dictionary_element** is provided and refers, as its domain, to another **data_type_BSU** that refers to a **class_BSU** (recursive definition);

Such references are checked by the **makes_reference_outside** function.

EXPRESS specification:

```
*)
FUNCTION makes_reference_outside (
```

```
        p: AGGREGATE OF property_or_data_type_BSU;
        l: LIST[1:?] OF class_BSU): BOOLEAN;

    LOCAL
        bool: BOOLEAN := FALSE;
        temp: SET[0:1] OF class_BSU := [];
    END_LOCAL;

    REPEAT j := 1 TO SIZEOF(p);
        IF ((('ISO13584_IEC61360_DICTIONARY_SCHEMA.PROPERTY_BSU'
            IN TYPEOF(p[j]))
            AND (NOT(p[j]\property_bsu.name_scope IN l)))
            OR
            (('ISO13584_IEC61360_DICTIONARY_SCHEMA.DATA_TYPE_BSU'
            IN TYPEOF(p[j]))
            AND (NOT(p[j]\data_type_bsu.name_scope IN l))))
        THEN
            bool := TRUE;
            RETURN(bool);
        END_IF;

        IF ((('ISO13584_IEC61360_DICTIONARY_SCHEMA.PROPERTY_BSU'
            IN TYPEOF(p[j]))
            AND (NOT(SIZEOF(p[j]\basic_semantic_unit.definition) = 0)))
            OR
            (('ISO13584_IEC61360_DICTIONARY_SCHEMA.DATA_TYPE_BSU'
            IN TYPEOF(p[j]))
            AND (NOT(SIZEOF(p[j]\basic_semantic_unit.definition) = 0))))
        THEN
            IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_INSTANCE_TYPE'
                IN data_type_typeof(p[j]))
            THEN
                temp := data_type_class_of(p[j]);
                IF NOT(temp[1] IN l)
                THEN
                    bool := bool OR TRUE ;
                ELSE
                    bool := bool OR FALSE;
                END_IF;
            END_IF;

            IF SIZEOF(data_type_named_type(p[j])) = 1
            THEN
                bool := bool OR makes_reference_outside
                    (data_type_named_type(p[j]), l);
            END_IF;
        END_IF;
    END_REPEAT;

    RETURN(bool);
```



```
END_FUNCTION; -- makes_reference_outside
(*
```

11.26.12 Prefix_ordered_class_list function

The **prefix_ordered_class_list** function checks that all the direct or indirect references from one **class_BSU** to another **class_BSU** are backward references with respect to the classes list order.

A **class_BSU** is referred to directly by a **class** through the following attributes:

— **class.its_superclass**: is-a relationship;

and, if the class is defined by means of an **a_priori_semantic_relationship**:

— **class\a_priori_semantic_relationship.referenced_classes**.

A **class_BSU** is referred to indirectly by a **class**:

— either when a **property_BSU** or a **data_type_BSU** referenced by this **class** references itself directly or indirectly this **class_BSU**, or

— when a **class_BSU_relationship** referring to this **class** by its **relating_class** attribute refers to the **class_BSU** by its **related_tokens** attributes.

A **property_BSU** or a **data_type_BSU** refers to a **class_BSU** when:

— its **name_scope** is this **class_BSU**, or

— its definition **dictionary_element** is provided and refers, as its domain, to a **class_BSU** (*is-part-of* relationship), or

— its definition **dictionary_element** is provided and refers, as its domain, to another **data_type_BSU** that refers to a **class_BSU** (recursive definition);

Such references are checked by the **makes_reference_outside** function.

In the **ISO13584_extended_dictionary_schema**, a **class_BSU** makes an indirect forward reference to a **class_BSU** with respect to the class list order in five cases.

a) when its **described_by** attribute contains a **property_BSU** that makes a **makes_reference_outside**;

b) or when the class is defined by means of an **a_priori_semantic_relationship**, and its imported properties defined by the **referenced_properties** inherited attribute contain a **property_BSU** that makes a **makes_reference_outside**;

c) or when its **defined_types** attribute contains a **data_type_BSU** that makes a **makes_reference_outside**;

d) or when the class is defined by means of an **a_priori_semantic_relationship**, and its imported data types defined by the **referenced_data_types** inherited attribute contain a **data_type_BSU** that makes a **makes_reference_outside**;

e) or when its associated_items attribute contains a class_BSU_relationship whose **related_tokens** makes a **makes_reference_outside**;

In the context of the **ISO13584_extended_dictionary_schema**, this situation may happen in two cases:

1) the **class_BSU_relationship** is a **class_table_relationship** that contains a **table_BSU** that makes through its **name_scope** attribute a **makes_reference_outside**, or

2) the **class_BSU_relationship** is a **class_document_relationship** that contains a **document_BSU** that makes through its **name_scope** attributes a **makes_reference_outside**.

EXPRESS specification:

```
*)
FUNCTION prefix_ordered_class_list(classes: LIST[2:?] OF class_BSU):
     BOOLEAN;

LOCAL
     related_token: class_related_BSU;-- items associated with
          -- a class through a class_BSU_relationship
END_LOCAL;

REPEAT i := 1 TO SIZEOF(classes);

     IF SIZEOF(classes[i].definition) = 1
     THEN

          IF (EXISTS(classes[i].definition[1]\class.its_superclass))
               AND (NOT((classes[i].definition[1]\class.its_superclass)
               IN (makes_sub_list(classes, 1, i - 1))))
          THEN
               RETURN(FALSE);
          END_IF;

          IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA'+
               '.A_PRIORI_SEMANTIC_RELATIONSHIP'
               IN TYPEOF(classes[i].definition[1]))
          THEN
               IF (QUERY(x <* classes[i].definition[1]
               \a_priori_semantic_relationship.
               referenced_classes | NOT(x IN makes_sub_list(
               classes, 1, i - 1))) <> [])
               THEN
                    RETURN(FALSE);
               END_IF;
          END_IF;

          IF NOT(SIZEOF(classes[i].definition[1]\class.described_by)
               = 0)
          THEN
               IF (makes_reference_outside(classes[i].definition[1]
                    \class.described_by, makes_sub_list(
                    classes, 1, i)))
               THEN
                    RETURN(FALSE);
```



```
     END_IF;
END_IF;

IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA'+
     '.A_PRIORI_SEMANTIC_RELATIONSHIP'
     IN TYPEOF(classes[i].definition[1]))
THEN
     IF makes_reference_outside(classes[i].definition[1]\
     a_priori_semantic_relationship.referenced_properties,
     makes_sub_list(classes, 1, i - 1))
     THEN
          RETURN(FALSE);
     END_IF;
END_IF;

IF NOT(SIZEOF(classes[i].definition[1]\class.defined_types)
     = 0)
THEN
     IF makes_reference_outside(classes[i].definition[1]\
          class.defined_types, makes_sub_list
          (classes, 1, i))
     THEN
          RETURN(FALSE);
     END_IF;
END_IF;

IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA'+
     '.A_PRIORI_SEMANTIC_RELATIONSHIP'
     IN TYPEOF(classes[i].definition[1]))
THEN
     IF makes_reference_outside(classes[i].definition[1]\
     a_priori_semantic_relationship.referenced_data_types,
     makes_sub_list(classes, 1, i - 1))
     THEN
          RETURN(FALSE);
     END_IF;
END_IF;

IF NOT(SIZEOF(classes[i].definition[1]\
     class.associated_items) = 0)
THEN
     REPEAT j := 1 TO SIZEOF(classes[i].definition[1]
     \class.associated_items);
          REPEAT k := 1 TO SIZEOF(classes[i].definition[1]
          \class.associated_items[j]
          \class_BSU_relationship.related_tokens);

               related_token := classes[i].definition[1]
                    \class.associated_items[j]
                    \class_BSU_relationship.related_tokens[k];

               IF (('ISO13584_EXTENDED_DICTIONARY_SCHEMA'+
```

```
                              '.TABLE_BSU') IN (TYPEOF(related_token)))
                              AND NOT(related_token\table_BSU.name_scope
                              IN makes_sub_list(classes, 1, i))
                         THEN
                              RETURN(FALSE);
                         END_IF;

                         IF (('ISO13584_EXTENDED_DICTIONARY_SCHEMA'+
                              '.DOCUMENT_BSU')
                              IN (TYPEOF(related_token)))
                              AND NOT(related_token\document_BSU
                              .name_scope IN makes_sub_list
                              (classes, 1, i))
                         THEN
                              RETURN(FALSE);
                         END_IF;
                    END_REPEAT;
               END_REPEAT;
          END_IF;
     END_IF;
END_REPEAT;

RETURN(TRUE);
END_FUNCTION; -- prefix_ordered_class_list
(*
```

11.26.13 Functional_view_v_c_v function

The **functional_view_v_c_v** function computes the list of properties defined as **view_control_variables** in a class, or any of its super-class(es) by a traversal of the inheritance tree defined by the class hierarchy. It calls the **retrieve_functional_view_v_c_v** function that computes recursively the properties defined as **view_control_variables**.

This function is intended to be called after the **all_class_descriptions_reachable** function. Therefore, if some **dictionary_element**s are not available, it returns an empty LIST.

If the **class_BSU cl** does not refer to a functional view class, it returns the empty set.

EXPRESS specification:

```
*)
FUNCTION functional_view_v_c_v(cl: class_BSU): LIST OF property_BSU;

IF NOT EXISTS(cl)
THEN
     RETURN([]); -- cl is indeterminate
END_IF;

IF NOT(all_class_descriptions_reachable(cl))
THEN
     RETURN([]); -- some dictionary_element are not available
END_IF;
```



```
IF NOT('ISO13584_EXTENDED_DICTIONARY_SCHEMA.FUNCTIONAL_VIEW_CLASS' IN
     TYPEOF(cl.definition[1]))
THEN
     RETURN([]);
END_IF;

RETURN(retrieve_functional_view_v_c_v(cl, []));

END_FUNCTION; -- functional_view_v_c_v
(*
```

11.26.14 Retrieve_functional_view_v_c_v function

The **retrieve_functional_view_v_c_v** function computes the list of properties defined as **view_control_variable**s in a class, or any of its super-classes by a traversal of the inheritance tree defined by the class hierarchy.

If the **class_BSU cl** does not refer to a functional view class, it returns the empty set.

EXPRESS specification:

```
*)
FUNCTION retrieve_functional_view_v_c_v(cl: class_BSU;
     pr: LIST OF property_BSU): LIST OF property_BSU;

LOCAL
     prop: LIST OF property_BSU := pr;
END_LOCAL;

IF SIZEOF(cl.definition) = 0 -- abnormal case
     THEN RETURN([]);
END_IF;

IF NOT('ISO13584_EXTENDED_DICTIONARY_SCHEMA.FUNCTIONAL_VIEW_CLASS' IN
     TYPEOF(cl.definition[1]))
THEN
     RETURN([]);
END_IF;

IF SIZEOF(cl.definition) = 1
THEN
     prop := prop + cl.definition[1]
          \functional_view_class.view_control_variables;
                    -- view_control_variables of this class
END_IF;

IF NOT(EXISTS(cl.definition[1]\class.its_superclass))
THEN
     RETURN(prop);
ELSE
     RETURN(retrieve_functional_view_v_c_v(
          cl.definition[1]\class.its_superclass, prop));
```

```
END_IF;

END_FUNCTION; -- retrieve_functional_view_v_c_v
(*
```

11.26.15 Data_type_named_type function

The **data_type_named_type** function computes the **named_type** used to specify the domain of a **property_BSU** or a **data_type_BSU**. It returns only the first **data_type_BSU** used in the domain definition.

If the **data_type** is not associated with a **named_type**, or if the definition of the parameter **type_spec** is not available, the function returns an empty set.

EXPRESS specification:

```
*)
FUNCTION data_type_named_type(type_spec: property_or_data_type_BSU):
     SET[0:1] OF data_type_BSU;

LOCAL
     res: BOOLEAN := FALSE;
     s: SET[0:1] OF data_type_BSU := [];
     x: data_type;
END_LOCAL;

IF NOT EXISTS(type_spec)
THEN
     RETURN([]); -- type_spec is indeterminate
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.PROPERTY_BSU' IN
     TYPEOF(type_spec))
THEN
     IF NOT(SIZEOF(type_spec.definition) = 0)
     THEN
          x := type_spec.definition[1]\property_DET.domain;
          res := TRUE;
     END_IF;
ELSE
     IF NOT(SIZEOF(type_spec.definition) = 0)
     THEN
          x := type_spec.definition[1]\data_type_element
               .type_definition;
          res := TRUE;
     END_IF;
END_IF;

IF res
THEN
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NAMED_TYPE'
          IN TYPEOF(x))
```



```
      THEN
           s := s + x\named_type.referred_type;
      END_IF;
END_IF;

RETURN(s);

END_FUNCTION; -- data_type_named_type
(*
```

11.26.16 Data_type_typeof function

The **data_type_typeof** function runs as the EXPRESS TYPEOF function concerning the **data_type** that defines the final domain of a **property_BSU** or a **data_type_BSU**.

If the **data_type** is associated with **named_type**s, the function recursively traverses their **referred_types** attributes, until arriving either to a **simple_type** or to a **complex_type**. Then the function returns the result of the EXPRESS TYPEOF function applied to this entity.

If some BSU definitions are not available, with the result that the function cannot be resolved to a **simple_type** or to a **complex_type**, the function returns an empty set of STRING.

EXPRESS specification:

```
*)
FUNCTION data_type_typeof(type_spec: property_or_data_type_BSU):
      SET OF STRING;

LOCAL
      res: BOOLEAN := FALSE;
      x: data_type;
END_LOCAL;

IF NOT EXISTS(type_spec)
THEN
      RETURN([]); -- type_spec is indeterminate
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.PROPERTY_BSU' IN
      TYPEOF(type_spec))
THEN
      IF NOT(SIZEOF(type_spec.definition) = 0)
      THEN
           x := type_spec.definition[1]\property_DET.domain;
           res := TRUE;
      END_IF;
ELSE
      IF NOT(SIZEOF(type_spec.definition) = 0)
      THEN
           x := type_spec.definition[1]
                \data_type_element.type_definition;
           res := TRUE;
      END_IF;
```

```
END_IF;

IF NOT(res)
THEN
     RETURN([]);
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NAMED_TYPE' IN TYPEOF(x))
THEN
     IF NOT(SIZEOF(x\named_type.referred_type.definition) = 0)
     THEN
          RETURN(data_type_typeof(x\named_type.referred_type));
     ELSE
          RETURN([]);
     END_IF;
ELSE
     RETURN(TYPEOF(x));
END_IF;

END_FUNCTION; -- data_type_typeof
(*
```

11.26.17 Data_type_class_of function

The **data_type_class_of** function computes the class that defines the final domain of a **property_BSU** or a **data_type_BSU**. This function is intended to be called after the **data_type_typeof** function.

If the **data_type** is not of type **class_instance_type**, or if a **data_type** is associated with a **named_type**, of which BSU definition is not available, the function returns an empty set.

EXPRESS specification:

```
*)
FUNCTION data_type_class_of(type_spec: property_or_data_type_BSU):
     SET[0:1] OF class_BSU;

LOCAL
     res: BOOLEAN := FALSE;
     s: SET[0:1] OF class_BSU := [];
     x: data_type;
END_LOCAL;

IF NOT EXISTS(type_spec)
THEN
     RETURN([]); -- type_spec is indeterminate
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.PROPERTY_BSU' IN
     TYPEOF(type_spec))
THEN
     IF NOT(SIZEOF(type_spec.definition) = 0)
```



```
      THEN
           x := type_spec.definition[1]\property_DET.domain;
           res := TRUE;
      END_IF;
ELSE
      IF NOT(SIZEOF(type_spec.definition) = 0)
      THEN
           x := type_spec.definition[1]\data_type_element
                .type_definition;
           res := TRUE;
      END_IF;
END_IF;

IF res
THEN
      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_INSTANCE_TYPE' IN
           TYPEOF(x))
      THEN
           s := s + x\class_instance_type.domain;
           RETURN(s);
      END_IF;

      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NAMED_TYPE'
           IN TYPEOF(x))
      THEN
           s := data_type_class_of(x\named_type.referred_type);
           RETURN(s);
      END_IF;
END_IF;

RETURN(s);

END_FUNCTION; -- data_type_class_of
(*
```

11.26.18 Data_type_type_name function

The **data_type_type_name** function computes the **type_name** attributes of the **entity_instance_type** that defines the final domain of a **property_BSU** or a **data_type_BSU**. This function is intended to be called after the **data_type_typeof** function.

If the **data_type** is not of type **entity_instance_type**, or if a **data_type** is associated with a **named_type**, of which BSU definition is not available, the function returns an empty set.

EXPRESS specification:

```
*)
FUNCTION data_type_type_name(t: property_or_data_type_BSU):
      SET [0:?] OF STRING;

LOCAL
      res: BOOLEAN := FALSE;
      s: SET [0:1] OF STRING := [];
```



```
     x: data_type;
END_LOCAL;

IF NOT EXISTS(t)
THEN
     RETURN([]); -- t is indeterminate
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.PROPERTY_BSU' IN TYPEOF(t))
THEN
     IF NOT(SIZEOF(t.definition) = 0)
     THEN
          x := t.definition[1]\property_DET.domain;
          res := TRUE;
     END_IF;
ELSE
     IF NOT(SIZEOF(t.definition) = 0)
     THEN
          x := t.definition[1]\data_type_element.type_definition;
          res := TRUE;
     END_IF;
END_IF;

IF res
THEN
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.ENTITY_INSTANCE_TYPE'
          IN TYPEOF(x))
     THEN
          s := x\entity_instance_type.type_name;
     END_IF;

     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NAMED_TYPE'
          IN TYPEOF(x))
     THEN
          s := data_type_type_name(x\named_type.referred_type);
     END_IF;
END_IF;

RETURN(s);

END_FUNCTION; -- data_type_type_name
(*
```

11.26.19 Data_type_level_spec function

The **data_type_level_spec** function computes the **levels** attribute of the **level_type** that defines the final domain of a **property_BSU** or a **data_type_BSU**. This function is intended to be called after the **data_type_typeof** function.

If the **data_type** is not of type **level_type**, or if some **data_type** is associated with a **named_type**, of which BSU definition is not available, the function returns an empty LIST.

EXPRESS specification:

```
*)
FUNCTION data_type_level_spec(t: property_or_data_type_BSU):
     LIST[0:4] OF UNIQUE Level;

LOCAL
     res: BOOLEAN := FALSE;
     s: LIST[0:4] OF UNIQUE level := [];
     x: data_type;
END_LOCAL;

IF NOT EXISTS(t)
THEN
     RETURN([]); -- t is indeterminate
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.PROPERTY_BSU' IN TYPEOF(t))
THEN
     IF NOT(SIZEOF(t.definition) = 0)
     THEN
          x := t.definition[1]\property_DET.domain;
          res := TRUE;
     END_IF;
ELSE
     IF NOT(SIZEOF(t.definition) = 0)
     THEN
          x := t.definition[1]\data_type_element.type_definition;
          res := TRUE;
     END_IF;
END_IF;

IF res
THEN
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.LEVEL_TYPE'
          IN TYPEOF(x))
     THEN
          s := x\level_type.levels;
     END_IF;

     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NAMED_TYPE'
          IN TYPEOF(x))
     THEN
          s := data_type_level_spec(x\named_type.referred_type);
     END_IF;
END_IF;

RETURN(s);

END_FUNCTION; -- data_type_level_spec
(*
```

11.26.20 Data_type_level_value_typeof function

The **data_type_level_value_type** of function runs like the EXPRESS TYPEOF function applied to the **value_type** attribute of the **level_type** that defines the final domain of a **property_BSU** or a **data_type_BSU**. This function is intended to be called after the **data_type_typeof** function.

If the **data_type** is not of type **level_type**, or if a **data_type** is associated with a **named_type**, of which BSU definition is not available, the function returns an empty set of string.

EXPRESS specification:

```
*)
FUNCTION data_type_level_value_typeof(t: property_or_data_type_BSU):
     SET OF STRING;

LOCAL
     res: BOOLEAN := FALSE;
     s: SET OF STRING := [];
     x: data_type;
END_LOCAL;

IF NOT EXISTS(t)
THEN
     RETURN([]); -- t is indeterminate
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.PROPERTY_BSU' IN TYPEOF(t))
THEN
     IF NOT(SIZEOF(t.definition) = 0)
     THEN
          x := t.definition[1]\property_DET.domain;
          res := TRUE;
     END_IF;
ELSE
     IF NOT(SIZEOF(t.definition) = 0)
     THEN
          x := t.definition[1]\data_type_element.type_definition;
          res := TRUE;
     END_IF;
END_IF;

IF res
THEN
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.LEVEL_TYPE'
          IN TYPEOF(x))
     THEN
          s := TYPEOF(x\level_type.value_type);
     END_IF;

     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NAMED_TYPE'
          IN TYPEOF(x))
     THEN
```



```
            s := data_type_level_value_typeof(
                 x\named_type.referred_type);
      END_IF;

END_IF;

RETURN(s);

END_FUNCTION; -- data_type_level_value_typeof
(*
```

11.26.21 Simple_type_data_type function

The **simple_type_data_type** function checks if the final domain of a **property_BSU** or a **data_type_BSU** corresponds to a **simple_type**.

NOTE Simple type means: integer, real, string or Boolean, or their subtypes.

If a **data_type** is associated with a **named_type**, of which BSU definition is not available, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION simple_type_data_type(type_spec: property_or_data_type_BSU):
      LOGICAL;

IF NOT EXISTS(type_spec)
THEN
      RETURN(UNKNOWN); -- type_spec is indeterminate
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.SIMPLE_TYPE' IN
      data_type_typeof(type_spec))
THEN
      RETURN(TRUE);
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.COMPLEX_TYPE' IN
      data_type_typeof(type_spec))
THEN
      RETURN(FALSE);
END_IF;

RETURN(UNKNOWN);

END_FUNCTION; -- simple_type_data_type
(*
```

11.26.22 Complex_type_data_type function

The **complex_type_data_type** function checks if the final domain of a **property_BSU** or a **data_type_BSU** corresponds to a **complex_type**.

NOTE Complex type means: class instance, item instance or level specification.

If a **data_type** is associated with a **named_type**, of which BSU definition is not available, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION complex_type_data_type(type_spec: property_or_data_type_BSU):
     LOGICAL;

IF NOT EXISTS(type_spec)
THEN
     RETURN(UNKNOWN); -- type_spec is indeterminate
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.COMPLEX_TYPE'
     IN data_type_typeof(type_spec))
THEN
     RETURN(TRUE);
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.SIMPLE_TYPE' IN
     data_type_typeof(type_spec))
THEN
     RETURN(FALSE);
END_IF;

RETURN(UNKNOWN);

END_FUNCTION; -- complex_type_data_type
(*
```

11.26.23 Compatible_subclass function

The **compatible_subclass** function returns TRUE if **c2** is a subclass of **c1**, or if **c2** is a subclass of a previous version of **c1**, i.e., a subclass of a class that has the same code as **c1**, the same supplier as **c1** and a version less or equal to the version of **c1**. Otherwise, it returns FALSE.

If the result may not be computed because a BSU definition is not available, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION compatible_subclass(c1, c2: class_BSU): LOGICAL;

IF (NOT EXISTS(c1)) OR (NOT EXISTS(c2))
THEN
     RETURN(UNKNOWN); -- c1 or c2 indeterminate
END_IF;
```



```
IF c1 = c2
THEN
     RETURN(TRUE);
END_IF;

IF ((c1.code = c2.code) AND (c1.version >= c2.version)
     AND (c1.defined_by.code = c2.defined_by.code))
THEN
     RETURN(TRUE);
END_IF;

IF SIZEOF(c2.definition) = 0
THEN
     RETURN(UNKNOWN);
ELSE
     IF EXISTS(c2.definition[1]\class.its_superclass)
     THEN
          RETURN(compatible_subclass(c1, c2.definition[1]
               \class.its_superclass));
     ELSE
          RETURN(FALSE);
     END_IF;
END_IF;

END_FUNCTION; -- compatible_subclass
(*
```

11.26.24 Compatible_types function

The function **compatible_types** checks if the two final domains of **property_BSU**s or **data_type_BSU**s whose types are defined by the **ISO13584_IEC61360_dictionary_schema** are such that the second type is assignment compatible with the first one.

A type **p2** is assignment compatible with a type **p1** when one of the following condition holds:

— **p1** = **p2**;

— the EXPRESS TYPEOF function applied to both entities provides the same result, and **p2** has the same code as **p1**, the same supplier code as **p1** and a version number less or equal to the version number of **p1**;

— **p1** and **p2** have the same **data_type_element**;

— the result of the function **data_type_typeof** applied to **p1** is not empty and is included in the result of the function **data_type_typeof** applied to **p2**, and this latter result does not contain **'ISO13584_IEC61360_DICTIONARY_SCHEMA**.**COMPLEX_TYPE'**;

— the results of the function **data_type_typeof** applied to both types contains **'ISO13584_IEC61360_DICTIONARY_SCHEMA**.**CLASS_INSTANCE_TYPE'** and the class corresponding to **p2** is a **compatible_subclass** of the class corresponding to **p1**, i.e., is a subclass of **p1** or is a subclass of a class that has the same supplier code, and the same class code as **p1**, and a class version less or equal to the version that corresponds to **p1**.

— the result of the function **data_type_typeof** applied to both types contains **'ISO13584_IEC61360_DICTIONARY_SCHEMA**.**ENTITY_INSTANCE_TYPE'** and the **type_name** attribute of **p1** is not empty and is contained in the **type_name** attribute of **p2**;

— the result of the function **data_type_typeof** applied to both types contains **'ISO13584_IEC61360_DICTIONARY_SCHEMA**.**LEVEL_TYPE'** and the **levels** attributes of both types are not empty and have the same (unordered) content, and the result of the function **data_type_level_value_typeof** applied to **p1** is not empty and is contained in the result of the function **data_type_level_value_typeof** applied to **p2**.

If the result may not be computed because some BSU definitions are not available, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION compatible_types(p1: property_or_data_type_BSU;
     p2: property_or_data_type_BSU): LOGICAL;

LOCAL
     p1_domain, p2_domain: data_type;
END_LOCAL;

IF (NOT EXISTS(p1)) OR (NOT EXISTS(p2))
THEN
     RETURN(UNKNOWN); -- p1 or p2 indeterminate
END_IF;

(* case 1 *)

IF p1 = p2
THEN
     RETURN(TRUE);
END_IF;

(* case 2 *)

IF ((TYPEOF(p1) = TYPEOF(p2))
     AND (p1\basic_semantic_unit.code = p2.code)
     AND (p1.name_scope\basic_semantic_unit.code =
          p2.name_scope\basic_semantic_unit.code)
     AND (p1\basic_semantic_unit.version >=
          p2\basic_semantic_unit.version))
THEN
     RETURN(TRUE);
END_IF;

(* case 8 *)

IF (SIZEOF(p1\basic_semantic_unit.definition) = 0)
     OR (SIZEOF(p2\basic_semantic_unit.definition) = 0)
THEN
```



```
      RETURN(UNKNOWN);
ELSE
      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.PROPERTY_BSU'
           IN TYPEOF(p1))
      THEN
           p1_domain := p1.definition [1]\property_DET.domain;
      ELSE
           p1_domain := p1.definition[1]\data_type_element
                .type_definition;
      END_IF;

      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.PROPERTY_BSU'
           IN TYPEOF(p2))
      THEN
           p2_domain := p2.definition [1]\property_DET.domain;
      ELSE
           p2_domain := p2.definition[1]\data_type_element
                .type_definition;
      END_IF;
END_IF;

(* case 3 *)

IF p1_domain = p2_domain
THEN
      RETURN(TRUE);
END_IF;

(* case 4 *)

IF (NOT(SIZEOF(data_type_typeof(p1)) = 0))
      AND (data_type_typeof(p1)<= data_type_typeof(p2))
      AND (NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA.COMPLEX_TYPE' IN
      data_type_typeof(p2)))
THEN
      RETURN(TRUE);
END_IF;

(* case 5 *)

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_INSTANCE_TYPE' IN
      data_type_typeof(p1))
      AND ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_INSTANCE_TYPE' IN
      data_type_typeof(p2))
THEN
      RETURN(compatible_subclass(data_type_class_of(p1)[1],
           data_type_class_of(p2)[1]));
END_IF;

(* case 6 *)

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.ENTITY_INSTANCE_TYPE' IN
```

```
      data_type_typeof(p1))
      AND ('ISO13584_IEC61360_DICTIONARY_SCHEMA.ENTITY_INSTANCE_TYPE'
      IN data_type_typeof(p2))
THEN
      IF ((data_type_type_name(p1) <= data_type_type_name(p2))
           AND (data_type_type_name(p1) <> []))
      THEN
           RETURN(TRUE);
      ELSE
           RETURN(FALSE);
      END_IF;
END_IF;

(* case 7 *)

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.LEVEL_TYPE' IN
      data_type_typeof(p1))
      AND ('ISO13584_IEC61360_DICTIONARY_SCHEMA.LEVEL_TYPE' IN
      data_type_typeof(p2))
THEN
      IF (data_type_level_spec(p1) = data_type_level_spec(p2))
           AND (data_type_level_value_typeof(p1) <> [])
           AND (data_type_level_value_typeof(p1)
           <= data_type_level_value_typeof(p2))
      THEN
           RETURN(TRUE);
      ELSE
           RETURN(FALSE);
      END_IF;
END_IF;

(* case 8 *)

IF ((data_type_typeof(p1) = []) OR (data_type_typeof(p2) = []))
THEN
      RETURN(UNKNOWN);
ELSE
      RETURN(FALSE);
END_IF;

END_FUNCTION; -- compatible_types
(*
```

11.26.25 Ordered_index_value function

The function **ordered_index_value** checks that the values of a **non_quantitative_int_type** are successive integers.

EXPRESS specification:

```
*)
FUNCTION ordered_index_value(x: value_domain): BOOLEAN;

REPEAT i := LOBOUND(x.its_values) TO SIZEOF(x.its_values);
     IF x.its_values[i].value_code <> x.its_values[i-1].value_code + 1
     THEN
          RETURN(FALSE);
     END_IF;
END_REPEAT;

RETURN(TRUE);

END_FUNCTION; -- ordered_index_value
(*
```

11.26.26 Makes_sub_list

The **makes_sub_list** function builds a sublist from a subset of another list **cla** of **class_BSU**s. It corresponds to **cla**[i..j].

EXPRESS specification:

```
*)
FUNCTION makes_sub_list(cla: LIST [1:?] OF class_BSU;
     i, j: INTEGER): LIST [1:?] OF class_BSU;

LOCAL
     c: LIST [0:?] OF class_BSU := [];
END_LOCAL;

REPEAT k := i TO j;
     c := c + cla[k];
END_REPEAT;

RETURN(c);

END_FUNCTION; -- makes_sub_list
(*
```

11.26.27 Sub_list_until

The **sub_list_until** function builds a sublist from another list (**cla**) of **class_BSU**s. This sublist consists of the beginning of the **cla** list up to the **cl** class. If **cl** does not belong to the **cla** list, the whole **cla** list is returned.

EXPRESS specification:

```
*)
FUNCTION sub_list_until(cla: LIST [1:?] OF class_BSU;
     cl: class_BSU): LIST [1:?] OF class_BSU;
```

```
LOCAL
     c: LIST [0:?] OF class_BSU := [];
END_LOCAL;

REPEAT k := 1 TO SIZEOF(cla);
     c := c + cla[k];
     IF cla[k] :=: cl
     THEN
          ESCAPE;
     END_IF;
END_REPEAT;

RETURN(c);

END_FUNCTION; -- sub_list_until
(*
```

11.26.28 Get_property_BSU_from_property_semantics function

The **get_property_BSU_from_property_semantics** function returns the list of **property_BSU**s associated to a list of **property_semantics**.

EXPRESS specification:

```
*)
FUNCTION get_property_BSU_from_property_semantics(
     l: AGGREGATE OF variable_semantics): LIST[1:?] OF property_BSU;

LOCAL
     res: LIST[0:?] OF property_BSU := [];
END_LOCAL;

REPEAT i := 1 TO SIZEOF(l);
     res := res + l[i]\property_semantics.the_property;
END_REPEAT;

RETURN(res);

END_FUNCTION; -- get_property_BSU_from_property_semantics
(*
```

11.26.29 Compatible_list_library_types_and_columns function

The function **compatible_list_library_types_and_columns** checks if a list of **column**s (**col**) is type compatible with a list of types (**dom**) defined by a list of **property_or_data_type_BSU**s.

Simple types are, like EXPRESS, not strongly typed: for instance, a real value may be assigned to a measure.

Complex types support, like EXPRESS, inheritance. Due to the meta-modelling approach used in ISO 13584, inheritance occurs at two levels:

— as EXPRESS-encoded: an **entity_instance_column** that declares, through its **type_name** attribute, to contain **placement**s may contain, for instance, **axis1_placement**, and

— at the meta-level: assuming that A is a superclass of B, and that B is a superclass of C, a column referring to B may contain instances of C and may be assigned to a property whose type is defined by A.

The function returns a logical that is TRUE when all the types are compatible and FALSE when they are not. This function returns UNKNOWN when some required **basic_semantic_unit** definitions are not present.

EXPRESS specifications:

```
*)
FUNCTION compatible_list_library_types_and_columns(
     dom: LIST [0:?] of property_or_data_type_BSU;
     col: LIST [0:?] of column): LOGICAL;

LOCAL
     res: LOGICAL := TRUE;
     set_string: SET OF STRING := [];
     set_integer: SET OF INTEGER := [];
     code_type: non_quantitative_code_type;
     int_type: non_quantitative_int_type;
END_LOCAL;

IF SIZEOF(dom) <> SIZEOF(col)
THEN
     RETURN(FALSE);
END_IF;

REPEAT i := LOINDEX(col) TO HIINDEX(col);
     IF data_type_typeof(dom[i]) = []
     THEN (* the final domain of one type is not available *)
          res := UNKNOWN;
     ELSE
          IF ('ISO13584_TABLE_RESOURCE_SCHEMA.INTEGER_COLUMN'
               IN TYPEOF(col[i]))
          THEN
               IF ((('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
                    '.NON_QUANTITATIVE_INT_TYPE') IN
                    data_type_typeof(dom[i]))
                    AND (SIZEOF(data_type_non_quantitative_int_type(
                    dom[i])) = 1))
               THEN
                    set_integer := [];
                    int_type := data_type_non_quantitative_int_type(
                         dom[i])[1];
                    REPEAT j :=1 TO SIZEOF(int_type.domain.its_values);
                         set_integer := set_integer
                              + int_type.domain.its_values[j]
                              .value_code;
                    END_REPEAT;
```



```
                REPEAT j := 1 TO SIZEOF(col[i].values);
                     IF (('INTEGER' IN TYPEOF(col[i].values[j]))
                          AND NOT (col[i].values[j] IN set_integer))
                     THEN
                          RETURN (FALSE);
                     END_IF;
                END_REPEAT;
           ELSE
                IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_TYPE'
                     IN data_type_typeof(dom[i]))
                     OR (('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
                     '.NUMBER_TYPE' IN data_type_typeof(dom[i]))
                     AND NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA'
                     + '.REAL_TYPE' IN data_type_typeof(dom[i])))
                THEN
                     ;
                ELSE
                     RETURN(FALSE);
                END_IF;
           END_IF;
      END_IF;

      IF ('ISO13584_TABLE_RESOURCE_SCHEMA.REAL_COLUMN'
           IN TYPEOF(col[i]))
      THEN
           IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE'
                IN data_type_typeof(dom[i]))
                OR (('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
                '.NUMBER_TYPE' IN data_type_typeof(dom[i]))
                AND NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
                '.INT_TYPE' IN data_type_typeof(dom[i])))
           THEN
                ;
           ELSE
                RETURN(FALSE);
           END_IF;
      END_IF;

      IF ('ISO13584_TABLE_RESOURCE_SCHEMA.BOOLEAN_COLUMN'
           IN TYPEOF(col[i]))
      THEN
           IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.BOOLEAN_TYPE'
                IN data_type_typeof(dom[i]))
           THEN
                ;
           ELSE
                RETURN(FALSE);
           END_IF;
      END_IF;

      IF ('ISO13584_TABLE_RESOURCE_SCHEMA.STRING_COLUMN'
```



```
     IN TYPEOF(col[i]))
THEN
     IF ((('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
          '.NON_QUANTITATIVE_CODE_TYPE') IN
          data_type_typeof(dom[i]))
          AND (SIZEOF(data_type_non_quantitative_code_type(
          dom[i])) = 1))
     THEN
          set_string := [];
          code_type := data_type_non_quantitative_code_type(
               dom[i])[1];
          REPEAT j:=1 TO SIZEOF(code_type.domain.its_values);
               set_string := set_string
                    + code_type.domain.its_values[j]
                    .value_code;
          END_REPEAT;
          REPEAT j := 1 TO SIZEOF(col[i].values);
               IF ((('STRING' IN TYPEOF(col[i].values[j]))
                    AND NOT (col[i].values[j] IN set_string))
                    OR ('ISO13584_INSTANCE_RESOURCE_SCHEMA'
                    + '.TRANSLATED_STRING_VALUE' IN
                    TYPEOF(col[i].values[j])))
               THEN
                    RETURN (FALSE);
               END_IF;
          END_REPEAT;
     ELSE
          IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
               '.STRING_TYPE' IN data_type_typeof(dom[i]))
          THEN
               ;
          ELSE
               RETURN(FALSE);
          END_IF;
     END_IF;
END_IF;

IF ('ISO13584_TABLE_RESOURCE_SCHEMA.ENTITY_INSTANCE_COLUMN'
     IN TYPEOF(col[i]))
THEN
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
          '.ENTITY_INSTANCE_TYPE' IN
          data_type_typeof(dom[i]))
          AND (data_type_type_name(dom[i]) <= col[i]\
          entity_instance_column.type_name)
          (* column of subtypes are allowed*)
     THEN
          ;
     ELSE
          RETURN(FALSE);
     END_IF;
END_IF;
```

```
            IF ('ISO13584_TABLE_RESOURCE_SCHEMA.CLASS_INSTANCE_COLUMN'
                 IN TYPEOF(col[i]))
            THEN
                 IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
                      '.CLASS_INSTANCE_TYPE' IN data_type_typeof(dom[i]))
                      AND (compatible_subclass(
                      data_type_class_of(dom[i])[1],
                      col[i]\class_instance_column.class_ref))
                 THEN
                      ;
                 ELSE
                      RETURN(FALSE);
                 END_IF;
            END_IF;

            IF ('ISO13584_TABLE_RESOURCE_SCHEMA.LEVEL_SPEC_COLUMN'
                 IN TYPEOF(col[i]))
            THEN
                 IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.LEVEL_TYPE'
                      IN data_type_typeof(dom[i]))
                 THEN
                 (* all values are checked against the specified levels*)
                      REPEAT j := 1 TO SIZEOF(col[i].values);
                           IF NOT(compatible_level_type_and_instance(
                                data_type_level_spec(dom[i]),
                                data_type_level_value_typeof(dom[i]),
                                col[i].values[j]))
                           THEN
                                RETURN(FALSE);
                           END_IF;
                      END_REPEAT;
                 ELSE
                      RETURN(FALSE);
                 END_IF;
            END_IF;
       END_IF;
  END_REPEAT;

  RETURN(res);

  END_FUNCTION; -- compatible_list_library_types_and_columns
  (*
```

11.26.30 Data_type_non_quantitative_int_type function

The **data_type_non_quantitative_int_type** function computes the **non_quantitative_int_type** that defines the final domain of a **property_BSU** or a **data_type_BSU**. This function is intended to be called after the **data_type_typeof** function.

If the **data_type** is not of type **non_quantitative_int_type**, or if a **data_type** is associated with a **named_type**, of which BSU definition is not available, the function returns an empty set.

EXPRESS specification:

```
*)
FUNCTION data_type_non_quantitative_int_type(
     type_spec: property_or_data_type_BSU):
     SET [0:1] OF non_quantitative_int_type;
LOCAL
     res: BOOLEAN := FALSE;
     s: SET [0:1] OF non_quantitative_int_type := [];
     x: data_type;
END_LOCAL;

IF NOT EXISTS(type_spec)
THEN
     RETURN([]); -- type_spec is indeterminate
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.PROPERTY_BSU' IN
     TYPEOF(type_spec))
THEN
     IF NOT(SIZEOF(type_spec.definition) = 0)
     THEN
          x := type_spec.definition[1]\property_DET.domain;
          res := TRUE;
     END_IF;
ELSE
     IF NOT(SIZEOF(type_spec.definition) = 0)
     THEN
          x := type_spec.definition[1]\
          data_type_element.type_definition;
          res := TRUE;
     END_IF;
END_IF;

IF res
THEN
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA'+
          '.NON_QUANTITATIVE_INT_TYPE') IN TYPEOF(x)
     THEN
          s := s + x;
          RETURN(s);
     END_IF;

     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NAMED_TYPE')
          IN TYPEOF(x)
     THEN
          s := data_type_non_quantitative_int_type(
               x\named_type.referred_type);
          RETURN(s);
     END_IF;
END_IF;
```

```
RETURN([]);

END_FUNCTION; -- data_type_non_quantitative_int_type
(*
```

11.26.31 Data_type_non_quantitative_code_type function

The **data_type_non_quantitative_code_type** function computes the **non_quantitative_code_type** that defines the final domain of a **property_BSU** or a **data_type_BSU**. This function is intended to be called after the **data_type_typeof** function.

If the **data_type** is not of type **non_quantitative_code_type**, or if a **data_type** is associated with a **named_type**, of which BSU definition is not available, the function returns an empty set.

EXPRESS specification:

```
*)
FUNCTION data_type_non_quantitative_code_type(
     type_spec: property_or_data_type_BSU):
     SET [0:1] OF non_quantitative_code_type;
LOCAL
     res: BOOLEAN := FALSE;
     s: SET [0:1] OF non_quantitative_code_type := [];
     x: data_type;
END_LOCAL;

IF NOT EXISTS(type_spec)
THEN
     RETURN([]); -- type_spec is indeterminate
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.PROPERTY_BSU' IN
     TYPEOF(type_spec))
THEN
     IF NOT(SIZEOF(type_spec.definition) = 0)
     THEN
          x := type_spec.definition[1]\property_DET.domain;
          res := TRUE;
     END_IF;
ELSE
     IF NOT(SIZEOF(type_spec.definition) = 0)
     THEN
          x := type_spec.definition[1]\
          data_type_element.type_definition;
          res := TRUE;
     END_IF;
END_IF;

IF res
THEN
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA'+
          '.NON_QUANTITATIVE_CODE_TYPE') IN TYPEOF(x)
```



```
      THEN
           s := s + x;
           RETURN(s);
      END_IF;

      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NAMED_TYPE')
           IN TYPEOF(x)
      THEN
           s := data_type_non_quantitative_code_type(
                x\named_type.referred_type);
           RETURN(s);
      END_IF;
END_IF;

RETURN([]);

END_FUNCTION; -- data_type_non_quantitative_code_type
(*
```

11.26.32 Applicable_properties_for_applicable_tables function

The **applicable_properties_for_applicable_tables** function checks that all the properties that are associated with **column**s of tables associated with a class by a **rel class_table_relationship** are applicable to the **rel.relating_class** class.

If some **table_BSU** definition is not available, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION applicable_properties_for_applicable_tables(
      rel: class_table_relationship): LOGICAL;
LOCAL
      table: table_bsu;
      cl: class_bsu;
      props: LIST OF property_bsu := [];
      res: LOGICAL := TRUE;
END_LOCAL;

IF QUERY(table <* rel.related_tokens
      | SIZEOF(table.definition) = 0) <> []
THEN
      RETURN(UNKNOWN);
END_IF;

REPEAT i := 1 TO SIZEOF(rel.related_tokens);
      table := rel.related_tokens[i];
      cl := rel\class_BSU_relationship.relating_class.identified_by;
      props := get_property_BSU_from_property_semantics(
           table\basic_semantic_unit.definition[1]
           \table_element.column_meaning);
      res := res AND applicable_properties(cl, list_to_set(props));
END_REPEAT;
```

```
RETURN(res);

END_FUNCTION; -- visible_properties_for_visible_tables
(*
```

11.26.33 Superclass_of_item_is_item function

The **superclass_of_item_is_item** function checks that the superclass of an **item_class cl**, if it exists, is an **item_class.**

If the **class** associated with a **class_BSU** cannot be computed, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION superclass_of_item_is_item(cl: item_class): LOGICAL;

IF NOT EXISTS(cl\class.its_superclass)
THEN
     RETURN(TRUE);
END_IF;

IF SIZEOF(cl\class.its_superclass.definition) = 0
THEN
     RETURN(UNKNOWN);
END_IF;

RETURN(('ISO13584_IEC61360_DICTIONARY_SCHEMA.ITEM_CLASS')
     IN TYPEOF(cl\class.its_superclass.definition[1]));

END_FUNCTION; -- superclass_of_item_is_item
(*
```

11.26.34 Compatible_content_and_specification function

The **compatible_content_and_specification** function checks that the number of **columns** of the **tab table_content** is correct with respect to the **table_element** description, and that the types and values of each **column** are compatible and correspond to the type defined in the **data_type** specification of the corresponding **property_semantics**.

If the **table_element** associated with a **table_BSU** of **tab** cannot be computed, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION compatible_content_and_specification(
     tab: table_content): LOGICAL;

IF SIZEOF(tab\content_item.dictionary_definition.definition) = 0
THEN
```



```
      RETURN(UNKNOWN);
END_IF;

RETURN(compatible_list_library_types_and_columns(
      get_property_BSU_from_property_semantics(
      tab\content_item.dictionary_definition.definition[1]\
      table_specification.column_meaning),
      tab\table_extension.content));

END_FUNCTION; -- compatible_content_and_specification
(*
```

11.26.35 Check_view_of_instance_datatype function

The **check_view_of_instance_datatype** function checks if the **fm_class_view_of view_of** attribute refers to an **item_class**.

EXPRESS specification:

```
*)
FUNCTION check_view_of_instance_datatype(
      fmc_view_of: fm_class_view_of): LOGICAL;

IF (SIZEOF(fmc_view_of.view_of.definition) = 1)
THEN
      RETURN('ISO13584_IEC61360_DICTIONARY_SCHEMA.ITEM_CLASS'
           IN TYPEOF(fmc_view_of.view_of.definition[1]));
ELSE
      RETURN(UNKNOWN);
END_IF;

END_FUNCTION; -- check_view_of_instance_datatype
(*
```

11.26.36 View_control_variables_attributes_belong_to_domain function

The **view_control_variables_attributes_belong_to_domain** function checks whether the **view_control_variable_range** attributes **range_lobound** and **range_hibound** belong or not belong to the domain of a view control variable.

EXPRESS specification:

```
*)
FUNCTION view_control_variables_attributes_belong_to_domain(
      vcv_range: view_control_variable_range): LOGICAL;

IF (data_type_typeof(vcv_range.parameter_type) <> [])
THEN
      RETURN((data_type_non_quantitative_int_type(
           vcv_range.parameter_type)[1].domain
           .its_values[1].value_code <= vcv_range.range_lobound)
           AND (vcv_range.range_hibound <=
```

```
            data_type_non_quantitative_int_type(
            vcv_range.parameter_type)[1].domain
            .its_values[HIINDEX(data_type_non_quantitative_int_type(
            vcv_range.parameter_type)[1].domain.its_values)].value_code)
      );
ELSE
      RETURN(UNKNOWN);
END_IF;

END_FUNCTION; -- view_control_variables_attributes_belong_to_domain
(*
```

11.26.37 Created_view_is_functional_view function

The **created_view_is_functional_view** function checks that the **cl** corresponding dictionary definition is a **functional_view_class** if and only if this dictionary definition exists in the same exchange context.

EXPRESS specification:

```
*)
FUNCTION created_view_is_functional_view(cl: class_BSU): LOGICAL;

IF (SIZEOF(cl\basic_semantic_unit.definition) = 1) THEN
      RETURN('ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
            '.FUNCTIONAL_VIEW_CLASS'
            IN TYPEOF(cl\basic_semantic_unit.definition[1]));
ELSE
      RETURN(UNKNOWN);
END_IF;

END_FUNCTION; -- created_view_is_functional_view
(*
```

11.26.38 Check_is_case_of_referenced_classes_definition function

The **check_is_case_of_referenced_classes_definition** returns TRUE if the **item_class_case_of is_case_of** set of referenced class dictionary definition(s) is type compatible with the given **cl item_class_case_of** instance. Otherwise, it returns FALSE.

EXPRESS specification:

```
*)
FUNCTION check_is_case_of_referenced_classes_definition(
      cl: item_class_case_of): BOOLEAN;
LOCAL
      class_def_ok: BOOLEAN := TRUE;
      done: BOOLEAN := FALSE;
END_LOCAL;

REPEAT i := 1 TO SIZEOF(cl.is_case_of);
      IF (SIZEOF(cl.is_case_of[i].definition) = 1)
      THEN
```



```
            IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
                 '.COMPONENT_CLASS_CASE_OF' IN TYPEOF(cl))
            THEN
                 IF (NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
                      '.COMPONENT_CLASS'
                      IN TYPEOF(cl.is_case_of[i].definition[1])))
                 THEN
                      class_def_ok := FALSE;
                 END_IF;
                 done := TRUE;
            END_IF;
            IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
                 '.FEATURE_CLASS_CASE_OF' IN TYPEOF(cl))
            THEN
                 IF (NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
                      '.FEATURE_CLASS'
                      IN TYPEOF(cl.is_case_of[i].definition[1])))
                 THEN
                      class_def_ok := FALSE;
                 END_IF;
                 done := TRUE;
            END_IF;
            IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
                 '.MATERIAL_CLASS_CASE_OF' IN TYPEOF(cl))
            THEN
                 IF (NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
                      '.MATERIAL_CLASS'
                      IN TYPEOF(cl.is_case_of[i].definition[1])))
                 THEN
                      class_def_ok := FALSE;
                 END_IF;
                 done := TRUE;
            END_IF;
            IF (('ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
                 '.ITEM_CLASS_CASE_OF' IN TYPEOF(cl))
                 AND (NOT done))
            THEN
                 IF (NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
                      '.ITEM_CLASS'
                      IN TYPEOF(cl.is_case_of[i].definition[1])))
                 THEN
                      class_def_ok := FALSE;
                 END_IF;
            END_IF;

            done := FALSE;
       END_IF;
  END_REPEAT;

  RETURN(class_def_ok);

  END_FUNCTION; -- check_is_case_of_referenced_classes_definition
```

```
(*

*)
END_SCHEMA; -- ISO13584_extended_dictionary_schema
(*
```

12 ISO13584_library_content_schema

This clause defines the requirements for the **ISO13584_library_content_schema**. The following EXPRESS declaration introduces the **ISO13584_library_content_schema** block and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA ISO13584_library_content_schema;

REFERENCE FROM ISO13584_IEC61360_dictionary_schema
     (all_class_descriptions_reachable,
     basic_semantic_unit,
     class,
     class_BSU,
     code_type,
     content_item,
     data_type,
     data_type_element,
     definition_available_implies,
     dictionary_element,
     int_measure_type,
     level_type,
     list_to_set,
     named_type,
     non_quantitative_int_type,
     property_BSU,
     property_DET,
     real_measure_type,
     revision_type,
     value_format_type,
     version_type);

REFERENCE FROM ISO13584_variable_semantics_schema
     (property_semantics);

REFERENCE FROM ISO13584_domain_resource_schema
     (domain_restriction,
     functional_domain_restriction);

REFERENCE FROM ISO13584_extended_dictionary_schema
     (abstract_functional_model_class,
     applicable_properties,
     applicable_tables,
```



```
      data_exchange_specification_identification,
      data_type_non_quantitative_int_type,
      data_type_type_name,
      data_type_typeof,
      dictionary_identification,
      dictionary,
      dictionary_in_standard_format,
      fm_class_view_of,
      get_property_BSU_from_property_semantics,
      library_iim_identification,
      functional_view_class,
      functional_view_v_c_v,
      view_control_variable_range,
      view_exchange_protocol_identification);

REFERENCE FROM ISO13584_external_file_schema
      (A6_illustration,
      A9_illustration,
      class_extension_external_item,
      dictionary_external_item,
      external_file_protocol,
      external_item,
      illustration,
      illustration_type,
      linked_interface_program_protocol,
      message,
      program_reference);

REFERENCE FROM ISO13584_library_expressions_schema
      (class_instance_constructor,
      collects_assigned_properties);

REFERENCE FROM ISO13584_method_schema
      (method);

REFERENCE FROM ISO13584_instance_resource_schema
      (context_dependent_property_value,
      dic_class_instance,
      dic_f_model_instance,
      primitive_value,
      property_or_data_type_BSU,
      property_value,
      same_translations,
      translated_string_value);

REFERENCE FROM ISO13584_expressions_schema
      (string_expression);

REFERENCE FROM measure_schema
      (amount_of_substance_unit,
      area_unit,
      context_dependent_unit,
```

```
        conversion_based_unit,
        derived_unit,
        derive_dimensional_exponents,
        derived_unit_element,
        dimensional_exponents,
        electric_current_unit,
        global_unit_assigned_context,
        length_unit,
        luminous_intensity_unit,
        mass_unit,
        named_unit,
        plane_angle_unit,
        ratio_unit,
        si_unit,
        solid_angle_unit,
        thermodynamic_temperature_unit,
        time_unit,
        unit,
        volume_unit);

    REFERENCE FROM representation_schema
        (representation);
    (*
```

NOTE The schemas referenced above can be found in the following documents:
ISO13584_IEC61360_dictionary_schema IEC 61360-2
(which is duplicated for convenience in informative annex D of ISO 13584-42),
ISO13584_expressions_schema ISO 13584-20,
ISO13584_domain_ressource_schema This part of ISO 13584,
ISO13584_variable_semantics_schema This part of ISO 13584,
ISO13584_extended_dictionary_schema This part of ISO 13584,
ISO13584_external_file_schema This part of ISO 13584,
ISO13584_library_expressions_schema This part of ISO 13584
ISO13584_method_schema This part of ISO 13584,
ISO13584_instance_resource_schema This part of ISO 13584
measure_schema ISO 10303-41,
representation_schema ISO 10303-43.

12.1 Introduction to the ISO13584_library_content_schema

The role of the **ISO13584_library_content_schema** is to describe the class extension for the classes that constitute a supplier library.

The **ISO13584_library_content_schema** models:

— the descriptions of the set of possible instances of a class,

NOTE 1 In a library exchange context, some classes may be associated with a **content_item** (for instance the **component_class**es that model simple families of parts) when some others have none (for instance the **component_class**es that model generic families of parts).

— the characterisation of those properties whose values are specified for each instance of a class,

NOTE 2 Only properties that are applicable to the class may be associated with values.

NOTE 3 It is a choice of the library data supplier to decide which (possible subset of the) applicable properties are associated with values.

NOTE 4 It is not allowed by ISO 13584 to reuse the same values of the identification characteristics at any time for two different parts, i.e., for two parts of which some non-identification characteristics are different. If such a situation is anticipated, some additional identification characteristics, such that a version, shall be added to discriminate both parts.

— the allowed domains of values for those properties whose values are specified for each instance of a class,

— the characterisation of those properties whose values may be set by the user to select a class instance, and associations of those properties with a domain of allowed values,

— the characterisation of those properties whose values may be computed by the system and associations of those properties with a mechanism that specifies its values,

— the description of the dialogue resources needed to support user access,

— the description of the methods possibly associated with a class specified instance.

The **ISO13584_library_content_schema** does not model:

— the description of those classes that are not associated with a dictionary element,

— the description of the content of those methods that are associated with class instances,

— the description of a software system that supports class instanciation.

12.2 Fundamental concepts and assumption for the ISO13584_library_content_schema

12.2.1 Class extension of non-leaf classes

In a component catalogue where component families are arranged in a tree classification structure, all the leaf classes describe the set of parts belonging to the class. In general, non-leaf classes are mainly used for classification purpose and do not describe any specific population. When a leaf class is further specialized and become a non-leaf class, it might contain a population. The **ISO13584_library_content_schema** enables a library data supplier to describe class extension both for leaf classes and for non-leaf class.

NOTE How instances are displayed when a user makes a query again a non-leaf class is implementation dependent. It is recommended to process the query against both the possible instances defined at the query level, and the union, of all the instances defined in all the subclasses.

12.2.2 Explicit description of class extensions

Explicit description of a class extension consists of describing the set of all instances satisfying a class definition by representing explicitly each instance of a class, and gathering all these representations within a set structure.

The **ISO13584_library_content_schema** supports the explicit description of extensions of the classes that are intentionally defined in the dictionary.

In an explicit description of a class extension, each instance is described by a set of property values. A subset of those properties corresponds to identification properties. The set of values of the identification properties identifies unambiguously each instance within its class. There is a functional dependency from the identification properties to the other properties, for any class instance, the value of any property described in the class extension may be derived by a system from the set of values of the identification properties of the class instance, and from the explicit description of its class extension.

When the set of all instances satisfying a class definition is specified by an explicit description, all the properties are supposed to be selectable by a user for selecting one particular instance within a class. When a sufficient number of properties have been selected, the system should be able to compute the identification properties for the selected instance, and then to derive the values of all the other properties described in the class extension

NOTE 1 Explicit description of class extension is only feasible when there exist a finite number of instances.

NOTE 2 Explicit description of class extension is a very simple model for describing a class population.

12.2.3 Implicit description of class extensions

Implicit description of a class extension consists of describing the set of all instances satisfying a class definition without representing explicitly each instance of the class. In an implicit description, the class is globally modeled, together with a mechanism for deciding whether an instance satisfies the class definition.

The **ISO13584_library_content_schema** also supports the implicit description of extensions of the classes that are intentionally defined in the dictionary.

NOTE 1 Implicit description of class extension is still feasible when there exists an infinite number of instances.

NOTE 2 Implicit description of class extension is a very powerful model for describing a class population. At the implementation level, supporting implicit description of class extension is more complex than supporting explicit class descriptions.

12.2.4 Common pieces of information in implicit description and in explicit description of class extensions

Together with the description of the set of all instances satisfying a class definition, each class extension is described by:

— a set of properties whose values are specified in the class extension;

— a set of **dialogue_resource**s intended to enable user access, and exchanged as separate files;

— **program_reference**s, **representation_reference**s, intended to represent library items, also exchanged as separate files.

NOTE 1 **program_reference**s, **representation_reference**s and **dialogue_ressource**s are subtypes of **class_extension_external_item**.

All the properties that are used to describe a class extension shall be applicable to the class. But, it is up to the library data supplier to decide for which applicable properties, values are specified in the class extension.

EXAMPLE 1 The mass may be defined as an applicable property for a screw class. The supplier may decide to provide the mass values for the different screws of the screw class. In this case, values of this property shall be defined in the class extension. The supplier may also decide to not provide a mass value. In this case, this property shall not be represented in the class extension.

Each class extension is associated with a **content_version** that characterises the extension of the class. The **content_version** shall be incremented when, and only when, the extension of the class is changed, i.e., new instances become allowed, or previous instances are no longer allowed. When the **content_version** of a class extension is incremented, the version of its class **dictionary_element** shall also be incremented. Conversely the **content_version** shall not be changed when the class definition is changed but its population remains unchanged.

Each class extension is associated with a **content_revision** that characterises how the class extension is described. The class revision shall be incremented for any changes in that description, but the changes that modifies the allowed instances of this class. In the latter case, the **content_revision** shall be reset to '000'.

NOTE 2 The **content_version** attribute allows to know the minimal set of class versions that needs to be recorded to be able to instanciate again any instance that was created at some point in time. Only one class per **content_version** value, e. g., the latest one, needs to be recorded. This might be useful for instance for computing the values of some derived properties of any instance, even when these properties were not stored with the instance.

Both **content_version** and **content_revision** are optional. When they are not defined, this means that the class extension gathers instances that where allowed at different point in time. This provides the capability to exchange or to record a whole class history.

NOTE 3 The mechanism to be used to specify the life cycle status of each instance gathered in a class extension without version number is outside the scope of this part of ISO13584.

EXAMPLE 2 A mechanism usable to specify the life cycle status of each instance gathered in an item class extension without version number might be to associate with this item class a functional model class recording for each item instance, e. g., the dates were it became allowed and obsolete, and the class version number were it became allowed and obsolete.

12.2.5 Properties modeling in explicit description of class extensions

In explicit descriptions of class extensions, no operation such that constraint, expression or method are modeled.

When the value of some context dependent characteristic is a function depending on the value of some context parameters, each value of the former shall be associated with a set of values of the latter's.

EXAMPLE For a particular spring, the *compressed-length* of the spring, that is a context dependent characteristic, depends on the context parameter that is the *strength* applied to this spring. In the instance representing this spring, either the *compressed-length* will not be represented, or each value of the *compressed length* will be associated with each value of the corresponding *strength*.

NOTE 1 A **context_dependent_property_value** allows to associate the value of a context dependent characteristic with values of some context parameters that specify the measure context of the former. Using **context_dependent_property_value**, several values may be defined for the same context dependent characteristics, each one associated with a particular measure context

NOTE 2 When an instance, modeled as a **dic_class_instance**, contains values for some context parameters, these context parameters values belong to the measure context of all the context dependent characteristics that depends on these context parameters. Thus, when an instance, modeled as a **dic_class_instance**, contains values for all the context parameters that specify the measure context of one particular context dependent characteristics, only one value may be provided for the latter property: the one corresponding to the unique context specified by the context parameters values.

NOTE 3 **context_dependent_property_value** and **dic_class_instance** are defined in the **ISO13584_instance_resource_schema** documented in clause 6 of this part of ISO 13584.

The following rules define the interpretation of each property that may be associated with a class instance.

— When the instance is an item class instance:

a) a value of a characteristic property is a characteristic of the corresponding instance;

b) a value of a context parameter partially specifies the measure context of any provided value of a context dependent characteristic that depends on this context parameter;

c) a value of a context dependent characteristic is a characteristic of the corresponding item instance in the particular measure context defined by means of context parameter values.

NOTE 4 A where rule, in **dic_item_instance**, ensures that every context dependent characteristics value is associated with a complete measure context.

NOTE 5 **dic_item_instance** is defined in the **ISO13584_instance_resource_schema** documented in clause 6 of this part of ISO 13584.

— When the instance is a functional model class instance:

a) a value of a representation property that is a view control variable imported from the functional view class to which the functional model class refers specifies which particular view instances the functional model instance is able to create;

NOTE 6 A where rule in **explicit_functional_model_class_extension** ensures that, when the functional model class instance creates an instanciable functional view class, all the view control variable of this functional view class are assigned values in the functional model class instance.

NOTE 7 **explicit_functional_model_class_extension** is defined in the clause 12.6.8 of this part of ISO 13584.

b) a value of a property that is a part characteristic imported from the item class to which the functional model class refers allows specifying of which item instance the functional model instance is view of;

NOTE 8 Part characteristics may only be applicable to a functional model class if there are imported from an item class.

c) a value of a representation property whose data type is a **representation**, a **representation_reference** or a **program_reference** defines the representation that belongs to the functional view class the functional model is able to create.

NOTE 9 A where rule in **explicit_functional_model_class_extension** ensures that a value of a representation property whose data type is a **representation**, a **representation_reference** or a **program_reference** only exists when the functional model creates an instanciable functional view class and that such a representation property is unique within a functional model instance.

d) a value of a representation property that is not a view control variable, describes the item the functional model instance is view of, either as part of a functional view class, when the functional view to which the functional model refers is instanciable, or not when the functional view is not instanciable.

EXAMPLE 2 The *price* of a part, and the *minimal quantity of order* of a part may be represented in a functional model class whose class extension is explicitly described. This functional model class should import the identification characteristics of the class of parts the functional model class is view of and there should be one functional model corresponding to each part instance, this functional model providing the *price* and *minimal quantity of order* of the corresponding part.

12.2.6 Typical usage of explicit description of class extensions

The information model for explicit description of class extensions was designed to support information exchanges in a number of typical situations including the following.

a) Description of component families where each component is associated with an explicit part number, and where context dependent characteristics are only defined for one, or a very small number, of measure contexts.

NOTE 1 In this case, each component is efficiently described by its set of characteristic values and by its set of context-dependent characteristics values, each one associated with its own measure context. Moreover, describing each instance according to the same structure allows the LMS to display the entire family instance in a "table-like" structure.

NOTE 2 The **table_like** attribute in the **explicit_model_class_extension** allows the library data supplier to specify that all the instances are defined by the same properties described in the same order.

b) Description of assembly families where there exists a restricted number of allowed configurations.

NOTE 3 When the number of feasible assemblies is small enough, the simple enumeration of all the allowed assembly configurations may be more efficient than modeling explicitly the assembly constraints.

c) Adding functional properties to part description.

NOTE 4 In the ISO 13584 standard series, only characteristics or context dependent characteristics may be associated with parts in their general model. A property that may change without changing the part itself shall be modeled as a functional property, defined in a functional model.

EXAMPLE 1 Price or delivery time are properties that may change without changing the part itself.

NOTE 5 When functional properties are not computed by formulas but stored in tables, a convenient way for exchanging them together with the item class they refer to might be:

— to define, or to reference, a non-instanciable view class that characterize these properties without any view control variable;

— to exchange each parts family using an explicit description of this class extension, each instance being described according to the same structure;

— to associate to each parts family a functional model class defined explicitly whose dictionary definition is a **fm_class_view_of** and whose class extension contains one instance for each instance of the parts family;

— to exchange this functional model class using an explicit description of its class extension, each instance being described according to the same structure.
If the identification properties of the functional model class only consist of characteristics described in the parts family, the LMS can display at the same time, and in the same table-like structure, both the components characteristics and the functional properties described in the functional model class. If view control variables are associated with the defined functional view class, the same kind of display would be possible once the user has set the view control variable values.

EXAMPLE 2 A procurement view may be considered as a a non-instanciable view class.

NOTE 6 When an **explicit_functional_model_class_extension** is associated with an **explicit_item_class_extension**, a global rule ensure that there exists a functional model instance for each item instance.

NOTE 7 The library integrated information model documented in ISO 13584-25[1] should provide for exchanging in the same exchange context general model, functional model and functional view classes provided that all class extensions are described explicitly.

d) Adding representations to part description.

NOTE 8 When there exists, in some functional view class, one or a small number of representations, distinguished by view control variable values, for each part of some part family, a convenient way for exchanging them together with the part families to which they refer might be:

— to define, or to reference, an instanciable view class that characterizes these representations;

— to exchange each parts family using an explicit description of its class extension;

— to associate to each parts family a functional model class defined explicitly whose dictionary definition is a **fm_class_view_of** and whose class extension contains one instance for each particular representation of each instance of the parts family;

[1] To be published

— to exchange this functional model class using an explicit description of its class extension, each instance being described according to the same structure.
If the identification properties of the functional model class only consist of (1) characteristics described in the part family and (2) all the view control variables of the above functional view class, once the user has specified the view control variable values, the LMS is able to generate the required view for any part in the part family.

NOTE 9 When an **explicit_functional_model_class_extension** refers to a functional view class with view control variables, a global rule ensures that for each item instance there exists one functional model instance able to create any of the view the **fm_class_view_of** declares to be able to create.

NOTE 10 **fm_class_view_of** is defined in the **ISO13584_extended_dictionary_schema** documented in clause 11 of this part of ISO 13584.

12.2.7 Properties modeling in implicit description of class extensions

In implicit descriptions of class extensions, relationships between property values may be modeled intentionally by means of operations. Properties of which values are provided in an implicit description of a class extension may be split into:

— **selectable_properties**, whose values are to be set by the user, within an allowed set of values,

— **required_properties**, whose values are intended to be copied by the system from a pre-existing instance,

NOTE 1 Such **required_properties** exist only when the class to be instantiated is associated with another class by an a priori semantic relationship. The required properties may be either characteristics of an item a functional model class instance is is-view-of, (in the case of an a priori is-view-of relationship), or view control variables of the required view for a functional model class instance.

— **derived_properties**, that depend functionally on the selectable properties and on required properties and whose values shall be computed by the system as soon as the values of the selectable properties and required properties of which they depend on, directly or by transitivity are set, and

— **method_variables**, that are internal variables, neither user selectable nor derived from the selectable properties and/or required properties; method-variables are computed during the running of a method.

These four kinds of properties define the template of a class instance in an object-oriented environment.

NOTE 2 In an **item_class**, it is up to the library data supplier to decide which item characteristics are **selectable_properties**, and which item characteristics are **derived_properties**. The only rule is that the set of characteristics that are selectable (called the identification characteristics) shall identify completely and unambiguously a part within its class.

EXAMPLE 1 Assume that in a (abstract) screw class, each screw is defined by three part characteristics *total_length*, *diameter* and *threaded_length* with the constraint: *threaded_length* = (*total_length*) / 2. The following structures may be defined (no context parameters are introduced, therefore all the **selectable_properties** are all identification characteristics).

selectable_properties (identification characteristics)	**derived_properties**
total_length, diameter	*threaded_length*
threaded_length, diameter	*total_length*
total_length, diameter, threaded_length	

In any case, the constraints defined for each selectable property must ensure that only allowed instances may be created at the end of the user selection process.

All the properties may be defined by the library data supplier as either optional or mandatory. If they are optional, they may be explicitly assigned a lack of value.

EXAMPLE 2 In a bolted assembly family, the washer may be defined as optional. In this case, the library user might select a bolted assembly instance with no washer.

NOTE 3 An ISO 13584-conformant LMS may enable the user to create partially defined instances of which all the mandatory properties are still not defined. However the defined values shall fulfil their relevant constraints.

EXAMPLE 3 In a screw family, where *length* and *diameter* are mandatory properties, the library end-user might want to represent a screw instance whose precise *length* is still not defined but whose *diameter* equals 5. The LMS may allow the library end-user to represent this concept as a partially defined instance.

Operations consist of:

— *constraints* that restrict the set of instances of the class and define implicitly the class extensions, and

— *methods* that create views.

All the constraints are modelled as **domain_restriction**s using the resource constructs defined in the **ISO13584_domain_resource_schema**.

Constraints are used in three different roles:

— each **selectable_properties** is associated with a **domain_restriction** that defines a set of values that include all the allowed values for this **selectable_properties** and enable the user selection; all the **domain_restriction**s constraints are gathered in the **class_extension** attribute of an **implicit_model_class_extension**; in this part of ISO 13584, this role is called: *domain definition;*

— each **derived_properties** is associated with a **functional_domain_restriction** that defines the derivation function that enables computation of its value, directly or indirectly, from the values of the **selectable_properties**; these constraints are gathered in the **derivation** attribute of an **implicit_model_class_extension**; in this part of ISO 13584, this role is called: *derivation function definition;*

NOTE A derivation function may use, within its parameters, other **derived_properties**, but these **derived_properties** may be derived, directly or indirectly from **selectable_properties**, and the dependency graph is acyclic.

— a*dditional* **domain_restriction**s may be specified in the **filters** attribute of an **implicit_model_class_extension**; they permit further restriction of the set of allowed instances and support different user selection processes; in this part of ISO 13584, this role is called: *filter definition*.

The Library Management System shall ensure that:

— for every **domain_restriction** that plays a role of domain definition and for which the **assumes** properties are set, only the values belonging to the specified domain are proposed for user selection when he/she wants to choose one property of the **defines** property set;

— when some filters further restrict the set of allowed values of a **selectable_properties** intended to be set by the user, either the illegal values are removed from the displayed values, or the messages associated with these filters are displayed before the user selection;

— for every **functional_domain_restriction** for which the **assumes** properties are set, whether it plays a role of derivation function or a role of filter, the **defines** properties are automatically assigned the values specified by the constraint;

— at the end of the selection process, all the constraints, in the three roles, are satisfied.

EXAMPLE 4 Assume that, in an engineering library, a pipe is identified, within its **item_class_extension** class, by its *diameter*, *thickness* and *material*. These identification characteristics constitute, together with the possible context parameters, the selectable properties of the class. Assume that from these properties, its *mass* and *supported_pressure* may be derived. These characteristics are derived properties. Assume that the library data supplier, or the library user during the user adaptation of the library, wants to enable a user selection based either on a *required_pressure* (only the pipes that support this pressure are proposed for user selection) or on the so-called *spec* (when the *spec* is defined, the only selection of the *diameter* completely defines the pipe). Under these assumptions, the pipe class extension would be modelled as follows:

— two context parameters, belonging to the **selectable_properties** would be defined: *required_pressure* and *spec*, each one associated with a **domain_restriction** that defines a set of allowed values;

— three part characteristics would be defined as **selectable_properties**, associated with a **domain_restriction** (for instance, a **table_defined_domain**), namely *diameter*, *thickness* and *material*;

— two part characteristics would be defined as **derived_properties**, associated with a **functional_domain_restriction** (for instance, a **table_defined_value**), namely *mass* and *supported_pressure*,

— two algorithms would be defined: one in the form of a **functional_domain_restriction** that would derive the *thickness* and *material* from the *spec* and the *diameter*, the other, in the form of a **domain_restriction** that would define for different intervals of the required *required_pressure* the triple (*diameter*, *thickness*, *material*) that support this pressure (for instance, a **table_defined_domain**). These algorithms would be recorded in the **filters** attribute of the **implicit_model_class_extension**.

NOTE 4 The user might select the pipe in three ways:

— selecting in any order *diameter, thickness* and *material*, or

— selecting in any order *spec* and *diameter*, or

— selecting first the required *required_pressure* and then, in the filtered domain, the *diameter, material* and *thickness*.

12.2.8 Assemblies modeling in explicit description of class extensions

A property of an item of which the data type is defined by an **item_class** stands for the composition is-part-of relationship in any of its possible meaning: aggregation part / whole relationship if the **item_class** is a **component_class**, composition relationship specifying of which material(s) an object is made if the **item_class** is a **material_class**, etc. Thus, similar assembled items may be defined by a class of which some or all properties have **item_class**es as their data types.

Explicit description of an assembled item class extension consists of describing the set of all allowed configurations of the assembled item as a set of assembled item instances, of which some or all of the property have **item_class** instances as values.

EXAMPLE 1 The set of all the allowed bolt + nut assemblies, where each bolt belongs to the C_bolt class, with two identification characteristics: *diameter* and *length*, and where each nut belongs to the C_nut class, with one identification characteristic: *diameter* may be modeled as an assembled item class C_bold_and_nut . This class would specify two properties: *the_bolt*, of which the data type is the C_bolt class, and *the_nut*, of which the data type is the C_nut class. Explicit description of the C_bold_and_nut class extension would consist of a set instances, each instance having two properties: *the_bolt* and *the_nut*, and the value of either property being an item class instance.

NOTE 1 A value that is an **item_class** instance is represented as a **dic_item_instance**, a **lib_item_instance** or any of their subtypes.

NOTE 2 **dic_item_instance**, **lib_item_instance** and their subtypes are defined in the **ISO13584_instance_resource_schema** documented in clause 6 of this part of ISO 13584.

To avoid the combinatorial blow-up of the number of instances needed to enumerate all the allowed configurations of an assembly when represented as an assembled item class extension, the following convention is specified by this part of ISO 13584.

When the **item_class** instance that constitutes the value of a property of an assembled item class instance is a **dic_item_instance** of which only some applicable properties are associated with values, this means that the assembly configuration remains allowed when this **dic_item_instance** is replaced by any instance of the class referenced by the **dic_item_instance**, or of any of its subclass, or of any class that is case-of of any of the previous classes provide that the latter instance also contains the set of property values specified for the former instance.

EXAMPLE 2 In the C_bolt_and_nut class described in EXAMPLE 1, if there exist for a bolt 10 possible lengths and 5 possible diameters, each length existing in any diameter, and if there exist only one nut per diameter, enumeration of all possible assembly configurations would request 50 assembled item instances (only one nut fits with each bolt). The above convention allows to represent only 5 assembled item instances. Each instance would reference, by means of a **dic_item_instance,** an instance of the C_bolt class without value for its *length* property, and, by means of a **lib_item_instance,** an fully defined instance of the C_nut class with a value of *diameter* equal to the value of the *diameter* of the instance of bolt. When selecting an assembly according to this specification, the library user would need to select by some means not only the assembled item instance, but also the value of the *length* property of the *the_bolt* property of the assembled item.

EXAMPLE 3 In the C_bold_and_nut class described in EXAMPLE 2, if the C_nut class has no longer instances, but is the superclass of two classes, C_hexagonal_nut and C_square_nut with each 5 instances, all the allowed configurations of the assembly may still be represented with 5 assembled item instances. Each instance would reference, by means of a **dic_item_instance,** an instance of C_bolt class without value for its *length* property, and, by means of a **dic_item_instance**, an instance of the C_nut class, with a value of *diameter* equal to the value of the diameter of the instance of bolt. When selecting an assembly according to this specification, the library user would need to select by some means not only the assembled item instance, but also the precise subclass of C_nut where the nut is taken, and the value of the *length* property of the bolt.

12.2.9 Assemblies modeling in implicit description of class extensions

Implicit description of an assembled item class extension consists of describing the set of all allowed configurations of the assembled item by means of constraints between properties that may have **item_class** instances as values.

— each **selectable_properties** of which the data type is defined by an **item_class** shall be associated with a **domain_restriction** that specifies its domain as a class, or part of a class.

NOTE 1 Using the resource constructs defined in clause 10 of this part of ISO 13584, this domain may be either the class that defines the data type of the property, or any of its subclass;

EXAMPLE 1 In the C_bolt_and_nut class described in EXAMPLE 1 of the previous clause, the *the_bolt* property might be defined as an identification characteristics. It might be associated with a domain restriction that specifies that it shall belong to the C_bolt class.

— each **derived_properties** of which the data type is defined by an **item_class** shall be associated with a **functional_domain_restriction** that enables computation of the **item_class** instance that constitutes its value, directly or indirectly, from the values of the **selectable_properties**;

EXAMPLE 2 In the C_bolt_and_nut class described in EXAMPLE 1 of the previous clause, the *the_nut* property might be defined as a derived property. It would be associated with a **functional_domain_restriction** that would consist of a **class_instance_constructor** able to generate an instance of the C_nut class with diameter x when the *the_bolt* attribute of the assembled item class has a diameter of x. Note that with such a description, the library user would have to select first the bolt, the nut being computed by the system.

NOTE 2 **class_instance_constructor** is defined in clause 10 of this part of ISO 13584.

— further restrictions of the set of allowed instances may be specified by additional **domain_restriction**s specified in the **filters** attribute of an **implicit_model_class_extension**.

EXAMPLE 3 In the C_bolt_and_nut class described in EXAMPLE 1 of the previous clause, both the *the_bolt* and the *the_nut* properties might be defined as identification characteristics, each one with its whole class as its domain. In this case a filter would be needed to specify the additional constraint that the *diameter* of the *the_bolt* property shall be equal to the *diameter* oft the *the_nut* property. Such a description would allow the user to select first either the bolt or the nut, the system computing the second component of the assembly.

12.2.10 Instances satisfying a class definition in an implicit description of a class extension

In an implicit description of a class extension, the set of allowed instances of a class are all the instances such that:

— each property is associated with one value of its data type, and

— all the constraints defined by the domain definitions, derivation functions and filter definitions evaluate to TRUE.

This extension, which may be infinite, is implicitly defined. It may be explicitly computed when all the data types are discrete and finite.

Thus, in an implicit description of a class extension, like in a number of paper catalogues, the set of allowed instances of a class is implicitly defined by a set of constraints.

Constraints perform a dual role:

— first, and during the user selection process, they allow the LMS to help the user to choose a correct instance. For instance, the LMS may generate a table to be looked up by the user. In this connection they constitute supplier-defined choice guides,

— second, and *a posteriori*, they allow the LMS to check that the selected instance belongs to the extension of a class.

In paper documents, numerous methods are used to express these constraints in order to allow the user to be helped to choose an item within its family and also to ensure final correctness of that choice. The following subclauses present the capabilities that may be used by library data suppliers to specify the content of their classes, and the minimal services that shall be provided by any ISO 13584-conformant LMS to process such a specification during the selection process of a library end-user.

12.2.10.1 Domain definition of the identification characteristics

The domain of an identification characteristic may be stated by four means.

a) Independent domains constraints: these allow the domains, from which **selectable_properties** must be chosen, to be stated. Independent domain constraints can be a range, the whole type domain, or a table. Independent domain constraints shall be represented by **domain_restriction**s that contain a unique **guarded_simple_domain** whose **guard** is **others**.

NOTE 1 When displaying some derived properties appears useful to facilitate the user selection of some identification characteristics, the domain of these identification characteristics may be defined by the library data supplier as a **table_defined_domain** whose table key includes the identification characteristics and whose other columns contain the values of the derived properties. These columns will be presented to the user when the selection of the corresponding identification characteristics is required.

b) Conditional domain constraints: when the domains, to which some **selectable_properties** must belong, cannot be easily defined unless other **selectable_properties** have already been chosen, this form of restriction allows the following to be stated:

   1) the property(ies) for which prior choice is necessary;

2) according to the values of those, the various domains within which the properties to be chosen must be selected.

Conditional domain constraints shall be represented by **domain_restriction**s that contain several **guarded_simple_domain**s guarded by **boolean_expression**s. Each **guarded_simple_domain** defines one of the domains within which the properties to be chosen must be selected.

NOTE 2 The use of conditional domain constraints enforces an order in the user selection process.

NOTE 3 The property(ies) for which prior choice is necessary may include other identification characteristics, context parameters, and, possibly, derived properties that result from the values of these identification characteristics and context parameters.

c) Degenerate domain constraints: when the domain to which some identification characteristics must belong, whether it is an independent domain constraint or a conditional domain constraint, degenerates to a singleton, i.e., a set that contains only one element, the assignment of this value to the corresponding identification characteristics shall not require any action from the user. This value shall be computed by the system as soon as such a domain is known. Degenerate domain constraints are modelled as **domain_restriction**s that are **functional_domain_restriction**s.

d) Exclusion constraints (domain + filters): the statement of some domains in the form of range or tables is sometimes impossible, unless excessive sizes are reached for the choice tables. Exclusion constraints permit statements, in the form of **boolean_expression**s, of the additional restrictions to which the values of some subsets of **selectable_properties** must adhere in order to make such values actually lawful, even though they already belong to their respective domains.

NOTE 4 The use of exclusion constraints enables an unordered user selection process. An exclusion constraint is only taken into account when the last **selectable_properties** involved in the constraint is selected.

NOTE 5 An exclusion constraint defines a mathematical relation between several properties. If the projection of this relation onto each involved property, is smaller than the domain defined for this property, a value for this property selected by the user within its domain may be found to be unlawful when other properties involved in the relation are selected. It is therefore recommended that the projection of the relation onto each property equals the domain of the property as defined by its domain definition. It is also recommended that, for every property involved in the relation, the projection of the relation onto the cross-product of all the other properties involved in the relation includes any t-tuple of values of these other properties that might be selected according to their independent or conditional domain constraints.

NOTE 6 Exclusion constraints were mainly introduced in this part of ISO 13584 to enable library data supplier to capture component selection knowledge. As shown in the next clause, context parameters provide for modelling design problems. Exclusion constraints provide for modelling component selection rules.

12.2.10.2 Part selection through context parameters

Context parameters enable the specification of the requirements that a part shall fulfil.

EXAMPLE The *dynamic_load* and the *required_life_time* of a bearing, the *strength* of a spring, the *required_working_temperature* of a pump and the *specifications* of a pipe are examples of properties that enable the specification of the requirements that a part shall fulfil.

Two kinds of relationships may be defined between context parameters and identification characteristics:

— *Domain definition depending on the context.* When user selection always starts by defining some context parameters of the needed part, the domain definition of some identification characteristics may be defined as a conditional domain constraint whose **assumes** attribute includes this context parameter.

NOTE 1 In this approach, the corresponding identification characteristics cannot be directly selected without first defining these context parameters.

— *Filters based on requirements*. In this approach, each context parameter that may be used to describe requirements are represented within filters. When a set of context parameters, and possibly a subset of the identification characteristics, are sufficient to completely specify the required part, a filter, associated to a **functional_domain_restriction**, generates automatically its missing identification characteristics, and therefore the part itself.

NOTE 2 Filters based on requirements enable the description of different user access methods.

The following subclause specifies the minimal support that shall be provided for user selection by any ISO 13584-conformant Library Management System.

12.2.11 Mandatory support of the user selection process when implicit description of class extensions are used

This clause specifies the services that shall be provided by an LMS to a library end-user when selecting an instance in a library class whose extension is implicitly defined. It defines what pieces of information shall be presented to the user according to the class extension description defined by the library data supplier. Thus, this clause defines an abstract protocol between library data suppliers, who define library classes, and LMS developers, whose software systems process the library classes when implicit description of class extensions are used.

NOTE When explicit description of class extensions are used, the explicit description of each instance is rather simple, and the process used to display the set of instances is not specified in this part of ISO 13584.

12.2.11.1 Design a class extension

Representing a class extension includes the following:

a) selecting among the properties, which ones should be selectable by the user, and thus represented as **selectable_properties**; this means, for an **item_class**:

   1) selecting among the item characteristics, which are the identification characteristics,

   2) deciding which context parameters are to be provided as **selectable_properties**;

b) selecting among the other dictionary-defined properties, which are the **derived_properties** that are to be provided in the class extension; this means, for an **item_class**:

   1) deciding which other item characteristics are to be provided as derived characteristics,

   2) deciding which context parameters are to be provided as **derived_properties**;

   3) deciding which context-dependent characteristics are to be provided;

c) for those classes that may only be instantiated when a related instance of another class exists:

   1) deciding which properties, called **required_properties**, shall exist within the related instance;

EXAMPLE An instance of a functional model class that provides the *basic_geometry* representation of a family of screws may only exist associated with an instance of screw of which the length and diameter are known.

d) defining the relevant constraints, i.e.:

   1) defining a single domain definition for each selectable property,

   2) designing a single derivation function for each derived properties,

3) and possibly, adding some filters.

12.2.11.2 Model of a class extension

The mandatory support of the user selection process for some class is based on the dependency graph, and on the set of constraints of this class.

The dependency graph **G** of a class is a directed graph built as follows:

a) let **X** be the set of nodes, and let $\mu \subset \mathbf{X} \times \mathbf{X}$ be the set of edges;

b) the nodes of the graph are the **selectable_properties**, the **required_properties** and the **derived_properties** defined by the class;

c) the edges of the graph are the dependency relationships between properties that appear in the **defines** attributes and properties that appear in the **assumes** of the same domain definition or derivation function defined by the class; formally, for every pair $(i,j) \in \mathbf{X} \times \mathbf{X}$, there exists an edge from **i** to **j** in **G** if and only if:

   1) there exists a **self_property_semantics** that references **i** through its **the_property** attribute and belonging to the **assumes** set of a domain definition or a derivation function defined by the class, and

   2) there exists a **self_property_semantics** that references **j** through its **the_property** attribute and belonging to the **defines** set of the same domain definition or derivation function as the one obtained above;

NOTE The filters are not represented in dependency graph.

From the **acyclic_class_extension_definition** function that constrains every **model_class_extension**, the result is that the built graph **G** is a directed acyclic graph.

The set of constraints **S** is the set of all the **domain_restriction**(s) defined by the class, whether they belong to the domain definitions, to the derivation functions or to the filters definitions.

12.2.11.3 Requirements for the user dialogue

At any time during the user selection process:

— let $\mathbf{X_1} \subset \mathbf{X}$ be the set of properties that are assigned a value and let $\mathbf{X_2} \subset \mathbf{X}$ be the set of properties that are not assigned a value. We have $\mathbf{X} = \mathbf{X_1} \cup \mathbf{X_2}$.and $\mathbf{X_1} \cap \mathbf{X_2} = \varnothing$;

— let $\mathbf{S_1} \subset \mathbf{S}$ be the set constraints for which all the **defines** and **assumes** properties have been assigned a value thus ensuring that the constraint holds, and $\mathbf{S_2} \subset \mathbf{S}$ the set of constraints that have not been checked. We have $\mathbf{S} = \mathbf{S_1} \cup \mathbf{S_2}$.and $\mathbf{S_1} \cap \mathbf{S_2} = \varnothing$.

The minimal mandatory support of the user selection process is defined by the following algorithm:

a) Compute **G** and **S**. Assign $\mathbf{X_1} = \varnothing$, $\mathbf{X_2} = \mathbf{X}$, $\mathbf{S_1} = \varnothing$, $\mathbf{S_2} = \mathbf{S}$.

b) Assign values to all the possible **required_properties** and update $\mathbf{X_1}$ and $\mathbf{X_2}$.

c) Compute recursively all the properties that are in the set $\mathbf{X}_2$ and belonging to the **defines** set of a **functional_domain_restriction** of $\mathbf{S}_2$, of which all the **assumes** set is included in $\mathbf{X}_1$. If two different values are assigned to the same property, display the property(ies) with the double assigned value(s) together with the messages in the **constraint_description** of the violated constraint and raise an error: the class extension shall be empty. Otherwise, move all the

computed properties from $\mathbf{X_2}$ to $\mathbf{X_1}$ and all the used **functional_domain_restriction**s from $\mathbf{S_2}$ to $\mathbf{S_1}$.

d) If there exists in $\mathbf{S_2}$ some **domain_restriction** for which all the properties belonging either to the **assumes** set or to the **defines** set are included in $\mathbf{X}_1$, check the corresponding constraints. If all these constraints hold, move them from $\mathbf{S_2}$ to $\mathbf{S_1}$. Otherwise, display the message in the **constraint_description** of the violated constraints, and raise an error: the class extension shall be empty.

e) Repeat until the user-defined selection process ends:

1) Save the current state of the instance.

2) Propose basic undo capabilities and propose for selection, at the minimum, all the **selectable_properties** that are in the $\mathbf{X_2}$ set and of which all the properties of the **assumes** set of the corresponding domain definition belong to $\mathbf{X_1}$.

NOTE 1 Minimal undo capability consists of coming back to end of step d.

3) When the user has selected one property he/she wants to assign a value:

i) build the set ***y*** of all the properties contained in the **defines** attribute of the domain definition ***d*** of this property,

ii) build the set ***f*** of all the filters contained in $\mathbf{S_2}$ of which all properties of the **defines** attribute and **assumes** attribute are included in $\boldsymbol{y} \cup \mathbf{X_1}$,

iii) if the domain definition ***d*** is a **table_defined_domain** ***T***, let $\boldsymbol{f' \subset f}$ be the set of filters such that:

— their **assumes** attribute is a subset of $\mathbf{X_1}$, and

— their **simple_domain** is a **table_defined_domain**.

then:

For every table $\boldsymbol{t_i}$ corresponding to one filter of ***f'***:

— select from $\boldsymbol{t_i}$ the rows where all the **assumes** properties have their values in the current instance,

— project this table on ***y***,

— define the new table **T** by a natural join of the previous table with **T**,

— if **T** is empty, display the message contained in the **constraint_description** of the filter corresponding to $\boldsymbol{t_i}$ and return to e.2, otherwise

— remove the filter corresponding to $\boldsymbol{t_i}$ from **f**;

iv) if there exists some property(ies) ***y'*** in ***y*** that are already assigned a value, and if the domain definition is a **table_defined_domain**, define the new table ***T*** by selecting in ***T*** (possibly modified according to step e.3.iii the row where all the columns corresponding to ***y'*** equal their already assigned value(s). If ***T*** is empty, display the current assigned values for properties in ***y'***, and return to e.2, otherwise remove all the properties of ***y'*** from ***y***,

v) display all the messages contained in the **constraint_description** attribute of the filters in **f** (possibly modified according to step e.3.iii),

vi) display all the properties in ***y*** (possibly modified according to step e.3.iv) that are already assigned a value, and their values,

vii) if there remains only one possible value for each property in ***y*** then select these values for assignment, else get the user assigned values to all the properties in ***y*** by displaying by some means the set of values in their domain definition, the only mandatory requirement is to display for a **table_defined_domain**, its table ***T,*** possibly modified according to steps e.3.iii. and e.3.iv, and possibly with some other columns computed from the class constraint definition.

NOTE 2 If the system removes by some algorithm all the values that would not fulfill the constraints in ***f***, or that would allow to select a new value for an already assigned property in ***y'***, then the display of e.3.v. and e.3.vi. have not to be done.

EXAMPLE When a filter in ***f*** is a **predicate_defined_domain** and the domain definition is a **table_defined_domain**, with a table ***T,*** an algorithm for removing from ***T*** all the rows that would violate ***f*** would be to remove from ***T*** all the rows for which the filter ***f*** evaluates to FALSE.

viii) assign to the properties in ***y*** the value (or values in the case of a t-uple) resulting from e.3.vii,

ix) check whether some properties in ***y*** have two different values in the current instance and in the instance recorded in stage e.1. If this is TRUE, display the property(ies) with double value definition together with the two values and return to e.2., otherwise

x) check if the constraints that correspond to the filters in **f** are fulfilled. If not, highlight the messages contained in the **constraint_description** of the violated constraints and return to e.2., otherwise move all the filters in **f** from $\mathbf{S_2}$ to $\mathbf{S_1}$, and all the properties in $\boldsymbol{y} \cap \mathbf{X_2}$ from $\mathbf{X_2}$ to $\mathbf{X_1}$.

xi) Compute recursively all the properties that are in the $\mathbf{X_2}$ set and belonging to the **defines** set of a **functional_domain_restriction** of $\mathbf{S_2}$, for which all the **assumes** set is included in $\mathbf{X_1}$. If two different values are assigned to the same property, display the property(ies) with the double assigned value(s) together with the messages in the **constraint_description** of the violated constraint and raise an error: the class extension is not consistent. Otherwise, move all the computed properties from $\mathbf{X_2}$ to $\mathbf{X_1}$ and all the used **functional_domain_restriction**s from $\mathbf{S_2}$ to $\mathbf{S_1}$.

xii) If there exists in $\mathbf{S}_2$ some **domain_restriction** for which all the properties belonging both to the **assumes** set or to the **defines** set are included in $\mathbf{X_1}$, check the corresponding constraints. If all these constraints hold, move them from $\mathbf{S_2}$ to $\mathbf{S_1}$. Otherwise, display the message in the **constraint_description** of the violated constraints, and raise an error: the class extension is not consistent.

xiii) End repeat (return to e.).

NOTE 3 When an error is raised, the LMS behavior is implementation dependent.

12.3 ISO13584_library_content_schema constant definitions

This clause introduces the constant definitions in the **ISO13584_library_content_schema**.

12.3.1 Classification_value

A **classification_value** is an INTEGER value used to specify that a particular property is intended to be processed in a particular way on a receiving system. Values from 0 to 9 are reserved for latter registration. All other values may be used by private agreement between the sender and the receiver.

NOTE This part of ISO 13584 does not make any assumption on how each **classification_value** should be interpreted on a receiving system.

EXAMPLE In a catalogue that a manufacturer sends to a reseller, some properties might be provided only for the reseller usage and not for distribution to customers. By private agreement between the manufacturer and the reseller these properties might be associated with a **classification_value** of -1.

EXPRESS specification:

```
*)
TYPE classification_value = INTEGER;
END_TYPE; -- classification_value
(*
```

12.4 ISO13584_library_content_schema: overall architecture of a library

library is the entity that specifies the overall architecture of a parts library. **library** is a subtype of **dictionary** that shall be used when an ISO 13584-conformant exchange context contains, for some of the classes, their class extensions.

In addition to its inherited attributes, a **library** contains the set of **linked_interfaces** that are referenced from the content of the classes belonging to that **library**.

In a **library**, all the view exchange protocols referenced in a class extension shall belong to the **supported_vep** inherited attribute, and all the **program_reference**s, **representation_reference**s and **dialogue_ressource**s referenced in a class extension shall reference protocols belonging either to the inherited **base_protocols** set, or to the **linked_interfaces** set.

In a **library** entity, the **external_file_protocol**s referenced by the inherited **base_protocols** attribute or by the **linked_interfaces** attribute may be any protocol. These two entities may therefore be used for an exchange between a library data supplier and an end-user who agree on some proprietary protocol or interfaces. The **library_in_standard_format**, subtype of **library** only permits those protocols that are defined in the referenced library integrated information model and in the referenced view exchange protocols.

EXAMPLE A library data supplier and a library end-user may agree to use the native format of some CAD system for exchanging the representations of the library components. Such a **library** cannot be exchanged as a **library_in_standard_format**.

EXPRESS specification:

```
*)
ENTITY library
SUPERTYPE OF(library_in_standard_format)
SUBTYPE OF(dictionary);
      linked_interfaces: SET [0:?] OF external_file_protocol;
WHERE
      WR1: QUERY(class <* SELF\dictionary.contained_classes |
           referenced_veps_exist_in_supported_veps(
```



```
          SELF, class)) = [];
     WR2: QUERY(class <* SELF.contained_classes |
          referenced_protocols_exist_in_supported_protocols(
          SELF, class)) = [];
     WR3: QUERY(int <* SELF.linked_interfaces
          | NOT(('ISO13584_EXTERNAL_FILE_SCHEMA'+
          '.LINKED_INTERFACE_PROGRAM_PROTOCOL')
          IN TYPEOF(int))) = [];
END_ENTITY; -- library
(*
```

Attribute definitions:

linked_interfaces: the set of **external_file_protocol**s referenced in the **library**.

Formal propositions:

WR1: all the view exchange protocols referenced in a class extension shall belong to the **supported_vep** set.

WR2: all the **program_reference**s, **representation_reference**s and **dialogue_ressource**s referenced in a class extension shall reference protocols belonging either to the **base_protocols** set, or to the **linked_interfaces** set.

WR3: the **linked_interfaces** shall be **linked_interface_program_protocol**s.

12.5 Library_in_standard_format

A **library_in_standard_format** entity is a **library** that only references in its **base_protocols** inherited attribute and in its **linked_interfaces** attribute those protocols that are allowed by the conformance classes of the library integrated information model and of the view exchange protocols to which it makes reference. Every user LMS supporting these library integrated information model conformance class and view exchange protocol conformance classes must be able to compile such a library.

EXPRESS specification:

```
*)
ENTITY library_in_standard_format
SUBTYPE OF(library, dictionary_in_standard_format);
WHERE
     WR1: QUERY(int <* SELF\library.linked_interfaces
          | SIZEOF(QUERY(vep <* SELF\dictionary.supported_vep
          | int\external_file_protocol.base_protocol IN
          vep\data_exchange_specification_identification
          .external_file_protocols)) = 0) = [];
END_ENTITY; -- library_in_standard_format
(*
```

Formal propositions:

WR1: the **base_protocols external_file_protocol**s referenced in the **linked_interfaces** attribute shall be allowed by the view exchange protocols referenced by the **supported_vep** attribute.

12.6 Extension of a class

This clause introduces the entities definitions for describing extensions of the various classes that may belong to a library.

12.6.1 Class_extension

A **class_extension** is a description of the set of all the different possible instances conforming to the specification defined by a class.

EXPRESS specification:

```
*)
ENTITY class_extension
ABSTRACT SUPERTYPE OF(ONEOF(model_class_extension))
SUBTYPE OF(content_item);
     SELF\content_item.dictionary_definition: class_BSU;
END_ENTITY; -- class_extension
(*
```

Attribute definitions:

SELF\content_item.dictionary_definition: the **class_BSU** that identifies the class extension.

12.6.2 Opt_or_mand_property_BSU

An **opt_or_mand_property_BSU** entity specifies the status of a property with respect to the user selection dialogue. An **opt_or_mand_property_BSU** specifies:

— whether a property is optional, and,

— whether or not the value of this property is intended by the library data supplier to be displayed for the library end-user.

NOTE 1 **opt_or_mand_property_BSU** entity is only used in an implicit description of class extensions. In explicit decription of class extensions, there are no needs to provide non-displayable properties, and optionality is explicit.

EXPRESS specification:

```
*)
ENTITY opt_or_mand_property_BSU;
     property: property_BSU;
     is_optional: BOOLEAN;
     displayable: BOOLEAN;
END_ENTITY; -- opt_or_mand_property_BSU
(*
```

Attribute definitions:

property: the referenced property.

is_optional: if TRUE, the referenced property may be assigned no value, even for a completely defined instance.

NOTE 2 Such a value corresponds to the NULL value.

displayable: if TRUE, the name and value of this property shall be displayed during end-user access.

12.6.3 Property_classification

property_classification is a **classification_value** associated with a particular property to characterize a particular processing of values of this property on a receiving system.

NOTE 1 A property which is not associated with a classification value is not associated with any particular processing.

NOTE 2 This part of ISO 13584 does not make any assumption on how each **classification_value** should be interpreted on a receiving system. Some values of **classification_value** are reserved for latter registration. All other values may be used by private agreement between the sender and the receiver, for instance to control the distribution of values of the referenced property.

EXAMPLE In a catalogue that a manufacturer sends to a reseller, some properties might be provided only for the reseller usage and not for distribution to customers. By private agreement between the manufacturer and the reseller, these properties might be associated with a **classification_value** of -1.

EXPRESS specification:

```
*)
ENTITY property_classification;
     its_value: classification_value;
     prop_def: property_BSU;
END_ENTITY; -- property_classification
(*
```

Attribute definitions:

its_value: the **classification_value** associated with the property.

prop_def: the property that describes the instance property to which the **its_value** refers.

12.6.4 Property_value_recommended_presentation

A **property_value_recommended_presentation** entity captures a recommendation from the library data supplier about how to present values of some property on a user display. It contains a unit, that shall be compatible with the unit defined in the dictionary for the corresponding property, and a value format for presenting the value if and only if it is converted in the **recommended_presentation_unit** unit. Such conversion capabilities are not required to be supported by ISO 13584 implementations. If they are not supported, values shall be presented as specified in the dictionary definition of the property, possibly modified according to the possible **value_format** defined in the **formatted_column**s of the tables that contains the properties.

NOTE 1 Within an ISO 13584-exchange context, the value of a property is always represented according to the unit specified in the dictionary definition of the property.

NOTE 2 **formatted_column** is defined in the **ISO13584_table_resource_schema** documented in clause 8 of this part of ISO 13584.

NOTE 3 ISO 13584 does not specify what unit should be used for a value of a property outside an ISO 13584-exchange context.

EXAMPLE In a product model conforming to some ISO 10303 application protocol, some product property may reference an ISO 13584 dictionary-defined property of which unit is metre. In product data, the value of this property for one particular product may be expressed in millimeter, using the STEP resource construct **measure_with_unit** if the STEP application protocol allows the use this resource construct.

EXPRESS specification:

```
*)
ENTITY property_value_recommended_presentation;
      prop_def: property_BSU;
      recommended_presentation_unit: unit;
      recommended_presentation_format: value_format_type;
WHERE
      WR1: presentation_unit_is_correct(SELF.prop_def,
           SELF.recommended_presentation_unit);
END_ENTITY;
(*
```

Attribute definitions:

prop_def: the property of which the library data supplier recommends to convert data for presentation purpose.

recommended_presentation_unit: the **unit** in which the library data supplier recommends to convert data for presentation purpose.

recommended_presentation_format: the presentation format recommended by the library data supplier for presenting the values of the **prop_def** property, if and only if these values are converted into the **recommended_presentation_unit** unit.

12.6.5 Model_class_extension

A **model_class_extension** entity describes the general structure of a model class extension, whether it is a **general_model_class**, represented as an **item_class_extension**, or it is a **functional_model_class**, represented as a **functional_model_class_extension**, and whether the class extension is explicitly described as a set of instances, or is implicitly described through a mechanism for deciding whether an instance satisfies the class definition.

EXPRESS specification:

```
*)
ENTITY model_class_extension
ABSTRACT SUPERTYPE OF(ONEOF(
      implicit_model_class_extension,
      explicit_model_class_extension))
SUBTYPE OF(class_extension);
      referenced_external_items: SET [0:?] OF
           class_extension_external_item;
      used_protocols: SET [0:?] OF external_file_protocol;
      referenced_view_exchange_protocol: LIST [0:?] OF
           view_exchange_protocol_identification;
      content_version: OPTIONAL version_type;
      content_revision: OPTIONAL revision_type;
```



```
      recommended_presentation: SET [0:?] OF
           property_value_recommended_presentation;
      classification: SET [0:?] OF property_classification;
WHERE
      WR1: QUERY(item <* SELF.referenced_external_items
           | NOT(item\external_item.used_protocol
           IN SELF.used_protocols)) = [];
      WR2: QUERY (prop <* classification | NOT
           applicable_properties(SELF\content_item.
           dictionary_definition, [prop.prop_def])) = [];
      WR3: (EXISTS (SELF.content_version)
           AND EXISTS (SELF.content_revision))
           OR (NOT EXISTS (SELF.content_version)
           AND NOT EXISTS (SELF.content_revision));
END_ENTITY; -- model_class_extension
(*
```

Attribute definitions:

referenced_external_items: the set of all the **class_extension_external_item**s referenced in the class description: these **class_extension_external_item**s include **dialogue_resource**s, **representation_reference**s and **program reference**s.

NOTE 1 **document_content**s are not **class_extension_external_item**s, they do not belong to the **referenced_external_item**s.

used_protocols: the set of **external_file_protocol**s referenced by the **referenced_external_items** protocols.

NOTE 2 The reference in the **document_content**s do not belong to the **used_protocols** attribute. If these referenced protocols are not supported by an implementation, the document content is just skipped, but the class may nevertheless be successfully compiled (see annex O).

referenced_view_exchange_protocol: the set of view exchange protocols required to process the **model_class_extension**.

content_version: the version number that characterises the extension of the class, i.e., the set of all allowed instances.

NOTE 3 When the **content_version** does not exist, instances may belong to various versions of the same class.

content_revision: the revision number that corresponds to the current description of the **content_version** version of the class extension.

NOTE 4 When the **content_version** does not exist, the **content_revision** shall not exist.

classification: the set of properties that are associated with a classification value.

recommended_presentation: the recommended units and value formats to be used when displaying the values of some properties.

NOTE 5 The **recommended_presentation** attribute captures a recommendation from the library data supplier about how to present values, exchanged according to their dictionary definitions within the exchange file, on a user display. It may involve a conversion in unit and a particular display format. Such a conversion is not required to be supported by ISO 13584 implementation. If it is not supported, values are to be presented as specified in the dictionary definition of the property, possibly modified according to the possible **value_format** defined in the **formatted_column**s of the tables that contains the properties.

NOTE 6 **formatted_column** is defined in the **ISO13584_table_resource_schema** documented in clause 8 of this part of ISO 13584.

Formal propositions:

WR1: all the **external_file_protocol**s referenced in **referenced_external_items** **class_extension_external_item**s shall belong to the **used_protocols** attribute.

WR2: all those properties that are associated with a classification value shall be applicable to the class.

WR3: **content_version** and **content_revision** shall exist together.

Informal propositions:

IP1: the **content_version** shall be incremented when, and only when, the extension of the class is changed, i.e., new instances become allowed, or previous instances are no longer allowed.

IP2: when the **content_version** of a class extension is incremented, the **version** of its class **dictionary_element** shall also be incremented.

NOTE 7 The **content_version** attribute will change less often than the class **version**. **content_version** allows to know the minimal set of class versions that needs to be recorded to be able to instanciate again any instance that was created at some point in time. Only one class per **content_version** value, e. g., the latest one, needs to be recorded.

IP3: the **content_revision** shall be incremented for any changes in the class extension description, but the changes that modifies the allowed instances of this class.

IP4: when the **content_version** of a class extension is incremented, the **content_revision** shall be reset to '000'.

12.6.6 Explicit_model_class_extension

An **explicit_model_class_extension** entity specifies the extension of a class by representing explicitly each instances of a class, and gathering all these instances within a set structure. The **instance_identification** attributes characterizes the subset of properties needed to identify unambiguously each instance within its class.The **table_like** attribute allows to specify that the various instances may be recorded in a table structure, i.e., they are described by the same properties in the same order, and properties that are themselves instances of classes, are also described by the same properties in the same order.

EXPRESS specification:

```
*)
ENTITY explicit_model_class_extension
ABSTRACT SUPERTYPE OF(ONEOF(explicit_item_class_extension,
          explicit_functional_model_class_extension))
SUBTYPE OF(model_class_extension);
     instance_identification: LIST[1:?] OF UNIQUE property_BSU;
     population: LIST[1:?] OF UNIQUE dic_class_instance;
     table_like: BOOLEAN;
WHERE
     WR1: NOT table_like OR (QUERY(inst <* SELF.population |
          NOT same_order_for_properties(population[1].properties,
```



```
            inst.properties)) = []);
        WR2: applicable_properties(
            SELF\content_item.dictionary_definition,
            list_to_set(SELF.instance_identification));
        WR3: all_properties_are_applicable(SELF);
        WR4: same_string_values_translations_for_class_extension(SELF);
        WR5: QUERY(inst <* SELF.population |
            inst.class_def :<>: SELF\content_item.dictionary_definition)
            = [];
        WR6: QUERY(inst <* SELF.population | NOT(
            QUERY(prop <* inst.properties | NOT(EXISTS(prop.its_value))
            AND (prop.prop_def IN SELF.instance_identification)) = []))
            = [];
    END_ENTITY; -- explicit_model_class_extension
    (*
```

Attribute definitions:

instance_identification: the properties that allow to identify unambiguously each described class instance.

NOTE It is not allowed by ISO 13584 to reuse the same values of the identification characteristics at any time for two different parts, i.e., for two parts of which some non-identification characteristics are different. If such a situation is anticipated, some additional identification characteristics, such that a version, shall be added to discriminate both parts.

population: the list of instances that describe the class population.

table_like: a Boolean value that specifies whether the described instances may be displayed using a table structure.

Formal propositions:

WR1: if **table_like** is TRUE, the properties that are associated with each instance shall be the same properties given in the same order.

WR2: all the properties in **instance_identification** shall be applicable to the class.

WR3: all the properties used to define any instance in the class **population** shall be applicable to the class.

WR4: all the instances that define the class **population** shall be such as all the referenced **property_value**s of which values are **translated_string_value**s be translated in the same language(s).

WR5: all the instances that define the class **population** shall reference the same class than the one referenced by the **explicit_model_class_extension** through its inherited **dictionary_definition** attribute.

WR6: the properties that belong to the **instance_identification** list shall never be associated to a null value in all the instances that define the class **population**.

Informal propositions:

IP1: from version to version of the same class, **instance_identification** properties shall not change.

IP2: in the same class, whatever be the version, same value of **instance_identification** properties shall correspond to the same part.

12.6.7 Explicit_item_class_extension

An **explicit_item_class_extension** represents explicitly the extension a general model class. The same structure defines the content of any general model class, whether it is involved in any is-case-of relationship, and whether it is a **component_class**, **material_class**, **feature_class**, or any other **item_class** subtype.

EXPRESS specification:

```
*)
ENTITY explicit_item_class_extension
SUBTYPE OF(explicit_model_class_extension);
      access_icon: OPTIONAL A9_illustration;
      content_msg: OPTIONAL message;
      create_icon: LIST [0:?] OF A6_illustration;
      create_msg: OPTIONAL message;
      class_presentation_on_paper: LIST [0:?] OF illustration;
      class_presentation_on_screen: LIST [0:?] OF illustration;
WHERE
      WR1: QUERY(inst <* SELF\explicit_model_class_extension.population
           | NOT('ISO13584_INSTANCE_RESOURCE_SCHEMA' +
           '.DIC_ITEM_INSTANCE' IN TYPEOF(inst))) = [];
      WR2: definition_available_implies(
           SELF\content_item.dictionary_definition,
           'ISO13584_IEC61360_DICTIONARY_SCHEMA.ITEM_CLASS' IN TYPEOF(
           (SELF\content_item.dictionary_definition.definition[1])));
      WR3: NOT(EXISTS(SELF.access_icon)) OR (SELF.access_icon IN
           SELF\model_class_extension.referenced_external_items);
      WR4: NOT(EXISTS(SELF.content_msg)) OR (SELF.content_msg IN
           SELF\model_class_extension.referenced_external_items);
      WR5: list_to_set(SELF.create_icon)
           <= SELF\model_class_extension.referenced_external_items;
      WR6: NOT(EXISTS(SELF.create_msg)) OR (SELF.create_msg IN
           SELF\model_class_extension.referenced_external_items);
      WR7: list_to_set(SELF.class_presentation_on_paper)
           <= SELF\model_class_extension.referenced_external_items;
      WR8: list_to_set(SELF.class_presentation_on_screen)
           <= SELF\model_class_extension.referenced_external_items;
      WR9: QUERY(icon <* SELF.class_presentation_on_paper |
           (NOT EXISTS(icon.width)) OR (icon.kind_of_content
           = illustration_type.not_static_picture)) = [];
      WR10: QUERY(icon <* SELF.class_presentation_on_screen |
           (NOT EXISTS(icon.width))) = [];
END_ENTITY; -- explicit_item_class_extension
(*
```


Attribute definitions:

access_icon: the icon that enables class presentation in a menu.

content_msg: the **message** that describes the content of the class if the class is intended to be instantiated.

create_icon: the icon(s) that enable(s) visual presentation of the selectable properties of the item and of its reference coordinate system if the class is intended to be instantiated.

create_msg: the **message** that describes the selectable properties of the item and of its reference coordinate system if the class is intended to be instantiated.

class_presentation_on_paper: the ordered set of **illustration**s that are recommended by the library data supplier to be presented to the user when the content of the class is presented on paper.

class_presentation_on_screen: the ordered set of **illustration**s that are recommended by the library data supplier to be presented to the user when the content of the class is presented on a screen.

Formal propositions:

WR1: all the **dic_class_instance**s of the **population** shall be defined as **dic_item_instance**s.

WR2: if data are available, then **IP1** holds.

WR3: the **access_icon** shall belong to the **SELF\model_class_extension.referenced_external_items** set.

WR4: the **content_msg** shall belong to the **SELF\model_class_extension.referenced_external_items** set.

WR5: the **create_icon** LIST item shall belong to the **SELF\model_class_extension. referenced_external_items** set.

WR6: the **create_msg** shall belong to the **SELF\model_class_extension.referenced_external_items** set.

WR7: the **class_presentation_on_paper illustration**s shall belong to the **SELF\model_class_extension. referenced_external_items** set.

WR8: the **class_presentation_on_screen illustration**s shall belong to the **SELF\model_class_extension. referenced_external_items** set.

WR9: the **class_presentation_on_paper illustration**s shall have width and height attributes but not a **not_static_picture** kind of content.

WR10: the **class_presentation_on_screen illustration**s shall have width and height attributes.

Informal propositions:

IP1: **SELF\content_item.dictionary_definition** shall be defined as **item_class**.

12.6.8 Explicit_functional_model_class_extension

An **explicit_functional_model_class_extension** models explicitly the extension of a functional model class. The same structure defines the content of any **functional_model_class**, whether it is involved in any is-view-of relationship.

When a **functional_model_class_extension** is defined as a **fm_class_view_of dictionary_element**, all the properties intended to match some item instance properties shall be imported from the **item_class** and they shall belong to the **required_item_values** attribute and to the **instance_identification** inherited attribute.

NOTE 1 When an **explicit_functional_model_class_extension** is defined as a **fm_class_view_of dictionary_element** with all the **instance_identification** properties belonging to the **required_item_values** properties, during a library user access, each of the **explicit_functional_model_class_extension** instances may be displayed together with the item instance it is is-view-of. This allows to display with the item both its part characteristics and some of its functional properties.

EXAMPLE Price, quantity of order, stock availability are examples of functional properties.

NOTE 2 When an **explicit_functional_model_class_extension** is defined as a **fm_class_view_of dictionary_element** with all the **instance_identification** properties belonging to the **required_item_values** properties, when the extension of the item class it is is-view-of is defined as an **explicit_item_class_extension**, and when both **explicit_model_class_extension**s have their **table_like** attribute equal to TRUE, each instances of the functional model class may be displayed together with the item instance it is is-view-of by making a natural joint of both tables representing both **explicit_model_class_extension**s populations.

When a **functional_model_class_extension** is defined as a **functional_model_class dictionary_element**, without any a priori is-view-of relationship, the **required_item_values** shall be empty, and the properties intended to match some item instance properties shall be defined as **representation_P_DET**, and they shall belong to the **instance_identification** inherited attribute.

NOTE 3 Only **representation_P_DET**s may be defined in a functional model class. Other kinds of properties may only be imported.

If the functional view class referenced by the **functional_model_class dictionary_element** of the **explicit_functional_model_class_extension** is instanciable, each functional model instance shall contain one, and only one property for which the data type is **representation_reference**, **program_reference** or **representation**. This particular property, referenced by the **referenced_representation** attribute, defines the **representation** to be included in the functional view class generated by the functional model instance. Moreover, if this representation is created by a program, referenced by a **program_reference**, the input parameter(s) of this program shall be defined by a **property_BSU** that is represented in each instance, and the program shall have neither output nor in out parameters. If the functional view class referenced by the **functional_model_class dictionary_element** of the **explicit_functional_model_class_extension** is not instanciable, then **referenced_representation** shall not exist.

If the functional view class referenced by the **functional_model_class dictionary_element** of the **explicit_functional_model_class_extension** defines some view control variables, each functional model instance shall import all these properties and it shall contain one value for each of them. These imported properties define the particular functional view class generated by the functional model instance. These imported properties shall belong to the **instance_identification** inherited attribute.

EXPRESS specification:

```
*)
ENTITY explicit_functional_model_class_extension
SUBTYPE OF(explicit_model_class_extension);
     measure_unit: OPTIONAL global_unit_assigned_context;
     required_item_values: SET [0:?] OF property_bsu;
     referenced_representation: OPTIONAL property_bsu;
     available_views_icon: OPTIONAL A6_illustration;
     available_views_msg: OPTIONAL message;
     context_param_icon: LIST [0:?] OF A6_illustration;
```



```
      context_param_msg: OPTIONAL message;
WHERE
      WR1:definition_available_implies(
           SELF\content_item.dictionary_definition,
           'ISO13584_EXTENDED_DICTIONARY_SCHEMA'+
           '.ABSTRACT_FUNCTIONAL_MODEL_CLASS' IN TYPEOF
           (SELF\content_item.dictionary_definition.definition[1]));
      WR2: required_values_are_non_dependent_p_det(SELF);
      WR3: required_values_are_imported_properties(SELF);
      WR4: SELF.required_item_values <= list_to_set(
           SELF\explicit_model_class_extension.instance_identification);
      WR5: NOT(EXISTS(SELF.available_views_icon))
           OR (SELF.available_views_icon IN
           SELF\model_class_extension.referenced_external_items);
      WR6: NOT(EXISTS(SELF.available_views_msg))
           OR (SELF.available_views_msg IN
           SELF\model_class_extension.referenced_external_items);
      WR7: list_to_set(SELF.context_param_icon) <=
           SELF\model_class_extension.referenced_external_items;
      WR8: NOT(EXISTS(SELF.context_param_msg))
           OR (SELF.context_param_msg IN
           SELF\model_class_extension.referenced_external_items);
      WR9: exists_representation_for_instanciable_view(SELF);
      WR10: all_view_control_variables_belong_to_each_view(SELF);
      WR11: QUERY(a_view <* SELF\explicit_model_class_extension.
           population | NOT('ISO13584_INSTANCE_RESOURCE_SCHEMA' +
           '.DIC_F_MODEL_INSTANCE' IN TYPEOF(a_view))) = [];
      WR12: all_vcvs_belong_to_instance_identification(SELF);
      WR13: NOT EXISTS(referenced_representation)
           OR (QUERY(inst <* SELF.population | NOT
           (is_provided_once_property_value(
           inst, referenced_representation))) = []);
      WR14: NOT EXISTS(referenced_representation)
           OR (QUERY(inst <* SELF.population | NOT
           (number_of_instance_representations(
           inst) = 1)) = []);
      WR15: EXISTS(referenced_representation)
           OR (QUERY(inst <* SELF.population | NOT
           (number_of_instance_representations(
           inst) = 0)) = []);
      WR16: NOT EXISTS(referenced_representation)
           OR (SIZEOF(referenced_representation.definition) = 0)
           OR NOT ('ISO13584_EXTENDED_DICTIONARY_SCHEMA.' +
           'PROGRAM_REFERENCE_TYPE' IN
           data_type_typeof(referenced_representation))
           OR (QUERY(inst <* SELF.population | NOT
           (correct_parameters_for_explicit_program(
           inst, referenced_representation))) = []);
END_ENTITY; -- explicit_functional_model_class_extension
(*
```

Attribute definitions:

measure_unit: the **global_unit_assigned_context** that defines the measure units for all the functional views created by the **functional_model_class**. If this optional attribute is not provided, the default value for **length_measure** is millimetre and for **planar_angle** measure it is degree. No default values are defined for the other units.

required_item_values: the item characteristics whose values are required to be able to instantiate the functional model class. These properties shall belong to the **imported_properties_from_item** of the **fm_class_view_of dictionary_element**.

NOTE 4 Only the item characteristics required to instantiate the **functional_model_class** should appear.

referenced_representations: the representation referenced in the class description; if it exists, this representation belongs to the content of the view the functional model is able to create

available_views_icon: the icon that enable visual presentation of the various views that may be created by the functional model class.

available_views_msg: the **message** that describes the various views that may be created by the functional model class.

context_param_icon: the icon(s) that enable(s) visual presentation of the selectable properties required for view creation.

context_param_msg: the **message** that describes the free model properties required for view creation.

Formal propositions:

WR1: if data are available, then **IP1** holds.

NOTE 5 **functional_model_class** and **fm_class_view_of** are subtypes of **abstract_functional_model_class**.

WR2: if **required_item_values** is not empty, **SELF\content_item.dictionary_definition** shall be defined as a **fm_class_view_of**, and all the **required_item_values** shall be defined as **non_dependent_P_DET**s.

WR3: if **required_item_values** is not empty, **SELF\content_item.dictionary_definition** shall be defined as a **fm_class_view_of**, and all the **required_item_values** shall belong to the **imported_properties_from_item** attribute of this **fm_class_view_of**.

WR4: the **required_item_values** properties shall belong to the set of the **instance_identification** properties.

NOTE 6 When the two sets are equal, the functional model class instance may be automatically computed by the system once the item instance is selected.

WR5: the **available_views_icon** shall belong to the **SELF\model_class_extension referenced_external_file** set.

WR6: the **available_views_msg** shall belong to the **SELF\model_class_extension referenced_external_file** set.

WR7: the **context_param_icon** LIST item shall belong to the **SELF\model_class_extension referenced_external_file** set.

WR8: the **context_param_msg** shall belong to the **SELF\model_class_extension referenced_external_file** set.

WR9: if the functional view class referenced by the **explicit_functional_model_class_extension** instance is not a **non_instanciable_view_class**, then the **referenced_representation** exists and its data type is either a **representation_type,** a **representation_reference_type** or **program_reference_type**. If the functional view class referenced by the **explicit_functional_model_class_extension** is a **non_instanciable_view_class**, then the **referenced_representation** does not exist.

WR10: all the view control variables that are defined in the functional view class referenced by the **explicit_functional_model_class_extension** dictionary definition shall be used to describe each **dic_class_instance** belonging to the **SELF\model_class_extension.population** list.

WR11: all the **dic_class_instance**s of the population shall be defined as **dic_f_model_instance**s.

WR12: all the view control variables that are defined in the functional view class referenced by the **explicit_functional_model_class_extension** dictionary definition and all the **required_item_values** shall belong to the **SELF\model_class_extension.instance_identification** list.

WR13: if **referenced_representation** exists then this **property_BSU** is associated with exactly one value in each **explicit_functional_model_class_extension** instance.

WR14: if **referenced_representation** exists then there exists exactly one **property_BSU** the data type of which is **representation_type**, **representation_reference_type** or **program_reference_type** in the **properties** attribute of each **explicit_functional_model_class_extension** instance.

WR15: if **referenced_representation** does not exist then there exists no **property_BSU** the data type of which is **representation_type**, **representation_reference_type** or **program_reference_type** in the **properties** attribute of each **explicit_functional_model_class_extension** instance.

WR16: if **referenced_representation** exists then this **property_BSU** is associated in each **explicit_functional_model_class_extension** instance with one **program_reference** of which both the **out_parameters** and **inout_parameters** lists are empty, and of which all the values of the **in_parameters** attributes are **property_BSU**s that are associated with one value in the same **explicit_functional_model_class_extension** instance.

Informal propositions:

IP1: **SELF\content_item.dictionary_definition** shall be defined as **abstract_functional_model_class** or any of its subtypes.

12.6.9 Implicit_model_class_extension

An **implicit_model_class_extension** entity specifies the extension of a class through a mechanism for deciding whether an instance satisfies the class definition.

The **class_extension** attributes contain a set of **domain_restriction**s, each one defining the domain of one, or several, selectable properties. Such a **domain_restriction** may be a **functional_domain_restriction** when the property domain degenerates to a singleton. In the latter case, the value of the selectable property or properties shall not be selected by the user: it shall be automatically computed by the system as soon as the corresponding function may be computed.

The **derivation** attribute contains the derivation functions that enable the computation value of some **derived_properties** of the SELF instance as a function of some other instance properties. The derivation process is defined by a **functional_domain_restriction**.

Performing a **functional_domain_restriction** consists of evaluating each **guard** that guards the **simple_functional_domain**s, and, if some of them evaluate to TRUE, performing one of the functions whose guard evaluates to TRUE. If all the guards evaluate to FALSE, an error shall occur.

NOTE If the derivation function does not depend upon any other property, only one **guarded_simple_domain** needs to be specified whose **guard** is a **boolean_literal**, as defined by ISO 13584-20.

The **filters** attribute enables the specification of constraints on **selectable_properties** that shall evaluate to TRUE for any allowed instance. Such constraints enable further restriction on the domains defined, by the **class_extension**, for the constrained **selectable_properties**. The use of these constraints to filter the displayed values of the constrained selectable properties is not mandatory for a LMS conforming to ISO 13584. Each filter is associated with a **constraint_description** message that shall be displayed when the system only checks the filters after a user selection process.

EXPRESS specification:

```
*)
ENTITY implicit_model_class_extension
ABSTRACT SUPERTYPE OF(ONEOF(item_class_extension,
functional_model_class_extension))
SUBTYPE OF(model_class_extension);
      selectable_properties: LIST [0:?] OF UNIQUE
           opt_or_mand_property_BSU;
      required_properties: LIST [0:?] OF UNIQUE
           opt_or_mand_property_BSU;
      derived_properties: LIST [0:?] OF UNIQUE
           opt_or_mand_property_BSU;
      class_extension: SET [0:?] OF domain_restriction;
      derivation: SET [0:?] OF functional_domain_restriction;
      filters: SET [0:?] OF domain_restriction;
WHERE
      WR1: QUERY(opt_or_mand <* SELF.selectable_properties
           | NOT applicable_properties
           (SELF\content_item.dictionary_definition,
           [opt_or_mand.property])) = [];
      WR2: QUERY(opt_or_mand <* SELF.derived_properties
           | NOT applicable_properties
           (SELF\content_item.dictionary_definition,
           [opt_or_mand.property])) = [];
      WR3: (QUERY(dom <*
           (SELF.class_extension + SELF.derivation + SELF.filters)
           | (QUERY(sem <* dom.assumes
           | NOT('ISO13584_VARIABLE_SEMANTICS_SCHEMA'
           + '.SELF_PROPERTY_SEMANTICS' IN TYPEOF(sem))) <> [])) = [])
           AND
           (QUERY(dom <* SELF.filters
           | (QUERY(sem <* dom.defines
           | NOT('ISO13584_VARIABLE_SEMANTICS_SCHEMA'
           + '.SELF_PROPERTY_SEMANTICS' IN TYPEOF(sem))) <> [])) = []);
      WR4: QUERY(dom <* (SELF.class_extension + SELF.derivation)
           | QUERY(sem <* dom.defines
           | NOT('ISO13584_VARIABLE_SEMANTICS_SCHEMA'
```



```
     +'.SELF_PROPERTY_VALUE_SEMANTICS' IN TYPEOF(sem))
     OR EXISTS(sem.its_own_property))
     <> []) = [];
WR5: NOT all_class_descriptions_reachable(
     SELF.dictionary_definition) OR (QUERY(dom <*
     (SELF.class_extension + SELF.derivation + SELF.filters)
     | (QUERY(sem <* dom.assumes
     | NOT(sem\property_semantics.the_property IN
     provided_properties_list(SELF.dictionary_definition)))
     <> [])) = []);
WR6: NOT all_class_descriptions_reachable(
     SELF.dictionary_definition) OR (QUERY(dom <*
     (SELF.class_extension + SELF.filters)
     | (QUERY(sem <* dom.defines
     | NOT(sem\property_semantics.the_property IN
     selectable_properties_list(SELF.dictionary_definition)))
     <> [])) = []);
WR7: NOT all_class_descriptions_reachable(
     SELF.dictionary_definition) OR (QUERY(dom <*
     (SELF.class_extension + SELF.derivation + SELF.filters)
     | (QUERY(tab <* dom.base_tables | NOT applicable_tables(
     SELF.dictionary_definition, [tab])) <> [])) = []);
WR8: acyclic_class_extension_definition(
     SELF.dictionary_definition);
WR9: QUERY(prop <* SELF.selectable_properties
     | SIZEOF(QUERY(choi <* SELF.class_extension
     | (prop.property IN get_property_BSU_from_property_semantics(
     choi\domain_restriction.defines)))) <> 1) = [];
WR10: QUERY(prop <* SELF.derived_properties
     | SIZEOF(QUERY(f <* SELF.derivation
     | (prop.property IN get_property_BSU_from_property_semantics(
     f.defines)))) <> 1) = [];
WR11: NOT all_class_descriptions_reachable(
     SELF.dictionary_definition) OR (QUERY(f <* SELF.derivation
     | (QUERY(prop <* f.defines
     | NOT(get_property_BSU_from_property_semantics([prop])[1]
     IN derived_properties_list(SELF.dictionary_definition)))
     <> [])) = []);
WR12: NOT all_class_descriptions_reachable(
     SELF.dictionary_definition)
     OR (QUERY(prop <* derived_properties_list(
     SELF.dictionary_definition)
     | SIZEOF(QUERY(f <* defined_derivation_function(
     SELF.dictionary_definition) | QUERY(sem <* f.defines
     | sem\property_semantics.the_property = prop) <> []))
     <> 1) = []);
WR13: QUERY(filt <* filters | NOT(EXISTS(
     filt.constraint_description))) = [];
WR14: QUERY(dom_1 <* class_extension | NOT(QUERY(dom_2 <*
     dom_1.domains | 'ISO13584_DOMAIN_RESOURCE_SCHEMA' +
     '.PREDICATE_DEFINED_DOMAIN' IN TYPEOF(dom_2.domain)) = [])
     ) = [];
```

```
      WR15: QUERY(prop <* SELF.required_properties |
           prop.is_optional) = [];
END_ENTITY; -- implicit_model_class_extension
(*
```

Attribute definitions:

selectable_properties: the instance properties that shall be set by the user when selecting an instance. The LIST order defines the default order they shall be presented to the user.

required_properties: the instance properties that shall exist in another instance when the class is instantiated through a semantic relationship and that will be copied by the system within the instance to be created.

derived_properties: the instance properties that will be derived by the system from the **selectable_properties**. The LIST order defines the default order to be presented to the user.

class_extension: the **domain_restrictions** that specify the set of allowed instances of the class by defining the domain of the various selectable properties.

derivation: the derivation functions that permit the computation of the derived properties.

filters: the **domain_restriction**s that enable the library data supplier to restrict the domain of some selectable properties when the values of some other properties are known.

Formal propositions:

WR1: all the **selectable_properties** shall be applicable to the class.

WR2: all the **derived_properties** shall be applicable to the class.

WR3: all the **domain_restrictions** referenced in the **class_extension**, **derivation**, and **filters** attributes shall refer through their **assumes** attributes only to **variable_semantics** that are **self_property_semantics** and all the **domain_restrictions** referenced in the **filters** attributes shall refer through their **defines** attributes only to **variable_semantics** that are **self_property_semantics**.

WR4: all the **domain_restrictions** referenced in the **class_extension** and **derivation** attributes shall refer through their **defines** attribute only to **self_property_value_semantics**, and these **self_property_value_semantics** shall not have **its_own_property** value.

WR5: all the **domain_restrictions** referenced in the **class_extension**, **derivation**, and **filters** attributes shall refer through their **assumes** attribute only to **self_property_semantics** that refer to property belonging to the provided properties as returned by the **provided_properties_list** function.

WR6: all the **domain_restrictions** of the **class_extension** and of the **filters** shall refer through their **defines** attribute only to **self_property_semantics** that refer to defined **selectable_properties**.

WR7: all the **domain_restrictions** referenced in the **class_extension**, **derivation**, and **filters** attributes shall only refer to tables that are applicable to the class.

WR8: no property shall be part of its own value definition through a **defines/assumes** chain of domain definition.

WR9: each selectable property in the **selectable_properties** shall correspond to one **domain_restriction** in the **class_extension**.

WR10: each derived property in the **derived_properties** shall correspond to one **functional_domain_restriction** in the **derivation**.

WR11: all the **functional_domain_restrictions** of the **derivation** shall return only defined **derived_properties**.

WR12: their exists exactly one derivation function for every defined **derived_properties**.

WR13: all the **domain_restriction**s referenced in **filters** shall contain a **constraint_description** message.

WR14: all the **domain_restriction**s in the **class_extension** attribute shall be constructive: no **predicate_defined_domain** is allowed.

WR15: all the **required_properties** shall be mandatory properties.

12.6.10 Item_class_extension

An **item_class_extension** represents implicitly the extension a general model class. The same structure defines the content of any general model class, whether it is involved in any is-case-of relationship, and whether it is a **component_class**, **material_class**, **feature_class**, or any other **item_class** subtype. It may contain the **supplier_identification** and **supplier_designation** string expressions that generate a human-readable identification of the class instance as defined by the class supplier. It also contains an **identified_item** attribute that specifies whether the class items are identified globally, or they are identified through their constituent components. Interpretation of these attributes shall be as follows. When **identified_item** is FALSE, the item is an assembly. Its human-readable identification shall consist of its **supplier_identification** if any, followed by the set of identifications of its constituent components computed recursively until those components for which **identified_item** is TRUE. When **identified_item** is TRUE, the item is identified on its own. If present, **supplier_identification** contains enough information for identifying unambiguously the item, whether it is a component or a sub-system.

NOTE 1 In electronic commerce, the above specifies what pieces of information need to be exchanged when ordering a component: if **identified_item** is TRUE, the **supplier_identification** value, if it exists, completely identify the item. If **identified_item** is FALSE, the set of **supplier_identification** values of the item and of all its constituent components until those for which **identified_item** equals TRUE are needed to completely identifies the assembly. This corresponds to a bill-of-material-like identification. When **identified_item** is TRUE and **supplier_identification** does not exist, no human-readable identification string is known.

EXPRESS specification:

```
*)
ENTITY item_class_extension
SUBTYPE OF(implicit_model_class_extension);
     selection_context_parameters: LIST [0:?] OF UNIQUE
          opt_or_mand_property_BSU;
     identification_characteristics: LIST [0:?] OF UNIQUE
          opt_or_mand_property_BSU;
     derived_characteristics: LIST [0:?] OF UNIQUE
          opt_or_mand_property_BSU;
     context_dependent_characteristics: LIST [0:?] OF UNIQUE
          opt_or_mand_property_BSU;
     identified_item: BOOLEAN;
     supplier_identification: OPTIONAL string_expression;
     supplier_designation: OPTIONAL string_expression;
     access_icon: OPTIONAL A9_illustration;
```



```
        content_msg: OPTIONAL message;
        create_icon: LIST [0:?] OF A6_illustration;
        create_msg: OPTIONAL message;
        class_presentation_on_paper: LIST [0:?] OF illustration;
        class_presentation_on_screen: LIST [0:?] OF illustration;
    DERIVE
        SELF\implicit_model_class_extension.selectable_properties:
            LIST [0:?] OF UNIQUE opt_or_mand_property_BSU
            := SELF.selection_context_parameters
            + SELF.identification_characteristics;
        SELF\implicit_model_class_extension.derived_properties:
            LIST [0:?] OF UNIQUE opt_or_mand_property_BSU
            := SELF.derived_characteristics
            + SELF.context_dependent_characteristics;
        SELF\implicit_model_class_extension.required_properties:
            LIST [0:?] OF UNIQUE opt_or_mand_property_BSU
            := [];
    WHERE
        WR1: definition_available_implies(
            SELF\content_item.dictionary_definition,
            'ISO13584_IEC61360_DICTIONARY_SCHEMA.ITEM_CLASS' IN
            TYPEOF((SELF\content_item.dictionary_definition.
            definition[1])));
        WR2: QUERY(elt <* SELF.selection_context_parameters |
            in_typeof('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
            '.CONDITION_DET', elt)) = SELF.selection_context_parameters;
        WR3: QUERY(elt <* SELF.identification_characteristics |
            in_typeof('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
            '.NON_DEPENDENT_P_DET', elt)) =
            SELF.identification_characteristics;
        WR4: QUERY(elt <* SELF.identification_characteristics |
            (data_type_typeof(elt.property) <> [])
            AND NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
            '.SIMPLE_TYPE' IN data_type_typeof(elt.property))
            AND NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
            '.CLASS_INSTANCE_TYPE' IN data_type_typeof(elt.property))
            ) = [];
        WR5: QUERY(elt <* SELF.derived_characteristics |
            in_typeof('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
            '.NON_DEPENDENT_P_DET', elt)) = SELF.derived_properties;
        WR6: QUERY(elt <* SELF.context_dependent_characteristics |
            in_typeof('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
            '.DEPENDENT_P_DET', elt)) =
            SELF.context_dependent_characteristics;
        WR7: NOT(EXISTS(SELF.access_icon)) OR (SELF.access_icon IN
            SELF\model_class_extension.referenced_external_items);
        WR8: NOT(EXISTS(SELF.content_msg)) OR (SELF.content_msg IN
            SELF\model_class_extension.referenced_external_items);
        WR9: list_to_set(SELF.create_icon)
            <= SELF\model_class_extension.referenced_external_items;
        WR10: NOT(EXISTS(SELF.create_msg)) OR (SELF.create_msg IN
```



```
            SELF\model_class_extension.referenced_external_items);
        WR11: list_to_set(SELF.class_presentation_on_paper)
            <= SELF\model_class_extension.referenced_external_items;
        WR12: list_to_set(SELF.class_presentation_on_screen)
            <= SELF\model_class_extension.referenced_external_items;
        WR13: QUERY(icon <* SELF.class_presentation_on_paper |
            (NOT EXISTS(icon.width)) OR (icon.kind_of_content
            = illustration_type.not_static_picture)) = [];
        WR14: QUERY(icon <* SELF.class_presentation_on_screen |
            (NOT EXISTS(icon.width))) = [];
    END_ENTITY; -- item_class_extension
    (*
```

Attribute definitions:

selection_context_parameters: the context parameters whose values may be provided by the user to facilitate the selection of the item.

identification_characteristics: the item characteristics whose values shall be set by the user to identify an item within its class.

NOTE 2 It is not allowed by ISO 13584 to reuse the same values of the identification characteristics at any time for two different parts, i.e., for two parts of which some non-identification characteristics are different. If such a situation is anticipated, some additional identification characteristics, such that a version, shall be added to discriminate both parts.

derived_characteristics: the item characteristics that are derived by the system through **functional_domain_restriction** from the **identification_characteristics**, directly or indirectly.

context_dependent_characteristics: the item properties that are derived by the system from the **selection_context_parameters** and possibly the **identification_characteristics**.

identified_item: a Boolean value that specifies whether the instances of the class are identified by the **supplier_identification** attribute alone, or they are also identified by their constituent components.

NOTE 3 An assembly has no identification on its own: the parts that participate to the assembly are part of the identification.

supplier_identification: the OPTIONAL **string_expression** that specifies completely or partially the item identification defined by the library data supplier.

supplier_designation: the OPTIONAL **string_expression** that specifies completely or partially the item designation defined by the library data supplier.

access_icon: the icon that enables class presentation in a menu.

content_msg: the **message** that describes the content of the class if the class is intended to be instantiated.

create_icon: the icon(s) that enable(s) visual presentation of the selectable properties of the item and of its reference coordinate system if the class is intended to be instantiated.

create_msg: the **message** that describes the selectable properties of the item and of its reference coordinate system if the class is intended to be instantiated.

class_presentation_on_paper: the ordered set of **illustration**s that are recommended by the library data supplier to be presented to the user when the content of the class is presented on paper.

class_presentation_on_screen: the ordered set of **illustration**s that are recommended by the library data supplier to be presented to the user when the content of the class is presented on a screen.

Formal propositions:

WR1: if data are available, then **IP1** holds.

WR2: all the **selection_context_parameters** shall be defined as **condition_DET**s.

WR3: all the **identification_characteristics** shall be defined as **non_dependent_P_DET**s.

WR4: the data type of all the **identification_characteristics** shall be either **simple_type**, for atomic items, or **class_instance_type**, for assemblies.

WR5: all the **derived_characteristics** shall be defined as **non_dependent_P_DET**s.

WR6: all the **context_dependent_characteristics** shall be defined as **dependent_P_DET**s.

WR7: the **access_icon** shall belong to the **SELF\model_class_extension.referenced_external_items** set.

WR8: the **content_msg** shall belong to the **SELF\model_class_extension.referenced_external_items** set.

WR9: the **create_icon** LIST item shall belong to the **SELF\model_class_extension.referenced_external_items** set.

WR10: the **create_msg** shall belong to the **SELF\model_class_extension.referenced_external_items** set.

WR11: the **class_presentation_on_paper illustration**s shall belong to the **SELF\model_class_extension.referenced_external_items** set.

WR12: the **class_presentation_on_screen illustration**s shall belong to the **SELF\model_class_extension.referenced_external_items** set.

WR13: the **class_presentation_on_paper illustration**s shall have **width** and **height** attributes but not a **not_static_picture** kind of content.

WR14: the **class_presentation_on_screen illustration**s shall have **width** and **height** attributes.

Informal propositions:

IP1: **SELF\content_item.dictionary_definition** shall be defined as **item_class**.

IP2: from version to version of the same class, **identification_characteristics** properties shall not change

IP3: In the same class, whatever be the version, same value of **identification_characteristics** properties shall correspond to the same part.

12.6.11 Functional_model_class_extension

A **functional_model_class_extension** models implicitly the extension of a functional model class. The same structure defines the content of any **functional_model_class**, whether it is involved in any is-view-of relationship.

The **selectable_properties** of a **functional_model_class_extension** are the **free_model_properties**. These properties may include properties that are **imported_properties_from_item** in the case of a **fm_class_view_of**.

The **required_properties** of a **functional_model_class_extension** may only exist when it is defined as a **fm_class_view_of dictionary_element**. In this case, the **functional_model_class_extension** may specify, through the **required_item_characteristics** which item characteristics shall exist in the **item_class** instance to be able to instantiate the **functional_model_class_extension** through the a priori is-view-of relationship. These required properties are supposed to be copied by the system in the **functional_model_class_extension** instance.

The **derived_properties** of a **functional_model_class_extension** are the **representation_properties**. These properties may include properties that are **imported_properties_from_item** in the case of a **fm_class_view_of**. The derivation functions of these properties are defined in the **functional_model_class_extension**.

NOTE 1 When the number of the properties that are applicable to an **item_class** is very large, it might prove useful to split the values of these properties into different classes. An **item_class** contains at the minimum the values of the properties that are needed to identify an item instance. Several **functional_model_class**, each one defined as a **fm_class_view_of** and corresponding to one particular user perspective on the data, import those properties that are pertinent for this perspective. The perspective may be characterised by defining **non_instanciable_functional_view_class**.

When a **functional_model_class_extension** is defined as a **functional_model_class dictionary_element**, without any a priori is-view-of relationship, all the properties intended to match some item instance properties shall be defined as **free_model_properties** without any **required_item_characteristics**.

Besides referencing programs or representations provided as **external_item**s, a functional model class extension may reference ISO 10303-43 **representation**s that are included in the library delivery file. These **representation**s are intended to be sent to the modelling system on request from the user (see clause 14, **send_representation_statement**). All the **representation**s are referenced in the **referenced_representation** attribute.

EXPRESS specification:

```
*)
ENTITY functional_model_class_extension
SUBTYPE OF(implicit_model_class_extension);
     measure_unit: OPTIONAL global_unit_assigned_context;
     required_item_characteristics: LIST [0:?] OF UNIQUE
          opt_or_mand_property_BSU;
     free_model_properties: LIST [0:?] OF UNIQUE
          opt_or_mand_property_BSU;
     representation_properties: LIST [0:?] OF UNIQUE
          opt_or_mand_property_BSU;
     method_variables: SET [0:?] OF opt_or_mand_property_BSU;
     referenced_representation: SET [0:?] OF representation;
     provided_methods: SET [0:?] OF method;
     available_views_icon: OPTIONAL A6_illustration;
     available_views_msg: OPTIONAL message;
     context_param_icon: LIST [0:?] OF A6_illustration;
     context_param_msg: OPTIONAL message;
DERIVE
     SELF\implicit_model_class_extension.selectable_properties
          : LIST [0:?] OF UNIQUE opt_or_mand_property_BSU
```



```
            := SELF.free_model_properties;
      SELF\implicit_model_class_extension.required_properties
            : LIST [0:?] OF UNIQUE opt_or_mand_property_BSU
            := SELF.required_item_characteristics;
      SELF\implicit_model_class_extension.derived_properties
            :LIST [0:?] OF UNIQUE opt_or_mand_property_BSU
            := SELF.representation_properties;
WHERE
      WR1:definition_available_implies(
            SELF\content_item.dictionary_definition,
            'ISO13584_EXTENDED_DICTIONARY_SCHEMA.'+
            'ABSTRACT_FUNCTIONAL_MODEL_CLASS' IN TYPEOF
            (SELF\content_item.dictionary_definition.definition[1]));
      WR2: required_properties_are_non_dependent_p_det(SELF);
      WR3: required_properties_are_imported_properties(SELF);
      WR4: QUERY(elt <* SELF.method_variables |
            in_typeof('ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
            '.REPRESENTATION_P_DET', elt)) = SELF.method_variables;
      WR5: QUERY(elt <* SELF.method_variables |
            applicable_properties(
            SELF\content_item.dictionary_definition,[elt.property]))
            = SELF.method_variables;
      WR6: NOT(EXISTS(SELF.available_views_icon))
            OR (SELF.available_views_icon IN
            SELF\model_class_extension.referenced_external_items);
      WR7: NOT(EXISTS(SELF.available_views_msg))
            OR (SELF.available_views_msg IN
            SELF\model_class_extension.referenced_external_items);
      WR8: list_to_set(SELF.context_param_icon) <=
            SELF\model_class_extension.referenced_external_items;
      WR9: NOT(EXISTS(SELF.context_param_msg))
            OR (SELF.context_param_msg IN
            SELF\model_class_extension.referenced_external_items);
END_ENTITY; -- functional_model_class_extension
(*
```

Attribute definitions:

measure_unit: the **global_unit_assigned_context** that defines the measure units for all the functional views created by the **functional_model_class**. If this optional attribute is not provided, the default value for **length_measure** is millimetre and for **planar_angle** measure it is degree. No default values are defined for the other units.

required_item_characteristics: the item characteristics whose values are required to be able to instantiate the functional model class. These properties shall belong to the **imported_properties_from_item** of the **fm_class_view_of dictionary_element**.

NOTE 2 Only the item characteristics required to instantiate the **functional_model_class** should appear on this list.

free_model_properties: the properties whose values shall be provided by the user in order to be able to instantiate the class. In case of a functional model class referring to a part, they normally correspond to context parameters associated with this part.

NOTE 3 The dictionary definitions of **free_model_properties** should either be **representation_P_DET** (when the property is defined (or inherited) in the functional model class) or **condition_DET** (when the property is imported from an **item_class** by means of the **imported_properties_from_item** attribute).

representation_properties: the functional model properties that are derived by the system from the **free_model_properties** and from the **required_item_characteristics**.

NOTE 4 The dictionary definitions of **representation_properties** may either be **representation_P_DET**, when the property is defined (or inherited) in the functional model class, or any of the dictionary elements that define a property of an **item_class** (i.e., **condition_DET**, **non_dependent_P_DET** or **dependent_P_DET**), when the property is imported from an **item_class**.

method_variables: the properties that are neither set by the user nor derived by derivation functions from the selectable properties. These properties enable the representation of the possible internal variables used, for instance in the context of the methods but belonging to the instance slots in an object oriented system. These properties are only assigned values during method running. They are not intended to be displayed when the user consults the library.

NOTE 5 When the **functional_model_class** imports **view_control_variables** for instance for storing their values in a table, these properties shall be represented in the **method_variable** attribute.

referenced_representations: the representations referenced in the class description.

provided_methods: the methods supported by the functional model instance.

NOTE 6 The **declared_created_views_are_created_rule** rule defined in section 12.7.2 ensures that the set of functional views that may be created by the various **method**s of a **functional_model_class_extension** includes the set of functional views that its corresponding **functional_model_class** declares to be able to create by means of its **created_view** and **v_c_v_range** inherited attributes.

NOTE 7 When a view control variable of the **SELF\content_item.dictionary_definition.definition[1]\ abstract_functional_model_class.created_view** is not represented in the **v_c_v_range** attribute of the **SELF\content_item.dictionary_definition.definition[1]**, or for a **method**, in its **specification.v_c_v_range** attribute, its range is its complete value domain.

available_views_icon: the icon that enable visual presentation of the various views that may be created by the functional model class.

available_views_msg: the **message** that describes the various views that may be created by the functional model class.

context_param_icon: the icon(s) that enable(s) visual presentation of the selectable properties required for view creation.

context_param_msg: the **message** that describes the free model properties required for view creation.

Formal propositions:

WR1: if data are available, then **IP1** holds.

WR2: when the **SELF\content_item.dictionary_definition.definition[1]** is defined as **fm_class_view_of**, all the **required_item_characteristics** shall be defined as **non_dependent_P_DET**s.

WR3: when the **SELF\content_item.dictionary_definition.definition[1]** is defined as **fm_class_view_of**, all the **required_item_characteristics** shall be imported from the **item_class**. Otherwise, it shall be an empty set.

WR4: all the **method_variables** shall be defined as **representation_P_DET**s.

WR5: all the **method_variables** shall be applicable to the class.

WR6: the **available_views_icon** shall belong to the **SELF\model_class_extension**.**referenced_external_file** set.

WR7: the **available_views_msg** shall belong to the **SELF\model_class_extension**.**referenced_external_file** set.

WR8: the **context_param_icon** LIST item shall belong to the **SELF\model_class_extension.referenced_external_file** set.

WR9: the **context_param_msg** shall belong to the **SELF\model_class_extension**.**referenced_external_file** set.

Informal propositions:

IP1: **SELF\content_item**.**dictionary_definition** shall be defined as **abstract_functional_model_class** or any of its subtypes.

NOTE **functional_model_class** and **fm_class_view_of** are subtypes of **abstract_functional_model_class**.

12.7 ISO13584_library_content_schema: RULE definitions

This section presents the EXPRESS rules in the **ISO13584_library_content_schema**.

12.7.1 Assert_oneof_for_library_rule rule

The **assert_oneof_for_library_rule** rule states that a **library** that is not a **library_in_standard_format** cannot be a **dictionary_in_standard_format**.

EXPRESS specification:

```
*)
RULE assert_oneof_for_library_rule FOR(library);
WHERE
     WR1: QUERY(temp <* library |
          NOT('ISO13584_LIBRARY_CONTENT_SCHEMA'
          + '.LIBRARY_IN_STANDARD_FORMAT'
          IN TYPEOF(temp))
          AND('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
          + '.DICTIONARY_IN_STANDARD_FORMAT'
          IN TYPEOF(temp))) = [];

END_RULE; -- assert_oneof_for_library_rule
(*
```

Formal propositions:

WR1: a **library** that is not a **library_in_standard_format** cannot be a **dictionary_in_standard_format**.

12.7.2 Declared_created_views_are_created_rule rule

The **declared_created_views_are_created_rule** rule states that the set of functional views that may be created by the various **method**s of an implicitly-defined **functional_model_class_extension** includes the set of functional views that its corresponding **functional_model_class** declares to be able to create.

EXPRESS specification:

```
*)
RULE declared_created_views_are_created_rule FOR(
     functional_model_class_extension, functional_view_class);
WHERE
     WR1: QUERY(a_model <* functional_model_class_extension |
          (computable_set_of_created_views_from_model
          (a_model\content_item.dictionary_definition)) AND
          NOT((declared_created_views(
               a_model\content_item.dictionary_definition)
          <= (created_views_by_methods(a_model\content_item
          .dictionary_definition))))) = [];

END_RULE; -- declared_created_views_are_created_rule
(*
```

Formal propositions:

WR1: the set of functional views that may be created by the various **method**s referenced by the **provided_methods** attribute of any **functional_model_class_extension**, as defined by its **specification.v_c_v_range** attribute, shall include the set of functional views that its **functional_model_class dictionary_element**, defined by its **dictionary_definition\abstract_functional_model_class** attribute, declares to be able to create through its own **v_c_v_range** attribute.

12.7.3 Complete_identification_for_instance_rule rule

The rule **complete_identification_for_instance_rule** asserts that the library content (if it exists) of the class referenced by a a **class_instance_constructor** is an **implicit_model_class_extension**, and that only optional properties have possibly no values.

NOTE When the content of a class does not exists, this rule does not define the properties whose values shall be provided. Application-dependent constraints may be defined in the view exchange protocol series of parts for the instance of classes that are only defined as a **dictionary_element**.

EXPRESS specification:

```
*)
RULE complete_identification_for_instance_rule FOR(
     class_instance_constructor);
WHERE
     WR1: (QUERY(inst <* class_instance_constructor |
          NOT(definition_available_implies(inst.expr_type,
          (SIZEOF(inst.expr_type.referenced_by) = 0) OR
          ('ISO13584_LIBRARY_CONTENT_SCHEMA.' +
          'IMPLICIT_MODEL_CLASS_EXTENSION'
```



```
            IN TYPEOF(inst.expr_type.referenced_by[1])))))
            = []);
      WR2: (QUERY(inst <* class_instance_constructor |
            QUERY(prop <*
            (list_to_set(collects_assigned_properties(inst.properties)) -
            list_to_set(optional_properties_list(inst.expr_type))) |
            NOT(EXISTS(prop.its_value))) = []) =[]);

END_RULE; -- complete_identification_for_instance_rule
(*
```

Formal propositions:

WR1: if data are available, then **IP1** holds.

WR2: only optional properties may be associated with no values.

Informal propositions:

IP1: the **class_extension** that corresponds to the **class_def** attribute of the **class_instance_constructor** shall be an **implicit_model_class_extension** if this content is available.

12.7.4 Complete_identification_for_item_instance_rule rule

The rule **complete_identification_for_item_instance_rule** asserts that, in an implicit description of an item class extension, an instance built by a **class_instance_constructor** is completely identified with respect to the class extension data model if it exists. It means that it shall reference all its identification characteristics,

NOTE 1 In an instance built by a **class_instance_constructor**, only optional properties have possibly no values.

NOTE 2 When the content of a class does not exists, this rule does not define the properties whose values shall be provided. Application-dependent constraints may be defined in the view exchange protocol series of parts for the instance of classes that are only defined as a **dictionary_element**.

EXPRESS specification:

```
*)
RULE complete_identification_for_item_instance_rule FOR(
      class_instance_constructor, item_class_extension);
WHERE
      WR1: QUERY(inst <* class_instance_constructor |
            NOT(
            definition_available_implies(inst.expr_type,
            (SIZEOF(inst.expr_type.referenced_by) = 0) OR
            (('ISO13584_LIBRARY_CONTENT_SCHEMA.ITEM_CLASS_EXTENSION'
            IN TYPEOF(inst.expr_type.referenced_by[1]))
            AND (collects_assigned_properties(inst.properties)
            = (gm_identification_characteristics_list(inst.expr_type))))
            ))) = [];
END_RULE; -- complete_identification_for_item_instance_rule
```



```
(*
```

Formal propositions:

WR1: if data are available, then **IP1** holds.

Informal propositions:

IP1: the list of properties whose values are specified by the **multiple_arity_class_instance_constructor** shall be the whole list of the identification characteristics for the referenced class.

12.7.5 Complete_identification_for_model_instance_rule rule

The rule **complete_identification_for_model_instance_rule** asserts that, in an implicit description of a functional model class extension, an instance built by a **class_instance_constructor** is completely identified with respect to the class extension data model if it exists. It means that it shall reference all its free properties.

NOTE 1 In an instance built by a **class_instance_constructor**, only optional properties have possibly no values.

NOTE 2 When the content of a class does not exists, this rule does not define the properties whose values shall be provided. Application-dependent constraints may be defined in the view exchange protocol series of parts for the instance of classes that are only defined as a **dictionary_element**.

EXPRESS specification:

```
*)
RULE complete_identification_for_model_instance_rule FOR(
     class_instance_constructor, functional_model_class_extension);
WHERE
     WR1: (QUERY(inst <* class_instance_constructor |
          NOT(
          definition_available_implies(inst.expr_type,
          (('ISO13584_LIBRARY_CONTENT_SCHEMA.'
          + 'FUNCTIONAL_MODEL_CLASS_EXTENSION'
          IN TYPEOF(inst.expr_type.referenced_by[1]))
          AND (collects_assigned_properties(inst.properties)
          = fm_free_model_properties_list(inst.expr_type))))
          )) = []);
END_RULE; -- complete_identification_for_model_instance_rule
(*
```

Formal propositions:

WR1: if data are available, then **IP1** holds.

Informal propositions:

IP1: the list of properties whose values are specified by the **multiple_arity_class_instance_constructor** shall be the whole list of the free model properties for the referenced class.

12.7.6 All_views_available_for_each_component_rule rule

The **all_views_available_for_each_component_rule** rule checks that each **functional_model_class_extension** specifies all the declared views for each item of the corresponding **explicit_item_class_extension**.

EXPRESS specification:

```
*)
RULE all_views_available_for_each_component_rule FOR (
     explicit_model_class_extension,
     explicit_functional_model_class_extension);
WHERE
     WR1: QUERY(a_model <* explicit_functional_model_class_extension |
          NOT(all_views_available_for_components(a_model\content_item
          .dictionary_definition))) = [];

END_RULE; -- all_views_available_for_each_component_rule
(*
```

Formal propositions:

WR1: the set of **dic_f_model_instance** specifying an **explicit_functional_model_class_extension** shall describe, for each **dic_item_instance** specifying an **explicit_model_class_extension** whose dictionary definition is referenced from the **view_of** attribute of the **fm_class_view_of** dictionary definition of that **explicit_functional_model_class_extension**, the complete set of declared views of the functional model.

12.8 ISO13584_library_content_schema function definitions

12.8.1 Acyclic_class_extension_definition

An **acyclic_class_extension_definition** function checks that no property participates in its own definition by belonging to the **assumes** attribute of a **domain_restriction** that defines recursively its own domain or derivation function.

The function calls the **acyclic_order** function that computes recursively the set of properties whose definition depends on some property and checks that this property does not belong to the set.

EXPRESS specification:

```
*)
FUNCTION acyclic_class_extension_definition(cl: class_BSU): LOGICAL;

LOCAL
     edges: SET OF domain_restriction;
     prop: LIST OF property_BSU;
END_LOCAL;

prop := provided_properties_list(cl);
edges := defined_domain(cl) + defined_derivation_function(cl);
```



```
REPEAT i := LOINDEX(prop) TO HIINDEX(prop);
     IF NOT acyclic_order(prop[i], edges, [])
     THEN
          RETURN(FALSE);
     END_IF;
END_REPEAT;

RETURN(TRUE);

END_FUNCTION; -- acyclic_class_extension_definition
(*
```

12.8.2 Acyclic_order

An **acyclic_order** function computes recursively the set of properties (**nodes**) whose definition depends on some property **p** and checks that **p** does not belong to **nodes**.

EXPRESS specification:

```
*)
FUNCTION acyclic_order(p: property_BSU;
     edges: SET OF domain_restriction;
     nodes: SET OF property_BSU): BOOLEAN;

LOCAL
     succ: SET OF property_BSU := [];
     -- set of property_BSU that depends (recursively) on p
     out_edges: SET OF domain_restriction;
     -- set of domain_restrictions that contain one of the
     -- property_BSU of the set nodes in the the_property attribute
     -- of one of their assumes variable_semantics
END_LOCAL;

out_edges := QUERY(e <* edges | (QUERY(v <* e.assumes
     | v\property_semantics.the_property IN nodes) <> []));

REPEAT i := LOINDEX(out_edges) TO HIINDEX(out_edges);

     REPEAT j := LOINDEX(out_edges[i].defines)
          TO HIINDEX(out_edges[i].defines);
          succ := succ + out_edges[i].defines[j]
               \property_semantics.the_property;
     END_REPEAT;

END_REPEAT;

-- p depends on itself:
IF p IN succ
THEN
     RETURN(FALSE);
END_IF;
```

```
-- all the depending properties are reached
IF succ <= nodes
THEN
     RETURN(TRUE);
END_IF;

RETURN(acyclic_order(p, edges, nodes + succ));

END_FUNCTION; -- acyclic_order
(*
```

12.8.3 Defined_domain function

The **defined_domain** function returns the set of **domain_restriction** defined as part of the **class_extension** attribute of a class.

EXPRESS specification:

```
*)
FUNCTION defined_domain(cl: class_BSU): SET OF domain_restriction;

IF NOT EXISTS(cl)
THEN
     RETURN([]); -- the class itself is indeterminate
END_IF;

IF (SIZEOF(cl.referenced_by) = 1)
THEN
     RETURN(cl.referenced_by[1]\implicit_model_class_extension.
          class_extension);
ELSE
     RETURN([]);
END_IF;

END_FUNCTION; -- defined_domain
(*
```

12.8.4 Defined_derivation_function function

The **defined_derivation_function** function returns the set of **functional_domain_restriction** defined as part of the **derivation** attribute of a class.

EXPRESS specification:

```
*)
FUNCTION defined_derivation_function(cl: class_BSU):
     SET OF functional_domain_restriction;

IF NOT EXISTS(cl)
THEN
     RETURN([]); -- the class itself is indeterminate
```



```
END_IF;

IF (SIZEOF(cl.referenced_by) = 1)
THEN
     RETURN(cl.referenced_by[1]
          \implicit_model_class_extension.derivation);
ELSE
     RETURN([]);
END_IF;

END_FUNCTION; -- defined_derivation_function
(*
```

12.8.5 Allowed_properties function

The **allowed_properties** function checks that the properties corresponding to **prop** are allowed for the **model_class_extension** identified by the **cl** parameter. A property is allowed for a class if its **property_BSU** is referred to, either as mandatory or as optional. Moreover, the **property_BSU** shall be referred in the **selectable_properties** attributes, in the **required_properties** attributes, or in the **derived_properties** attributes of the content of this class.

EXPRESS specification:

```
*)
FUNCTION allowed_properties(cl: class_BSU;
     prop: SET OF property_BSU): LOGICAL;

IF NOT EXISTS(cl)
THEN
     RETURN(UNKNOWN); -- the class itself is indeterminate
END_IF;

IF (prop <= (list_to_set(provided_properties_list(cl))))
THEN
     RETURN(TRUE);
ELSE
     RETURN(FALSE);
END_IF;

END_FUNCTION; -- allowed_properties
(*
```

12.8.6 Provided_properties_list function

The **provided_properties_list** function retrieves the list of properties defined as **selectable_properties**, as **required_properties** or as **derived_properties** by a class.

EXPRESS specification:

```
*)
FUNCTION provided_properties_list(cl: class_BSU):
     LIST OF property_BSU;
```

```
LOCAL
     provided_prop: LIST OF opt_or_mand_property_BSU := [];
     prop: LIST OF property_BSU := [];
END_LOCAL;

IF NOT EXISTS(cl)
THEN
     RETURN([]); -- the class itself is indeterminate
END_IF;

IF (SIZEOF(cl.referenced_by) = 1)
THEN
     provided_prop :=
          cl.referenced_by[1]\implicit_model_class_extension.
               derived_properties
          + cl.referenced_by[1]\implicit_model_class_extension.
               required_properties
          + cl.referenced_by[1]\implicit_model_class_extension.
               selectable_properties;

     REPEAT i := 1 TO SIZEOF(provided_prop);
          prop := prop + provided_prop[i].property;
     END_REPEAT;
END_IF;

RETURN(prop);

END_FUNCTION; -- provided_properties_list
(*
```

12.8.7 Provided_properties_or_method_variables function

The **provided_properties_or_method_variables** function computes the set of properties defined as **selectable_properties**, as **required_properties**, as **derived_properties** or as **method_variables** by a class, by collecting the properties returned by the **provided_properties_list** and the method variables returned by the **method_variables** functions.

EXPRESS specification:

```
*)
FUNCTION provided_properties_or_method_variables(cl: class_BSU):
     SET OF property_BSU;

IF NOT EXISTS(cl)
THEN
     RETURN([]); -- the class itself is indeterminate
END_IF;

RETURN(method_variables(cl) + provided_properties_list(cl));

END_FUNCTION; -- provided_properties_or_method_variables
```



```
(*
```

12.8.8 Selectable_properties_list function

The **selectable_properties_list** function computes the list of properties defined as selectable in a class.

EXPRESS specification:

```
*)
FUNCTION selectable_properties_list(cl: class_BSU):
     LIST OF property_BSU;

LOCAL
     prop: LIST OF property_BSU := [];
END_LOCAL;

IF NOT EXISTS(cl)
THEN
     RETURN([]); -- the class itself is indeterminate
END_IF;

IF (SIZEOF(cl.referenced_by) = 1)
THEN
     REPEAT i := 1 TO SIZEOF(cl.referenced_by[1]
               \implicit_model_class_extension.selectable_properties);
          prop := prop + cl.referenced_by[1]\
          implicit_model_class_extension.selectable_properties[i].
          property;
     END_REPEAT;
END_IF;

RETURN(prop);

END_FUNCTION; -- selectable_properties_list
(*
```

12.8.9 Required_defined_properties function

The **required_defined_properties** function computes the list of properties defined as required in a class.

EXPRESS specification:

```
*)
FUNCTION required_defined_properties(cl: class_BSU):
     LIST OF property_BSU;

LOCAL
     prop: LIST OF property_BSU := [];
END_LOCAL;
```

```
IF NOT EXISTS(cl)
THEN
     RETURN([]); -- the class itself is indeterminate
END_IF;

IF (SIZEOF(cl.referenced_by) = 1)
THEN
     REPEAT i := 1 TO SIZEOF(cl.referenced_by[1]
               \implicit_model_class_extension.required_properties);
          prop := prop + cl.referenced_by[1]\
               implicit_model_class_extension.required_properties[i].
               property;
     END_REPEAT;
END_IF;

RETURN(prop);

END_FUNCTION; -- required_defined_properties
(*
```

12.8.10 Derived_properties_list function

The **derived_properties_list** function computes the list of properties defined as derived in a class.

EXPRESS specification:

```
*)
FUNCTION derived_properties_list(cl: class_BSU):
     LIST OF property_BSU;

LOCAL
     prop: LIST OF property_BSU := [];
END_LOCAL;

IF NOT EXISTS(cl)
THEN
     RETURN([]); -- the class itself is indeterminate
END_IF;

IF (SIZEOF(cl.referenced_by) = 1)
THEN
     REPEAT i := 1 TO SIZEOF(cl.referenced_by[1]
               \implicit_model_class_extension.derived_properties);
          prop := prop + cl.referenced_by[1]
               \implicit_model_class_extension.derived_properties[i].
               property;
     END_REPEAT;
END_IF;

RETURN(prop);
```



```
END_FUNCTION; -- derived_properties_list
(*
```

12.8.11 Optional_properties_list function

The **optional_properties_list** function computes the list of properties defined as optional in a class.

EXPRESS specification:

```
*)
FUNCTION optional_properties_list(cl: class_BSU):
     LIST OF property_BSU;

LOCAL
     prop: LIST OF property_BSU := [];
END_LOCAL;

IF NOT EXISTS(cl)
THEN
     RETURN([]); -- the class itself is indeterminate
END_IF;

IF (SIZEOF(cl.referenced_by) = 1)
THEN
     REPEAT i := 1 TO SIZEOF(cl.referenced_by[1]
          \implicit_model_class_extension.derived_properties);
          IF (cl.referenced_by[1]\implicit_model_class_extension
               .derived_properties[i].is_optional)
          THEN
               prop := prop + cl.referenced_by[1]
                    \implicit_model_class_extension.
                    derived_properties[i].property;
          END_IF;
     END_REPEAT; -- derived optional properties of this class

     REPEAT i := 1 TO SIZEOF(cl.referenced_by[1]
          \implicit_model_class_extension.selectable_properties);
          IF (cl.referenced_by[1]\implicit_model_class_extension
               .selectable_properties[i].is_optional)
          THEN
               prop := prop + cl.referenced_by[1]
                    \implicit_model_class_extension.
                    selectable_properties[i].property;
          END_IF;
     END_REPEAT; -- selectable optional properties of this class
END_IF;

RETURN(prop);

END_FUNCTION; -- optional_properties_list
(*
```

12.8.12 Method_variables function

The **method_variables** function computes the method variables defined in a **functional_model_class**. This function is intended to be called on a functional model class extension. Therefore, if it is called on another kind of class, it returns the empty set.

EXPRESS specification:

```
*)
FUNCTION method_variables(cl: class_BSU): SET OF property_BSU;

LOCAL
     prop: SET OF property_BSU := [];
END_LOCAL;

IF NOT EXISTS(cl)
THEN
     RETURN([]); -- the class itself is indeterminate
END_IF;

IF (SIZEOF(cl.referenced_by) = 1)
THEN
     IF NOT('ISO13584_LIBRARY_CONTENT_SCHEMA.'
          + 'FUNCTIONAL_MODEL_CLASS_EXTENSION'
          IN TYPEOF(cl.referenced_by[1])) -- abnormal case
     THEN
          RETURN([]);
     ELSE
          REPEAT i := 1 TO SIZEOF(cl.referenced_by[1]
               \functional_model_class_extension.method_variables);
               prop := prop + cl.referenced_by[1]\
                    functional_model_class_extension.
                    method_variables[i].property;
          END_REPEAT;
     END_IF;
END_IF;

RETURN(prop);

END_FUNCTION; -- method_variables
(*
```

12.8.13 Gm_identification_characteristics_list function

The **gm_identification_characteristics_list** function computes the list of properties defined as identification characteristics in a general model class. This function is intended to be called on a general model class. Therefore, if it is not the case, it returns an empty LIST.

EXPRESS specification:

```
*)
FUNCTION gm_identification_characteristics_list(cl: class_BSU):
```



```
     LIST OF property_BSU;

LOCAL
     prop: LIST OF property_BSU := [];
END_LOCAL;

IF NOT EXISTS(cl)
THEN
     RETURN([]); -- the class itself is indeterminate
END_IF;

IF (SIZEOF(cl.referenced_by) = 1)
THEN
     IF NOT('ISO13584_LIBRARY_CONTENT_SCHEMA.ITEM_CLASS_EXTENSION'
          IN TYPEOF(cl.referenced_by[1])) -- abnormal case
     THEN
          RETURN([]); -- abnormal case
     END_IF;

     REPEAT i := 1 TO SIZEOF(cl.referenced_by[1]
          \item_class_extension.identification_characteristics);
          prop := prop + cl.referenced_by[1]
               \item_class_extension.identification_characteristics[i]
               .property;
     END_REPEAT;
END_IF;

RETURN(prop);

END_FUNCTION; -- gm_identification_characteristics_list
(*
```

12.8.14 Fm_free_model_properties_list function

The **fm_free_model_properties_list** function computes the list of properties defined as free model properties in a functional model class. This function is intended to be called on a functional model class. Therefore, if it is not the case, it returns an empty LIST.

EXPRESS specification:

```
*)
FUNCTION fm_free_model_properties_list(cl: class_BSU):
     LIST OF property_BSU;

LOCAL
     prop: LIST OF property_BSU := [];
END_LOCAL;

IF NOT EXISTS(cl)
THEN
     RETURN([]); -- the class itself is indeterminate
END_IF;
```

```
IF (SIZEOF(cl.referenced_by) = 1)
THEN
     IF NOT('ISO13584_LIBRARY_CONTENT_SCHEMA'
          + '.FUNCTIONAL_MODEL_CLASS_EXTENSION'
          IN TYPEOF(cl.referenced_by[1])) -- normal case
     THEN
          RETURN([]); -- abnormal case
     END_IF;

     REPEAT i := 1 TO SIZEOF(cl.referenced_by[1]
          \functional_model_class_extension.free_model_properties);
          prop := prop + cl.referenced_by[1]
               \functional_model_class_extension
               .free_model_properties[i].property;
     END_REPEAT;
END_IF;

RETURN(prop);

END_FUNCTION; -- fm_free_model_properties_list
(*
```

12.8.15 Exists_super function

The **exists_super** function checks if a class identified by the **cl class_BSU** parameter has a superclass. It returns:

— TRUE if **cl** has a superclass,

— FALSE if **cl** has no superclass, and

— UNKNOWN if the **class dictionary_element** is not available.

EXPRESS specification:

```
*)
FUNCTION exists_super(cl: class_BSU): LOGICAL;

IF NOT EXISTS(cl)
THEN
     RETURN(UNKNOWN); -- the class itself is indeterminate
END_IF;

IF (SIZEOF(cl.definition) = 0)
THEN
     RETURN(UNKNOWN);
ELSE
     IF EXISTS(cl.definition[1]\class.its_superclass)
     THEN
          RETURN(TRUE);
     ELSE
```



```
          RETURN(FALSE);
     END_IF;
END_IF;

END_FUNCTION; -- exists_super
(*
```

12.8.16 Super function

The **super** function computes the super class of a class identified by its **class_BSU**. If the class has no superclass, or if the superclass cannot be computed, it returns an empty set.

EXPRESS specification:

```
*)
FUNCTION super(cl: class_BSU): SET[0:1] OF class_BSU;

IF NOT EXISTS(cl)
THEN
     RETURN([]); -- the class itself is indeterminate
END_IF;

IF (SIZEOF(cl.definition) = 1)
THEN
     IF EXISTS(cl.definition[1]\class.its_superclass)
     THEN
          RETURN([cl.definition[1]\class.its_superclass]);
     END_IF;
END_IF;

RETURN([]);

END_FUNCTION; -- super
(*
```

12.8.17 Is_in_v_c_v_range function

The **is_in_v_c_v_range** function checks if a **property_BSU p** appears once as a **parameter_type** attribute of a **view_control_variable_range** entity in a list **l** of **view_control_variable_range** entities.

EXPRESS specification:

```
*)
FUNCTION is_in_v_c_v_range(p: property_BSU;
     l: SET OF view_control_variable_range): BOOLEAN;

IF (SIZEOF(QUERY(elt <* l | elt.parameter_type = p)) = 1)
THEN
     RETURN(TRUE);
ELSE
     RETURN(FALSE);
END_IF;
```

```
END_FUNCTION; -- is_in_v_c_v_range
(*
```

12.8.18 Get_v_c_v_range function

The **get_v_c_v_range** function returns the **view_control_variable_range** for which the **parameter_type** attribute is equal to a given **property_BSU p** in a list **l** of **view_control_variable_range** entities. It requires that the function **is_in_v_c_v_range** applied to **p** and **l** returns TRUE.

EXPRESS specification:

```
*)
FUNCTION get_v_c_v_range(p: property_BSU;
     l: SET OF view_control_variable_range):
     view_control_variable_range;

LOCAL
     x: SET OF view_control_variable_range;
END_LOCAL;

x := QUERY(elt <* l | elt.parameter_type = p);

RETURN(x[1]);

END_FUNCTION; -- get_v_c_v_range
(*
```

12.8.19 All_v_c_v_range_available function

The function **all_v_c_v_range_available** takes a list **l** of **property_BSU**s and checks that for each **property_BSU** in the **l** list, its **data_type** is a **non_quantitative_int_type** that is available. It returns TRUE if this condition holds, otherwise it returns FALSE.

EXPRESS specification:

```
*)
FUNCTION all_v_c_v_range_available(l: LIST OF property_BSU):
     BOOLEAN;
LOCAL
     res: BOOLEAN:= TRUE;
END_LOCAL;

REPEAT i := 1 TO SIZEOF(l);
     IF NOT(SIZEOF(data_type_non_quantitative_int_type(l[i])) = 1)
     THEN
          res := FALSE;
     END_IF;
END_REPEAT;

RETURN(res);
```



```
END_FUNCTION; -- all_v_c_v_range_available
(*
```

12.8.20 Make_ordered_list_of_v_c_v_range function

The **make_ordered_list_of_v_c_v_range** function takes a list **l** of **property_BSU**s and a list **l_range** of **view_control_variable_range** such that the **all_v_c_v_range_available** function applied to **l** returns TRUE. It returns a list of **view_control_variable_range** based on the **l** list of **property_BSU**s where each **property_BSU** in the **l** list is replaced by a **view_control_variable_range** of which the given **property_BSU** is the **parameter_type** attribute. Each **view_control_variable_range** is either extracted from the **l_range** list, if there exists in the **l_range** list a **view_control_variable_range** of which the given **property_BSU** is the **parameter_type** attribute, otherwise the **view_control_variable_range** is built from its whole domain of the given **property_BSU**. If the **all_v_c_v_range_available** applied to **l** does not return TRUE, the **make_ordered_list_of_v_c_v** function returns the empty list.

EXPRESS specification:

```
*)
FUNCTION make_ordered_list_of_v_c_v_range(
     l: LIST OF property_BSU;
     l_range: SET OF view_control_variable_range):
     LIST OF view_control_variable_range;
LOCAL
     y: view_control_variable_range;
     res: LIST OF view_control_variable_range:=[];
     s: SET[0:1] OF non_quantitative_int_type;
     x: non_quantitative_int_type;
END_LOCAL;

IF NOT all_v_c_v_range_available(l)
THEN
     RETURN([]);
END_IF;

REPEAT i := 1 TO SIZEOF(l);
     IF is_in_v_c_v_range(l[i], l_range)
     THEN
          res := res + get_v_c_v_range(l[i], l_range);
     ELSE
          s := data_type_non_quantitative_int_type(l[i]);
          x := s[1];
          y := view_control_variable_range(l[i],
               x.domain.its_values[1].value_code,
               x.domain.its_values[1].value_code +
               SIZEOF(x.domain.its_values) - 1);
          res := res + y;
     END_IF;
END_REPEAT;

RETURN(res);
```

```
END_FUNCTION; -- make_ordered_list_of_v_c_v_range
(*
```

12.8.21 Cdr_list function

The function **cdr_list** computes the sublist of a list **l** of **view_control_variable_range**. This sublist is the list **l** with its first element removed. This function requires that the list **l** contains at least two elements.

EXPRESS specification:

```
*)
FUNCTION cdr_list(l: LIST [2:?] OF GENERIC: type_elem):
     LIST OF GENERIC: type_elem;

LOCAL
     cdr: LIST OF GENERIC: type_elem := [];
END_LOCAL;

REPEAT i := 2 TO SIZEOF(l);
     cdr := cdr + l[i];
END_REPEAT;

RETURN(cdr);

END_FUNCTION; -- cdr_list
(*
```

12.8.22 Make_tuple function

The **make_tuple** function computes the set of integer tuples belonging to the Cartesian product of an ordered list of integer intervals defined by some **view_control_variable_range**s. The order of the **view_control_variable_range**s in the list **l** defines the order of the tuples values.

EXPRESS specification:

```
*)
FUNCTION make_tuple(l: LIST[1:?] of view_control_variable_range):
     SET [1:?] OF LIST[1:?] OF INTEGER;

LOCAL
     result: SET OF LIST OF INTEGER := [];
     list_sub_tuple: SET OF LIST OF INTEGER;
END_LOCAL;

IF SIZEOF(l) = 1 THEN
     REPEAT i := l[1]\view_control_variable_range.range_lobound TO
          l[1]\view_control_variable_range.range_hibound;
          result := [[i]] + result;
     END_REPEAT;
ELSE
     list_sub_tuple := make_tuple(cdr_list(l));
```



```
      REPEAT i := 1 TO SIZEOF(list_sub_tuple);  -- for each subtuple
           REPEAT j := l[1]\view_control_variable_range.range_lobound
                TO l[1]\view_control_variable_range.range_hibound;
                -- creates one new tuple for each value of l[1]
                result := result + [j + list_sub_tuple[i]];
           END_REPEAT;
      END_REPEAT;
END_IF;

RETURN(result);

END_FUNCTION; -- make_tuple
(*
```

12.8.23 Computable_set_of_created_views_from_model

The **computable_set_of_created_views_from_model** function checks if the set of functional views that may be created by a functional model class identified by the **cl class_BSU** may be computed. It checks whether the **dictionary_element** associated with the **cl** parameter, if available, is an **abstract_functional_model_class**, whether the **content_item** associated with the **cl** parameter, if available, is a **functional_model_class_extension**, and whether all the **view_control_variable**s, defined or inherited by the **functional_view_class** referenced by the **abstract_functional_model_class** have, if available, a **data_type**. The **computable_set_of_created_views_from_model** function returns TRUE if the set of functional views that may be created by the functional model class identified by **cl** may be computed. Otherwise, it returns FALSE.

EXPRESS specification:

```
*)
FUNCTION computable_set_of_created_views_from_model(
      cl: class_BSU): BOOLEAN;

IF NOT EXISTS(cl)
THEN
      RETURN(FALSE); -- the class itself is indeterminate
END_IF;

IF SIZEOF(cl.definition) = 0
THEN
      RETURN(FALSE);
END_IF;

IF SIZEOF(cl.referenced_by) = 0
THEN
      RETURN(FALSE);
END_IF;
IF NOT('ISO13584_EXTENDED_DICTIONARY_SCHEMA.'+
      'ABSTRACT_FUNCTIONAL_MODEL_CLASS' IN TYPEOF(cl.definition[1]))
THEN
      RETURN(FALSE);
END_IF;
IF NOT('ISO13584_LIBRARY_CONTENT_SCHEMA.'+
```



```
      'FUNCTIONAL_MODEL_CLASS_EXTENSION' IN TYPEOF(
      cl.referenced_by[1]))
      AND NOT ('ISO13584_LIBRARY_CONTENT_SCHEMA.'+
      'EXPLICIT_FUNCTIONAL_MODEL_CLASS_EXTENSION' IN TYPEOF(
      cl.referenced_by[1]))
THEN
      RETURN(FALSE);
END_IF;

IF SIZEOF(functional_view_v_c_v(cl.definition[1]
      \abstract_functional_model_class.created_view)) = 0
THEN
      RETURN(FALSE);
END_IF;

RETURN(all_v_c_v_range_available(functional_view_v_c_v(cl.definition[1]
      \abstract_functional_model_class.created_view)));

END_FUNCTION; -- computable_set_of_created_views_from_model
(*
```

12.8.24 Declared_created_views function

The **declared_created_views** function computes the set of functional views that are declared to be created by the **cl abstract_functional_model_class** by means of its **v_c_v_range** attribute. Each functional view is represented by a tuple of integers view control variable value. If the set of functional views that are declared to be created cannot be computed, the **declared_created_views** function returns the empty set.

EXPRESS specification:

```
*)
FUNCTION declared_created_views(cl: class_BSU):
      SET OF LIST OF INTEGER;

LOCAL
      res: SET OF LIST OF INTEGER:=[];
      v_c_vs: LIST OF view_control_variable_range;
END_LOCAL;

IF NOT computable_set_of_created_views_from_model(cl)
THEN
      RETURN([]);
END_IF;

v_c_vs := make_ordered_list_of_v_c_v_range(functional_view_v_c_v(
      cl.definition[1]\abstract_functional_model_class.created_view),
      cl.definition[1]\abstract_functional_model_class.v_c_v_range);

res := make_tuple(v_c_vs);

RETURN(res);
```



```
END_FUNCTION; -- declared_created_views
(*
```

12.8.25 Created_views_by_methods function

The **created_views_by_methods** function computes the set of functional views that are created by the various **method**s of the **cl abstract_functional_model_class**. Each functional view is represented by a tuple of integers view control variable values. If the set of functional views that are created by the various **method**s cannot be computed, the **created_views_by_methods** function returns the empty set.

EXPRESS specification:

```
*)
FUNCTION created_views_by_methods(cl: class_BSU):
     SET OF LIST OF INTEGER;
LOCAL
     res: SET OF LIST OF INTEGER:=[];
     v_c_vs: LIST OF view_control_variable_range;
END_LOCAL;

IF NOT computable_set_of_created_views_from_model(cl)
THEN
     RETURN([]);
END_IF;

REPEAT i := 1 TO SIZEOF(cl.referenced_by[1]
     \functional_model_class_extension.provided_methods);
     v_c_vs := make_ordered_list_of_v_c_v_range(
          functional_view_v_c_v(cl.definition[1]
          \abstract_functional_model_class.created_view),
          cl.referenced_by[1]\functional_model_class_extension
          .provided_methods[i].specification.v_c_v_range);
     res := res + make_tuple(v_c_vs);
END_REPEAT;

RETURN(res);

END_FUNCTION; -- created_views_by_methods
(*
```

12.8.26 In_typeof function

The **in_typeof** function checks the data type **typ** of an **opt_or_mand_property_BSU** if and only if the property dictionary definition exists in the same exchange context.

EXPRESS specification:

```
*)
FUNCTION in_typeof(typ: STRING; elt: opt_or_mand_property_BSU):
     LOGICAL;
```



```
IF SIZEOF(elt.property.definition) = 1 THEN
     RETURN(typ IN TYPEOF(elt.property.definition [1]));
ELSE
     RETURN(TRUE);
END_IF;

END_FUNCTION; -- in_typeof
(*
```

12.8.27 Referenced_veps_exist_in_supported_veps function

The **referenced_veps_exist_in_supported_veps** function checks that the view exchange protocols referenced in a class extension identified by its **class_bsu cl** belong to the **library supported_vep** set.

EXPRESS specification:

```
*)
FUNCTION referenced_veps_exist_in_supported_veps(
     lib: library; cl: class_BSU): LOGICAL;
LOCAL
     class_extension: SET [0:1] OF content_item :=
          cl\basic_semantic_unit.referenced_by;
     class_extension_referenced_veps: SET OF
          view_exchange_protocol_identification;
     dictionary_supported_veps: SET OF
          view_exchange_protocol_identification;
     tmp: LOGICAL;
END_LOCAL;

IF (SIZEOF(class_extension) = 1)
THEN
     class_extension_referenced_veps :=
          list_to_set(class_extension[1]\model_class_extension
          .referenced_view_exchange_protocol);
     dictionary_supported_veps := lib\dictionary.supported_vep;
     tmp := (class_extension_referenced_veps <=
          dictionary_supported_veps);
     RETURN(('ISO13584_LIBRARY_CONTENT_SCHEMA.MODEL_CLASS_EXTENSION'
          IN TYPEOF(class_extension[1]))
          AND NOT(tmp));
ELSE
     RETURN(FALSE);
END_IF;

END_FUNCTION; -- referenced_veps_exist_in_supported_veps
(*
```

12.8.28 Referenced_protocols_exist_in_supported_protocols function

The **referenced_protocols_exist_in_supported_protocols** function checks that the **program_reference**s, **representation_references** and **dialogue_ressource**s referenced in a class

extension identified by its **class_bsu cl** reference protocols belonging either to the **library base_protocols** set, or to the **library linked_interfaces** set.

EXPRESS specification:

```
*)
FUNCTION referenced_protocols_exist_in_supported_protocols(
     lib: library; cl: class_BSU): LOGICAL;
LOCAL
     class_extension: SET [0:1] OF content_item :=
          cl\basic_semantic_unit.referenced_by;
END_LOCAL;

IF SIZEOF(class_extension) = 1
THEN
     RETURN(('ISO13584_LIBRARY_CONTENT_SCHEMA' +
          '.MODEL_CLASS_EXTENSION' IN TYPEOF(class_extension))
          AND
          (SIZEOF(QUERY(pr <* class_extension[1]\model_class_extension
          .referenced_external_items | NOT(
          pr\external_item.used_protocol IN
          lib\dictionary.base_protocols)
          AND NOT(pr\external_item.used_protocol IN
          lib.linked_interfaces))) <> 0));
ELSE
     RETURN(FALSE);
END_IF;

END_FUNCTION; -- referenced_protocols_exist_in_supported_protocols
(*
```

12.8.29 Required_properties_are_non_dependent_p_det function

The **required_properties_are_non_dependent_p_det** function checks that when a **functional_model_class_extension** has some **required_item_characteristics**, its **dictionary_definition** is defined as a **fm_class_view_of**, and all the **required_item_characteristics** shall be defined as **non_dependent_P_DET**s.

EXPRESS specification:

```
*)
FUNCTION required_properties_are_non_dependent_p_det(
     fm_class_ext: functional_model_class_extension): LOGICAL;

LOCAL
     res: LOGICAL := TRUE;
     prop: property_bsu;
END_LOCAL;

IF (SIZEOF(fm_class_ext.required_item_characteristics) <> 0)
THEN
     IF (SIZEOF(fm_class_ext\content_item.
```



```
            dictionary_definition.definition) = 1)
        THEN
            IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
                '.FM_CLASS_VIEW_OF') IN
                TYPEOF(fm_class_ext\content_item
                .dictionary_definition.definition[1])
            THEN
                REPEAT i := 1 TO SIZEOF(fm_class_ext.
                                    required_item_characteristics);
                    prop := fm_class_ext.
                        required_item_characteristics[i].property;

                    IF (SIZEOF(prop.definition) = 1)
                    THEN
                        IF NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
                            '.NON_DEPENDENT_P_DET' IN
                            TYPEOF(prop.definition[1]))
                        THEN
                            res := FALSE;
                        END_IF;
                    ELSE
                        res := res AND UNKNOWN;
                    END_IF;

                END_REPEAT;
            ELSE
                res := FALSE;
            END_IF;
        ELSE
            res := UNKNOWN;
        END_IF;
    END_IF;

    RETURN(res);

    END_FUNCTION; -- required_properties_are_non_dependent_p_det
    (*
```

12.8.30 Required_properties_are_imported_properties function

The **required_properties_are_imported_properties** function checks that when a **functional_model_class_extension** has some **required_item_characteristics**, its **dictionary_definition** is defined as a **fm_class_view_of**, and all the **required_item_characteristics** shall belong to the **imported_properties_from_item** attribute of the **fm_class_view_of**.

EXPRESS specification:

```
    *)
    FUNCTION required_properties_are_imported_properties(
        fm_class_ext: functional_model_class_extension): LOGICAL;
```



```
LOCAL
     res: LOGICAL := TRUE;
     prop: property_bsu;
END_LOCAL;

IF (SIZEOF(fm_class_ext.required_item_characteristics) <> 0)
THEN
     IF (SIZEOF(fm_class_ext\content_item.dictionary_definition.
          definition) = 1)
     THEN
          IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
               '.FM_CLASS_VIEW_OF') IN
               TYPEOF(fm_class_ext\content_item
               .dictionary_definition.definition[1])
          THEN
               REPEAT i := 1 TO SIZEOF(fm_class_ext.
                                  required_item_characteristics);
                    prop := fm_class_ext.
                         required_item_characteristics[i].property;

                    IF NOT(prop IN fm_class_ext\content_item.
                         dictionary_definition.definition[1]\
                         fm_class_view_of.
                         imported_properties_from_item)
                    THEN
                         res := FALSE;
                    END_IF;

               END_REPEAT;
          ELSE
               res := FALSE;
          END_IF;
     ELSE
          res := UNKNOWN;
     END_IF;
END_IF;

RETURN(res);

END_FUNCTION; -- required_properties_are_imported_properties
(*
```

12.8.31 Same_order_for_properties function

The **same_order_for_properties** function checks that the properties defined in the **first** list of **property_value**s are the same, and in the same order that the properties defined in the **current** list of **property_value**s. If some couple of **property_value**s are both **context_dependent_property_value**, the **same_order_for_properties** function checks that the properties defined in the **context** attribute of both **context_dependent_property_value**s are the same, and in the same order. The **same_order_for_properties** function returns TRUE when the checking is positive. Otherwise, it returns FALSE.

EXPRESS specification:

```
*)
FUNCTION same_order_for_properties(
     first, current: LIST [1:?] OF property_value): BOOLEAN;

IF SIZEOF(first) = SIZEOF(current)
THEN ;
ELSE  RETURN(FALSE);
END_IF;

REPEAT i := 1 TO SIZEOF(first);
     IF NOT (first[i].prop_def = current[i].prop_def)
     THEN
          RETURN(FALSE);
     END_IF;

     IF (('ISO13584_INSTANCE_RESOURCE_SCHEMA' +
          '.CONTEXT_DEPENDENT_PROPERTY_VALUE')
          IN TYPEOF(first[i]))
     THEN
          IF NOT(('ISO13584_INSTANCE_RESOURCE_SCHEMA' +
               '.CONTEXT_DEPENDENT_PROPERTY_VALUE')
               IN TYPEOF(current[i]))
          THEN
               RETURN(FALSE);
          END_IF;

          IF NOT same_order_for_properties(
               first[i]\context_dependent_property_value.the_context,
               current[i]\context_dependent_property_value.
               the_context)
          THEN
               RETURN(FALSE);
          END_IF;
     END_IF;

     IF (('ISO13584_INSTANCE_RESOURCE_SCHEMA' +
          '.CONTEXT_DEPENDENT_PROPERTY_VALUE')
          IN TYPEOF(current[i]))
     THEN
          IF NOT(('ISO13584_INSTANCE_RESOURCE_SCHEMA' +
               '.CONTEXT_DEPENDENT_PROPERTY_VALUE')
               IN TYPEOF(first[i]))
          THEN
               RETURN(FALSE);
          END_IF;
     END_IF;

     IF (('ISO13584_INSTANCE_RESOURCE_SCHEMA' +
          '.DIC_CLASS_INSTANCE') IN TYPEOF (first[i].its_value))
```



```
      THEN
           IF NOT
                same_order_for_properties(first[i].its_value.properties,
                                          current[i].its_value.properties)
           THEN RETURN (FALSE);
           END_IF;
      END_IF;

END_REPEAT;

RETURN(TRUE);

END_FUNCTION; -- same_order_for_properties
(*
```

12.8.32 All_properties_are_applicable function

The **all_properties_are_applicable** function returns TRUE if all the properties used as **property_value**s to define any **dic_class_instance** of an **explicit_model_class_extension** are applicable. Otherwise, it returns FALSE.

EXPRESS specification:

```
*)
FUNCTION all_properties_are_applicable(
      expl: explicit_model_class_extension): LOGICAL;

LOCAL
      inst: dic_class_instance;
      prop_val: property_value;
      res: LOGICAL := TRUE;
END_LOCAL;

REPEAT i := 1 TO SIZEOF(expl.population);
      inst := expl.population[i];
      REPEAT j := 1 TO SIZEOF(inst.properties);
           prop_val := inst.properties[j];
           res := res AND applicable_properties(expl\
                content_item.dictionary_definition,
                [prop_val.prop_def]);
      END_REPEAT;

END_REPEAT;

RETURN(res);

END_FUNCTION; -- all_properties_are_applicable
(*
```

12.8.33 Required_values_are_non_dependent_p_det function

The **required_values_are_non_dependent_p_det** function checks that when an **explicit_functional_model_class_extension** has some **required_item_values**, its

dictionary_definition is defined as a **fm_class_view_of** and all the **required_item_values** shall be defined as **non_dependent_P_DET**s.

EXPRESS specification:

```
*)
FUNCTION required_values_are_non_dependent_p_det(
     fm_class_ext: explicit_functional_model_class_extension):
     LOGICAL;

LOCAL
     res: LOGICAL := TRUE;
     prop: property_bsu;
END_LOCAL;

IF (SIZEOF(fm_class_ext.required_item_values) <> 0)
THEN
     IF (SIZEOF(fm_class_ext\content_item.dictionary_definition.
          definition) = 1)
     THEN
          IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
               '.FM_CLASS_VIEW_OF') IN
               TYPEOF(fm_class_ext\content_item
               .dictionary_definition.definition[1])
          THEN
               REPEAT i := 1 TO SIZEOF(
                         fm_class_ext.required_item_values);
                    prop := fm_class_ext.required_item_values[i];

                    IF (SIZEOF(prop.definition) = 1)
                    THEN
                         IF NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA'
                              + '.NON_DEPENDENT_P_DET' IN
                              TYPEOF(prop.definition[1]))
                         THEN
                              res := FALSE;
                         END_IF;
                    ELSE
                         res := res AND UNKNOWN;
                    END_IF;
               END_REPEAT;
          ELSE
               res := FALSE;
          END_IF;
     ELSE
          res := UNKNOWN;
     END_IF;
END_IF;

RETURN(res);
```



```
END_FUNCTION; -- required_values_are_non_dependent_p_det
(*
```

12.8.34 Required_values_are_imported_properties function

The **required_values_are_imported_properties** function checks that when an **explicit_functional_model_class_extension** has some **required_item_values**, its **dictionary_definition** is defined as a **fm_class_view_of** and all the **required_item_values** shall belong to the **imported_properties_from_item** attribute of the **fm_class_view_of**.

EXPRESS specification:

```
*)
FUNCTION required_values_are_imported_properties(
     fm_class_ext: explicit_functional_model_class_extension):
     LOGICAL;

LOCAL
     res: LOGICAL := TRUE;
     prop: property_bsu;
END_LOCAL;

IF (SIZEOF(fm_class_ext.required_item_values) <> 0)
THEN
     IF (SIZEOF(fm_class_ext\content_item.dictionary_definition.
          definition) = 1)
     THEN
          IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA' +
               '.FM_CLASS_VIEW_OF') IN
               TYPEOF(fm_class_ext\content_item
               .dictionary_definition.definition[1])
          THEN
               REPEAT i := 1 TO SIZEOF(
                         fm_class_ext.required_item_values);
                    prop := fm_class_ext.required_item_values[i];

                    IF NOT(prop IN fm_class_ext\content_item.
                         dictionary_definition.definition[1]\
                         fm_class_view_of.
                         imported_properties_from_item)
                    THEN
                         res := FALSE;
                    END_IF;
               END_REPEAT;
          ELSE
               res := FALSE;
          END_IF;
     ELSE
          res := UNKNOWN;
     END_IF;
END_IF;
```

```
RETURN(res);

END_FUNCTION; -- required_values_are_imported_properties
(*
```

12.8.35 Data_type_of_BSU function

The **data_type_of_BSU** function computes the **data_type** that defines the final domain of a **property_BSU** or a **data_type_BSU**.

If the **data_type** is associated with **named_type**s, the function recursively traverses their **referred_types** attributes, until arriving either to a **simple_type** or to a **complex_type**. Then the function returns this **simple_type** or **complex_type**.

If a BSU definition is not available, with the result that the function cannot be resolved to a **simple_type** or to a **complex_type**, the function returns an empty set of **data_type**s.

EXPRESS specification:

```
*)
FUNCTION data_type_of_BSU(type_spec: property_or_data_type_BSU):
     SET[0:1] OF data_type;
LOCAL
     res: BOOLEAN := FALSE;
     x: data_type;
END_LOCAL;

IF NOT EXISTS(type_spec)
THEN
     RETURN([]); -- type_spec is indeterminate
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.PROPERTY_BSU' IN
     TYPEOF(type_spec))
THEN
     IF NOT(SIZEOF(type_spec.definition) = 0)
     THEN
          x := type_spec.definition[1]\property_DET.domain;
          res := TRUE;
     END_IF;
ELSE
     IF NOT(SIZEOF(type_spec.definition) = 0)
     THEN
          x := type_spec.definition[1]
               \data_type_element.type_definition;
          res := TRUE;
     END_IF;
END_IF;

IF NOT(res)
THEN
     RETURN([]);
```



```
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NAMED_TYPE' IN TYPEOF(x))
THEN
     IF NOT(SIZEOF(x\named_type.referred_type.definition) = 0)
     THEN
          RETURN(data_type_of_BSU(x\named_type.referred_type));
     ELSE
          RETURN([]);
     END_IF;
ELSE
     RETURN([x]);
END_IF;

END_FUNCTION; -- data_type_of_BSU
(*
```

12.8.36 Presentation_unit_is_correct function

The function **presentation_unit_is_correct** checks if the dictionary-defined data type of a **prop property_BSU** is compatible with the **to_unit unit**.

If the result may not be computed because some BSU definitions are not available, the function returns UNKNOWN. If the data type of the **prop property_BSU** is not associated with a unit, the function returns FALSE.

EXPRESS specification:

```
*)
FUNCTION presentation_unit_is_correct(prop: property_BSU;
     to_unit: unit): LOGICAL;
LOCAL
     prop_domain: data_type;
     prop_typeof: SET OF STRING := [];
END_LOCAL;

IF (SIZEOF(prop\basic_semantic_unit.definition) = 0)
THEN
     RETURN(UNKNOWN);
END_IF;

prop_typeof := data_type_typeof(prop);

IF (prop_typeof = []) -- some DET not present
THEN
     RETURN(UNKNOWN);
END_IF;

prop_domain := data_type_of_BSU(prop)[1]; -- not empty

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_MEASURE_TYPE' IN
     TYPEOF(prop_domain))
THEN
```



```
      RETURN(derive_dimensional_exponents(to_unit) =
           derive_dimensional_exponents(prop_domain\
           int_measure_type.unit.structured_representation));
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_MEASURE_TYPE' IN
      TYPEOF(prop_domain))
THEN
      RETURN(derive_dimensional_exponents(to_unit) =
           derive_dimensional_exponents(prop_domain\
           real_measure_type.unit.structured_representation));
END_IF;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.LEVEL_TYPE' IN
      TYPEOF(prop_domain))
THEN
      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_MEASURE_TYPE' IN
           TYPEOF(prop_domain\level_type.value_type))
      THEN
           RETURN(derive_dimensional_exponents(to_unit) =
                derive_dimensional_exponents(
                prop_domain\level_type.value_type
                .unit.structured_representation));
      END_IF;
      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_MEASURE_TYPE' IN
           TYPEOF(prop_domain\level_type.value_type))
      THEN
           RETURN(derive_dimensional_exponents(to_unit) =
                derive_dimensional_exponents(
                prop_domain\level_type.value_type
                .unit.structured_representation));
      END_IF;
END_IF;

RETURN(FALSE);

END_FUNCTION; -- presentation_unit_is_correct
(*
```

12.8.37 Exists_representation_for_instanciable_view function

The **exists_representation_for_instanciable_view** function checks that when the functional view class referenced by the **ext explicit_functional_model_class_extension** is not a **non_instanciable_view_class**, then the **referenced_representation** exists and its data type is either a **representation_type**, a **representation_reference_type** or a **program_reference_type**, and that, if the functional view class referenced by the **ext explicit_functional_model_class_extension** is a **non_instanciable_view_class**, then the **referenced_representation** does not exist.

The function returns UNKNOWN if the functional view class or the functional model class dictionary definitions are not available.

EXPRESS specification:

```
*)
FUNCTION exists_representation_for_instanciable_view(
     ext: explicit_functional_model_class_extension): LOGICAL;
LOCAL
     cpt: INTEGER := 0;
     prop_val: property_value;
END_LOCAL;

IF (SIZEOF(ext.dictionary_definition.definition) = 1)
THEN
     IF (SIZEOF(ext.dictionary_definition.definition[1]\
          abstract_functional_model_class.created_view.definition) = 1)
     THEN
          IF (NOT('ISO13584_EXTENDED_DICTIONARY_SCHEMA.' +
               'NON_INSTANTIABLE_FUNCTIONAL_VIEW_CLASS' IN TYPEOF(
               ext.dictionary_definition.definition[1]\
               abstract_functional_model_class.created_view.
               definition[1])))
          THEN
               RETURN(EXISTS(ext.referenced_representation)
                         AND
                         (('ISO13584_EXTERNAL_FILE_SCHEMA.' +
                         'PROGRAM_REFERENCE' IN
                         data_type_type_name(
                         ext.referenced_representation))
                         OR ('ISO13584_EXTERNAL_FILE_SCHEMA.' +
                         'REPRESENTATION_REFERENCE' IN
                         data_type_type_name(
                         ext.referenced_representation))
                         OR ('REPRESENTATION_SCHEMA.' +
                         'REPRESENTATION' IN
                         data_type_type_name(
                         ext.referenced_representation)))) ;
          ELSE
               RETURN(NOT EXISTS(ext.referenced_representation));
          END_IF;
     ELSE
          RETURN(UNKNOWN);
     END_IF;
ELSE
     RETURN(UNKNOWN);
END_IF;

END_FUNCTION; -- exists_representation_for_instanciable_view
(*
```

12.8.38 Is_provided_once_property_value function

The **is_provided_once_property_value** function checks that the **prop property_or_data_type_BSU** is a **property_BSU** that is associated with exactly one value in the **a_model dic_class_instance**. It

returns FALSE if the **prop property_or_data_type_BSU** is a **data_type_BSU** or if this **property_BSU** is not associated with exactly one value.

EXPRESS specification:

```
*)
FUNCTION is_provided_once_property_value(
          a_model: dic_class_instance;
          prop: property_or_data_type_BSU): BOOLEAN;
LOCAL
     cpt: INTEGER := 0;
END_LOCAL;

IF NOT (('ISO13584_IEC61360_DICTIONARY_SCHEMA.' +
     'PROPERTY_BSU') IN TYPEOF (prop))
THEN
     RETURN(FALSE);
END_IF;

REPEAT i := 1 TO SIZEOF(a_model.properties);
     IF ((prop = a_model.properties[i].prop_def) AND
          EXISTS(a_model.properties[i].its_value))
     THEN
          cpt := cpt + 1;
     END_IF;
END_REPEAT;
RETURN (cpt = 1);
END_FUNCTION; -- is_provided_once_property_value
(*
```

12.8.39 Number_of_instance_representations

The **number_of_instance_representations** function computes the number of **property_BSU**s referenced in the **a_model properties** attribute of which the data type is a **representation_type,** a **representation_reference_type** or a **program_reference_type**. The function returns indeterminate if the **dictionary_definition** of some **property_BSU** is not available.

EXPRESS specification:

```
*)
FUNCTION number_of_instance_representations(
          a_model: dic_class_instance): INTEGER;
LOCAL
     cpt: INTEGER := 0;
     prop_val: property_value;
END_LOCAL;

REPEAT i := 1 TO SIZEOF(a_model.properties);
     prop_val := a_model.properties[i];
     IF data_type_typeof(prop_val.prop_def) = []
     THEN
```



```
            RETURN(?);
        END_IF;
        IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.' +
            'ENTITY_INSTANCE_TYPE' IN
            data_type_typeof(prop_val.prop_def))
        THEN
            IF (('ISO13584_EXTERNAL_FILE_SCHEMA.' +
                'PROGRAM_REFERENCE' IN
                data_type_type_name(prop_val.prop_def))
                OR ('ISO13584_EXTERNAL_FILE_SCHEMA.' +
                'REPRESENTATION_REFERENCE' IN
                data_type_type_name(prop_val.prop_def))
                OR ('REPRESENTATION_SCHEMA.' +
                'REPRESENTATION' IN
                data_type_type_name(prop_val.prop_def)))
            THEN
                cpt := cpt + 1;
            END_IF;
        END_IF;
    END_REPEAT;

    RETURN(cpt);

    END_FUNCTION; -- number_of_instance_representations
    (*
```

12.8.40 Correct_parameters_for_explicit_program function

The **correct_parameters_for_explicit_program** function checks that the **prop property_BSU**, the data type of which is **program_reference_type**, is associated with a **program_reference** of which both the **out_parameters** and **inout_parameters** lists are empty, and of which all the values of the **in_parameters** attributes are **property_BSU**s that are associated with one value in the **a_model** instance.

EXPRESS specification:

```
    *)
    FUNCTION correct_parameters_for_explicit_program(
            a_model: dic_class_instance;
            prop: property_BSU): BOOLEAN;

    IF NOT('ISO13584_EXTENDED_DICTIONARY_SCHEMA.' +
            'PROGRAM_REFERENCE_TYPE' IN
            data_type_typeof(prop))
    THEN
        RETURN (FALSE); --not a program
    END_IF;

    REPEAT i := 1 TO SIZEOF(a_model.properties);
        IF (prop = a_model.properties[i].prop_def)
        THEN --characteristics of the program_reference
            IF EXISTS(a_model.properties[i].its_value) AND
                ('ISO13584_EXTERNAL_FILE_SCHEMA.' +
```

```
                'PROGRAM_REFERENCE' IN
                TYPEOF(a_model.properties[i].its_value)) AND
                (SIZEOF(a_model.properties[i].its_value.
                out_parameters) = 0)
                AND (SIZEOF(a_model.properties[i].its_value.
                inout_parameters) = 0) AND
                (QUERY (in_p <* a_model.properties[i].its_value.
                in_parameters
                | NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA.' +
                'PROPERTY_BSU' IN TYPEOF (in_p))
                OR NOT is_provided_once_property_value(
                a_model, in_p)) = [])
           THEN
                RETURN(TRUE); --correct program_reference
           ELSE
                RETURN(FALSE); --not correct program_reference
           END_IF;
      END_IF;
END_REPEAT;

RETURN(FALSE); -- program_reference was not found

END_FUNCTION; -- correct_parameters_for_explicit_program
(*
```

12.8.41 Get_dic_item_instances_from_required_item_properties function

The **get_dic_item_instances_from_required_item_properties** function retrieves a set of tuples of values. Each tuple represents a subset of the property values used to specify one of the **dic_item_instance** of the **explicit_item_class_extension** whose dictionary definition is referenced from the **cl.definition[1]\fm_class_view_of.view_of** attribute. Each value of this tuple corresponds to one property of the **required_item_values** attribute of the **cl.referenced_by[1]\explicit_functional_model_class_extension** entity. The tuple order is computed according to the order defined in the **cl.definition[1]\fm_class_view_of.imported_properties_from_item** list attribute.

This function returns an empty set if the **cl** dictionary definition or content specification is not available, or if the **cl** dictionary definition data type is not the **fm_class_view_of** data type, or if the **cl** content specification data type is not the **explicit_functional_model_class_extension** data type. It also returns an empty set if the library specification of the referenced item class is not available or, if available, is not the **explicit_item_class_extension** data type.

EXPRESS specification:

```
*)
FUNCTION get_dic_item_instances_from_required_item_properties(
      cl: class_bsu): SET OF LIST OF primitive_value;
LOCAL
      required_props: LIST OF property_bsu := [];
      definition: fm_class_view_of;
      content: explicit_functional_model_class_extension;
      item_bsu: class_bsu;
```



```
     item_content: explicit_item_class_extension;
     result: SET OF LIST OF primitive_value := [];
END_LOCAL;

IF (SIZEOF(cl.definition) = 1)
THEN
     IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA.FM_CLASS_VIEW_OF'
          IN TYPEOF(cl.definition[1]))
     THEN
          definition := cl.definition[1];
          item_bsu := definition.view_of;
     ELSE
          RETURN([]);
     END_IF;
ELSE
     RETURN([]);
END_IF;

IF (SIZEOF(cl.referenced_by) = 1)
THEN
     IF ('ISO13584_LIBRARY_CONTENT_SCHEMA.' +
          'EXPLICIT_FUNCTIONAL_MODEL_CLASS_EXTENSION'
          IN TYPEOF(cl.referenced_by[1]))
     THEN
          content := cl.referenced_by[1];
     ELSE
          RETURN([]);
     END_IF;
ELSE
     RETURN([]);
END_IF;

IF (SIZEOF(item_bsu.referenced_by) = 1)
THEN
     IF ('ISO13584_LIBRARY_CONTENT_SCHEMA.' +
          'EXPLICIT_MODEL_CLASS_EXTENSION'
          IN TYPEOF(item_bsu.referenced_by[1]))
     THEN
          item_content := item_bsu.referenced_by[1];
     ELSE
          RETURN([]);
     END_IF;
ELSE
     RETURN([]);
END_IF;

required_props := get_list_of_required_properties(definition,
     content.required_item_values);
result := properties_projection_on_population(required_props,
     item_content.population);

RETURN(result);
```

```
END_FUNCTION; -- get_dic_item_instances_from_imported_item_properties
(*
```

12.8.42 Get_list_of_required_properties function

The **get_list_of_required_properties** function computes an ordered list of **property_bsu** from a set of **property_bsu**s. If a **property_bsu** of the **cl.imported_properties_from_item** does not belong to the **required_properties** set, it is not included in the output list. The output list order is defined according to the underlying order of the **cl.imported_properties_from_item** list attribute.

EXPRESS specification:

```
*)
FUNCTION get_list_of_required_properties(cl: fm_class_view_of;
     required_properties: SET OF property_bsu)
     : LIST OF property_bsu;
LOCAL
     result: LIST OF property_bsu := [];
END_LOCAL;

REPEAT i := 1 TO SIZEOF(cl.imported_properties_from_item);
     IF (cl.imported_properties_from_item[i] IN required_properties)
     THEN
          result := result + cl.imported_properties_from_item[i];
     END_IF;
END_REPEAT;

RETURN(result);

END_FUNCTION; -- get_list_of_required_properties
(*
```

12.8.43 Properties_projection_on_population function

The **properties_projection_on_population** function retrieves a set of tuples of **primitive_value**s from a list of **dic_class_instance**s. It applies a projection of the **properties** list on each **dic_class_instance.properties** list contained in **population**.

EXPRESS specification:

```
*)
FUNCTION properties_projection_on_population(
     properties: LIST OF property_bsu;
     population: LIST OF dic_class_instance)
     : SET OF LIST OF primitive_value;
LOCAL
     result: SET OF LIST OF INTEGER := [];
     tuple: LIST OF INTEGER := [];
END_LOCAL;

REPEAT i := 1 TO SIZEOF(population);
```



```
        REPEAT j := 1 TO SIZEOF(properties);
            REPEAT k := 1 TO SIZEOF(population[i].properties);
                IF (population[i].properties[k].prop_def =
                    properties[j])
                THEN
                    tuple := tuple +
                        population[i].properties[k].its_value;
                END_IF;
            END_REPEAT;
        END_REPEAT;
        result := result + tuple;
        tuple := [];
    END_REPEAT;

    RETURN(result);

    END_FUNCTION; -- properties_projection_on_population
    (*
```

12.8.44 All_views_available_for_components function

The **all_views_available_for_components** function checks that the set of **dic_f_model_instance** specifying the given **cl.referenced_by[1]\explicit_functional_model_class_extension** shall describe, for each **dic_item_instance** specifying an **explicit_model_class_extension** whose dictionary definition is referenced from the **view_of** attribute of the **fm_class_view_of** dictionary definition of that **explicit_functional_model_class_extension**, the complete set of declared views of the functional model.

This function returns UNKNOWN if the **cl** dictionary definition or content specification is not available, or if the **cl** dictionary definition data type is not the **fm_class_view_of** data type, or if the **cl** content specification data type is not the **explicit_functional_model_class_extension** data type.

EXPRESS specification:

```
    *)
    FUNCTION all_views_available_for_components(cl: class_bsu): LOGICAL;
    LOCAL
        components_views: SET OF LIST OF INTEGER := [];
        component_view: LIST OF INTEGER := [];
        components: SET OF LIST OF INTEGER := [];
        declared_views: SET OF LIST OF INTEGER := [];
        definition: fm_class_view_of;
        content: explicit_functional_model_class_extension;
    END_LOCAL;

    IF (SIZEOF(cl.definition) = 1)
    THEN
        IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA.FM_CLASS_VIEW_OF'
            IN TYPEOF(cl.definition[1]))
        THEN
            definition := cl.definition[1];
        ELSE
            RETURN(UNKNOWN);
```



```
      END_IF;
ELSE
      RETURN(UNKNOWN);
END_IF;

IF (SIZEOF(cl.referenced_by) = 1)
THEN
      IF ('ISO13584_LIBRARY_CONTENT_SCHEMA.' +
           'EXPLICIT_FUNCTIONAL_MODEL_CLASS_EXTENSION'
           IN TYPEOF(cl.referenced_by[1]))
      THEN
           content := cl.referenced_by[1];
      ELSE
           RETURN(UNKNOWN);
      END_IF;
ELSE
      RETURN(UNKNOWN);
END_IF;
-- id x vcvs in model
components_views := available_components_views(cl);
-- id in item
components := get_dic_item_instances_from_required_item_properties(cl);
-- vcv tuples in declaration
declared_views := declared_created_views(cl);

IF (SIZEOF(components) <> 0)
THEN
      REPEAT i := 1 TO SIZEOF(components);
           REPEAT j := 1 TO SIZEOF(declared_views);
                component_view := components[i] + declared_views[j];
                IF NOT(component_view IN components_views)
                THEN
                     RETURN(FALSE);
                END_IF;
           END_REPEAT;
      END_REPEAT;

      RETURN(TRUE);
ELSE
      RETURN(UNKNOWN);
END_IF;

END_FUNCTION; -- all_views_available_for_components
(*
```

12.8.45 Available_components_views function

The **available_components_views** function returns a set of tuples of values corresponding to the projection of the union of the re-ordered **cl.referenced_by[1]\explicit_functional_model_class_extension.required_item_values** set and the list of available view control variables as returned by the **functional_view_v_c_v** function, on the

list of **dic_f_model_instance**s used for specifying the **cl.referenced_by[1]\explicit_functional_model_class_extension**.

This function returns an empty set if the **cl** dictionary definition or content specification is not available, or if the **cl** dictionary definition data type is not the **fm_class_view_of** data type, or if the **cl** content specification data type is not the **explicit_functional_model_class_extension** data type.

EXPRESS specification:

```
*)
FUNCTION available_components_views(cl: class_bsu): SET OF LIST OF
INTEGER;
LOCAL
     required_props: LIST OF property_bsu := [];
     vcvs: LIST OF property_bsu := [];
     view_properties: LIST OF property_bsu := [];
     definition: fm_class_view_of;
     content: explicit_functional_model_class_extension;
     result: SET OF LIST OF INTEGER := [];
END_LOCAL;

     IF (SIZEOF(cl.definition) = 1)
     THEN
          IF ('ISO13584_EXTENDED_DICTIONARY_SCHEMA.FM_CLASS_VIEW_OF'
               IN TYPEOF(cl.definition[1]))
          THEN
               definition := cl.definition[1];
     ELSE
          RETURN([]);
     END_IF;
ELSE
     RETURN([]);
END_IF;

IF (SIZEOF(cl.referenced_by) = 1)
THEN
     IF ('ISO13584_LIBRARY_CONTENT_SCHEMA.' +
          'EXPLICIT_FUNCTIONAL_MODEL_CLASS_EXTENSION'
          IN TYPEOF(cl.referenced_by[1]))
     THEN
          content := cl.referenced_by[1];
     ELSE
          RETURN([]);
     END_IF;
ELSE
     RETURN([]);
END_IF;

vcvs := functional_view_v_c_v(definition\
     abstract_functional_model_class.created_view);

IF (SIZEOF(vcvs) <> 0)
```

```
THEN
     required_props := get_list_of_required_properties(definition,
          content.required_item_values);
     view_properties := required_props + vcvs;
     result := properties_projection_on_population(view_properties,
          content.population);
END_IF;

RETURN(result);

END_FUNCTION; -- available_components_views
(*
```

12.8.46 All_view_control_variables_belong_to_each_view function

The **all_view_control_variables_belong_to_each_view** function returns TRUE if all the view control variables that are defined in the functional view class referenced by the **class_ext** dictionary definition are all used for describing each **dic_class_instance** belonging to the **SELF\model_class_extension.population** list. Otherwise, it returns FALSE.

This function returns UNKNOWN if either the **class_ext** dictionary definition is not available, or if the referenced functional view class definition is not available.

EXPRESS specification:

```
*)

FUNCTION all_view_control_variables_belong_to_each_view(
     class_ext: explicit_model_class_extension): LOGICAL;
LOCAL
     created_view: class_bsu;
     cl: class_bsu;
     vcvs: SET OF property_bsu;
     i, max: INTEGER;
     result: LOGICAL := TRUE;
END_LOCAL;

cl := class_ext\content_item.dictionary_definition;

IF (SIZEOF(cl.definition) = 1)
THEN
     created_view := cl.definition[1]\
          abstract_functional_model_class.created_view;
     vcvs := list_to_set(functional_view_v_c_v(created_view));

     IF (SIZEOF(created_view.definition) = 1)
     THEN
          IF (SIZEOF(vcvs) > 0)
          THEN
               i := 1;
               max := SIZEOF(class_ext.population);
               REPEAT WHILE((i <= max) AND (result));
```



```
                    result :=
                    check_all_view_control_variables_belong_to_view(
                         vcvs, class_ext.population[i]);
                    i := i + 1;
               END_REPEAT;

          ELSE
               result := TRUE;
          END_IF;

     ELSE
          result := UNKNOWN;
     END_IF;

     RETURN(result);
ELSE
     RETURN(UNKNOWN);
END_IF;

END_FUNCTION; -- all_view_control_variables_belong_to_each_view
(*
```

12.8.47 Check_all_view_control_variables_belong_to_view function

The **check_all_view_control_variables_belong_to_view** function checks that all the given **vcvs** are used to defined the **a_view dic_f_model_instance**.

EXPRESS specification:

```
*)

FUNCTION check_all_view_control_variables_belong_to_view(
     vcvs: SET OF property_bsu;
     a_view: dic_f_model_instance): LOGICAL;

LOCAL
     used_properties: SET OF property_bsu := [];
END_LOCAL;

REPEAT i := 1 TO SIZEOF(a_view.properties);
     used_properties := used_properties +
          a_view.properties[i].prop_def;
END_REPEAT;

RETURN(vcvs <= used_properties);

END_FUNCTION; -- check_all_view_control_variables_belong_to_view
(*
```

12.8.48 All_vcvs_belong_to_instance_identification function

The **all_vcvs_belong_to_instance_identification** function returns TRUE if all the view control variables that are used to characterize the

class_ext.definition[1]\abstract_functional_model_class.created_view functional view class are referenced into the **class_ext\explicit_model_class_extension.instance_identification** set. Otherwise it returns FALSE.

This function returns UNKNOWN if either the **class_ext** dictionary definition is not available, or if the referenced functional view class definition is not available.

EXPRESS specification:

```
*)

FUNCTION all_vcvs_belong_to_instance_identification(
     class_ext: explicit_model_class_extension): LOGICAL;
LOCAL
     vcvs: SET OF property_bsu := [];
     created_view: class_bsu;
     cl: class_bsu;
END_LOCAL;

cl := class_ext\content_item.dictionary_definition;

IF (SIZEOF(cl.definition) = 1)
THEN
     created_view := cl.definition[1]\
          abstract_functional_model_class.created_view;
     vcvs := list_to_set(functional_view_v_c_v(created_view));

     IF (SIZEOF(created_view.definition) = 1)
     THEN
          RETURN(vcvs <= list_to_set(class_ext\
               explicit_model_class_extension.
               instance_identification));
     ELSE
          RETURN(UNKNOWN);
     END_IF;

ELSE
     RETURN(UNKNOWN);
END_IF;

END_FUNCTION; -- vcvs_and_required_properties_belong_to_identification
(*
```

12.8.49 Same_string_values_translations_for_class_extension function

The **same_string_values_translations_for_class_extension** function returns TRUE if, for each instance that defines the **explicit_model_class_extension** population, any first **property_value** of which value is a **translated_string_value** (if it exists) is defined using the same language(s). Otherwise, it returns FALSE. It returns UNKNOWN if no **translated_string_value** is used for defining all the **population** instances.

NOTE This function is not in charge of checking that all the **translated_string_value**s used for defining **property_value**s of each instance are all given in the same language(s). This checking is performed in the **dic_class_instance** entity data type specification.

EXPRESS specification:

```
*)
FUNCTION same_string_values_translations_for_class_extension(
     class_ext: explicit_model_class_extension): LOGICAL;
LOCAL
     comp: SET OF translated_string_value := [];
     translated_property_values: LIST OF property_value := [];
END_LOCAL;

REPEAT i := 1 TO SIZEOF(class_ext.population);
     translated_property_values := QUERY(prop_val <*
          class_ext.population[i].properties |
          'ISO13584_INSTANCE_RESOURCE_SCHEMA.' +
          'TRANSLATED_STRING_VALUE' IN
          TYPEOF(prop_val.its_value));
     IF (SIZEOF(translated_property_values) <> 0)
     THEN
          comp := comp + translated_property_values[1].its_value;
     END_IF;
END_REPEAT;

RETURN(same_translations(comp));

END_FUNCTION; -- same_string_values_translations_for_class_extension
(*

*)
END_SCHEMA; -- ISO13584_library_content_schema
(*
```

13 ISO13584_external_file_schema

This clause defines the requirement for the **ISO13584_external_file_schema**. The following EXPRESS declaration introduces the **ISO13584_external_file_schema** block and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA ISO13584_external_file_schema;

REFERENCE FROM ISO13584_IEC61360_dictionary_schema
     (basic_semantic_unit,
     class_BSU,
     class_related_BSU,
     content_item,
```



```
        definition_available_implies,
        graphics,
        item_names,
        list_to_set,
        revision_type,
        supplier_BSU,
        supplier_element,
        supplier_related_BSU,
        version_len);

    REFERENCE FROM ISO13584_IEC61360_language_resource_schema
        (global_language_assignment,
        language_code,
        present_translations);

    REFERENCE FROM ISO13584_instance_resource_schema
        (property_or_data_type_BSU);

    REFERENCE FROM ISO13584_extended_dictionary_schema
        (absolute_url_type,
        document_BSU,
        program_library_BSU);

    REFERENCE FROM ISO13584_library_content_schema
        (model_class_extension);

    REFERENCE FROM support_resource_schema
        (identifier,
        label);

    REFERENCE FROM person_organization_schema
        (organization);

    REFERENCE FROM measure_schema
        (length_measure,
        length_measure_with_unit);

    (*
```

NOTE The schemas referenced above can be found in the following documents:
ISO13584_IEC61360_dictionary_schema IEC 61360-2
(which is duplicated for convenience in informative annex D of ISO 13584-42),
ISO13584_IEC61360_language_resource_schema IEC 61360-2
(which is duplicated for convenience in informative annex D of ISO 13584-42),
ISO13584_instance_resource_schema This part of ISO 13584,
ISO13584_extended_dictionary_schema This part of ISO 13584,
ISO13584_library_content_schema This part of ISO 13584,
support_resource_schema ISO 10303-41,
person_organization_schema ISO 10303-41,
measure_schema ISO 10303-41.

13.1 Introduction to the ISO13584_external_file_schema

A library exchange context includes a library delivery file conforming to one conformance class of a library integrated information model and several library external files. The library delivery file specifies both the structure and the content of the supplier library. The library external files, provide for exchanging associated information whose EXPRESS information models are not described in ISO 13584. They may be described by other standards, using either the EXPRESS information modelling language or other specification techniques. Their usage, in a supplier library, shall be allowed either by the conformance class of the library integrated information model referenced by the library delivery file, or in one of the view exchange protocol conformance classes referenced by the library delivery file.

The information contained in library external files is represented within the library delivery file using **external_item** entities. Such entities reference the external files used to exchange this content and the protocol that shall be used to process them. Three categories of **external_item**s are defined:

— **dictionary_external_item**s are associated with a BSU for which they provide the content. They can only be referenced through their BSU, and their life cycle is independent from any class extension. In particular, they may be associated with a class that is only described by dictionary data;

— **class_extension_external_item**s provide for describing aspects of the extensions of classes. They are directly referenced from the entities that describe the **model_class_extension**. They shall be provided whenever a new version of a **model_class_extension** is provided and their life cycles are the same as the **model_class_extension** to which they belong.

— **property_value_external_item**s provide for defining item properties the value of which are an external item.

The planning model in Figure 8 shows the different categories of library external files defined in this part of ISO 13584.

NOTE 1 This planning model uses the EXPRESS-G graphical notations for the EXPRESS language, but, for clarification of the diagram, some of the relationships that are defined in the EXPRESS model are omitted, and some inter-schema references are not represented.

A **program_library_content** is a set of programs that may be delivered as part of a library exchange context, and that shall be linked with an API, defined by a **program_protocol**, to define a new higher level API, modeled by a **linked_interface_program_protocol** entity. Then, such a new API may be referenced from a program associated with a library class by means of a **linked_interface_program_protocol** entity instance.

EXAMPLE ISO 13584-31 defines an API that may be used, together with a file exchange format, to define a **program_protocol**. An upper layer may be defined on top of this API, by means of a **program_library**, to define a new **program_protocol** that is a **linked_interface_program_protocol**. Such a **program_protocol** might be useful to ensure the compatibility with some pre-existing APIs, for instance the API defined by the German Standard DIN V 66304.

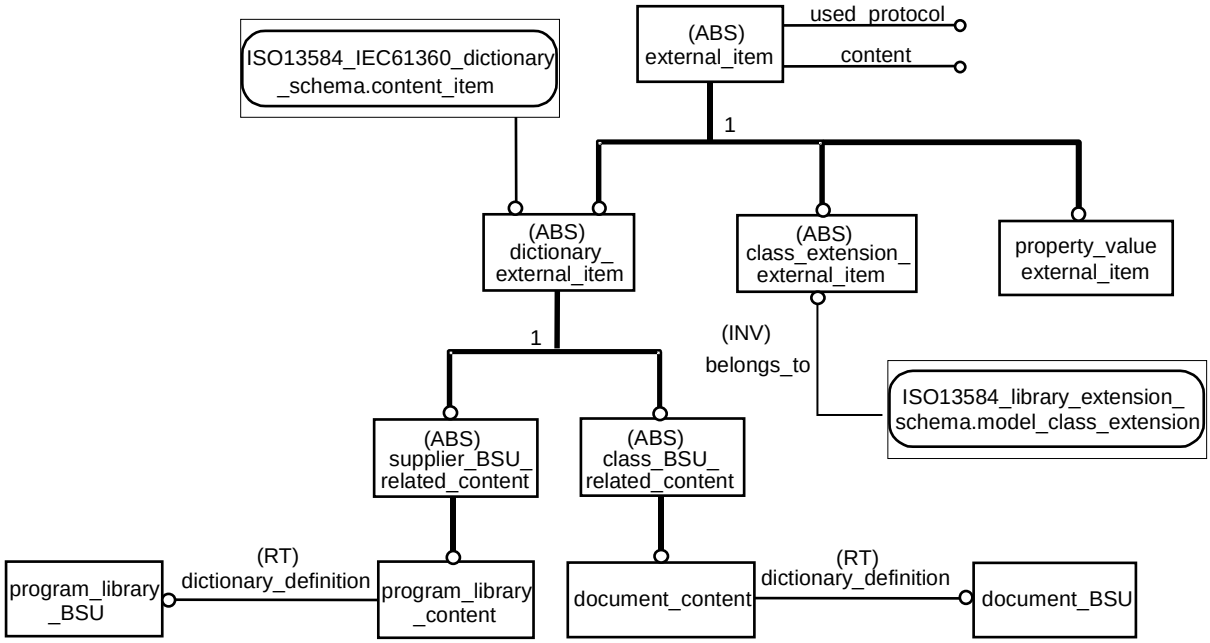


Figure 8 – External_item planning model

A **program_library_content** is a set of programs that may be delivered as part of a library exchange context, and that shall be linked with an API, defined by a **program_protocol**, to define a new higher level API, modeled by a **linked_interface_program_protocol** entity. Then, such a new API may be referenced from a program associated with a library class by means of a **linked_interface_program_protocol** entity instance.

EXAMPLE ISO 13584-31 defines an API that may be used, together with a file exchange format, to define a **program_protocol**. An upper layer may be defined on top of this API, by means of a **program_library**, to define a new **program_protocol** that is a **linked_interface_program_protocol**. Such a **program_protocol** might be useful to ensure the compatibility with some pre-existing APIs, for instance the API defined by the German Standard DIN V 66304.

A **program_library_content**:

— corresponds to a **program_library_BSU** that identifies it and enables to reference it;

— refers to a **program_protocol** that defines the API to which it is intended to be linked;

— is associated with library external file(s) that constitutes its content.

A **document_content** is the content of a document identified by a **document_BSU** and described by a **document** entity. It contains information defined at the discretion of the supplier and accessible to the user through a set of names. In particular, within the context of a class, the user shall be informed about the documents applicable to the class and shall be able to request display of them.

A **document_content**:

— is associated with a **document_BSU** that identifies it and enables to reference it;

— refers to an **external_file_protocol** that specifies how it shall be processed;

— may be provided in more than one language;

— refers to the library external file(s) that constitutes its content.

A **class_extension_external_item** provides some information about the extension of a class by means of external file(s). It is not associated with a BSU, and it shall only be referred to from the entities that describe the extension of a class.

Each **class_extension_external_item**:

— is associated with a **model_class_extension** that defines its scope and its life cycle;

— is associated with an **external_file_protocol** that specifies how it shall be processed;

— is described by an entity (a **message** or an **illustration**) that specifies its nature and, therefore, the transformation, if any, it must undergo in the compilation phase;

— refers to the library external file(s) that constitutes its content.

A **property_value_external_item**:

— is a **PLIB_entity_instance_value** and thus a **primitive_value**.

— it may constitute the value of any property the data type of which is defined as an **entity_instance_type**, the **type_name** attribute of which contains 'ISO13584_EXTERNAL_FILE_SCHEMA.PROPERTY_VALUE_EXTERNAL_ITEM'.

NOTE 2 **PLIB_entity_instance_value** and **primitive_value** are defined in the **ISO13584_instance_resource_schema** documented in clause 6.

NOTE 3 **entity_instance_type** is defined in the **ISO13584_IEC61360_dictionary_schema** documented in ISO 13584-42:1998.

These files shall be processed by the LMS of the library end-user, depending on the **external_file_protocol**s supported on this LMS.

The **ISO13584_external_file_schema** models:

— the mechanisms that provide reference to library external files,

— the specification of the way a library external file shall be processed,

— the resource constructs that specify the role of each library external file,

— the mechanisms that enable a library end-user to navigate between documents,

— the mechanisms that enable a library end-user to navigate from documents to library classes.

The **ISO13584_external_file_schema** does not model:

— the information models of the library external files.

13.2 Fundamental concepts and assumptions for the ISO13584_external_file_schema

13.2.1 Representations of items

The goal of this International Standard is to provide for the exchange of any category of representation of library items. Taking into consideration that several international, national or de facto standards already enable the description of various categories of representations of various items, a library

delivery file shall be able to refer to external files, conforming to other standards, to describe the different representation categories, for the items belonging to this library.

NOTE VHDL and ISO 10303-203, are examples of International Standards that provide for the description of two different categories of representation for parts.

13.2.2 Explicit and implicit description of item representations

For one category of representation, the set of the representations of all the instances of an item class may be modelled either explicitly or implicitly. If it is modelled explicitly, each item representation is described as a set of data. If it is modelled implicitly, there is an algorithm that shall be triggered and provided with parameter values to generate each item representation.

In the **ISO13584_external_file_schema**, explicit representations are referenced by **representation_reference** entities, implicit representations are referenced by **program_reference** entities.

A **program_reference** is associated with a **program_protocol** and with parameters. When a supplier library is processed on the library end-user site, processing of a **program_reference** shall consist of verifying the availability of the **program_protocol** and then, according to the **program_status** of the program, compiling the content of the library external file referenced by the **program_reference** and/or linking this content. In a latter stage, running will require evaluation of its parameters and a call to run it. This call shall be issued by the LMS, on request from the user.

NOTE A **program_status** is defined in 13.4.10. It is used in 13.5.6 to specify whether a program is in source code or compiled.

A **representation_reference** is associated with a **data_protocol**. When a supplier library is processed on the library end-user site, processing of a **representation_reference** shall consist of verifying the availability of the **data_protocol** and then, if appropriate, converting the content to another format, for instance the native format of the library end-user LMS.

13.2.3 Support of user dialogue

A LMS shall be able to display various informative items that may be either graphic or textual. To enable an automatic integration of a supplier library in a user library, these informative items shall come from the library data supplier. These informative items are modelled as **dialogue_resource**s.

dialogue_resource is a subtype of a **class_extension_external_item**. It represents a piece of information that shall be automatically displayed by the LMS in some particular work contexts. It can be either graphic, an **illustration**, or textual, a **message**. A **dialogue_resource** may be provided in more than one language and it may consist of several library external files.

13.2.4 Http files storage

One particular **data_protocol** that may be supported by a library integrated information model or a view exchange protocol is the http protocol. This **external_file_protocol** is modelled by the **http_protocol** entity. Library external files that are associated with the **http_protocol** are modelled as **http_file**s. A **http_file** is associated, in particular, with a **http_file_name_type**, that specifies the name that shall be associated to this file on the user local Internet server (see clause 5.6.2), and with an oprtional **http_class_directory** that specifies the name of the directory where this file shall be stored on the user local Internet server. Each **http_class_directory** refers to one **class** and each class that is associated with **http_file**s is referenced by exactly one **http_class_directory**.

The rule in the following subclause shall be satisfied by any LMS implementation that claims conformance to ISO 13584.

13.2.4.1 Http_file storage rule

All the **class_extension_external_item**s delivered by one particular library data supplier as **http_file**s shall be stored in a directory hierarchy that contains only two levels:

— the root of this hierarchy is a directory whose name is the **supplier_code** of this library data supplier. An implementation is allowed to abbreviate the directory names if the software does not support the length specified in ISO 13584:42 for **supplier_code** for naming directories;

NOTE 1 The length specified in ISO 13584-42 for **supplier_code** is 70 characters.

NOTE 2 The methods of abbreviating names for the root directories associated with the various library data suppliers are not specified in this International Standard and may be implementation dependent.

— each leaf of this hierarchy is a directory whose name is defined by the **name** attribute of the **http_class_directory** entity that represents the directory to be created in the user file management system in the library delivery file; the class associated with the directory is defined by the **class** attribute of the **http_class_directory** entity that represents it.

This rule ensures that when several **http_file**s associated with the same supplier library are delivered by a library data supplier, the library data supplier knows the file names and the directory structure on the user local Internet server. Therefore the library data supplier may define hyperlinks between these **http_file**s.

This rule does not apply to **http_file**s representing property_value_external_item the storage of which shall be implementation dependant.

13.2.5 Hyper-text link usage

The following requirements shall be fulfilled by any library data supplier who wants to define hypertext links between several **http_file**s provided as library external files in a library exchange context.

All the hypertext links between **http_file**s provided as library external files in a library exchange context shall be identified by relative URL. These relative URLs shall be based on the assumption that the **http_file**s are stored on the user local Internet server according to the http-file storage rule defined in the previous subclause.

NOTE The concept of user local Internet server is discussed in 5.6.2.

13.2.6 Escape mechanism from document navigation to data retrieval and selection

A Library Management System that supports a library integrated information model or a view exchange protocol that enables the use of the **http_protocol** for library external files may provide two different ways for consulting a library content, for choosing a part family and for selecting a part within its family:

— a document oriented navigation, where the user follows the hypertext links to navigate from class-associated documents to class-associated documents, and selects the target classes or selects one part that is one instance of some class, and

— a database oriented navigation, where the user queries the data repository to retrieve a class and/or to select a part that fulfils some particular requirements.

An escape mechanism proves a useful means for a library end-user to express that he/she wants to switch from a document-oriented navigation to a database oriented navigation.

The Common Gateway Interface access rules specified in the next subclause enables the library data supplier to provide a capability for a library end-user to specify that he/she wants to switch from a document-oriented navigation to a database oriented navigation.

The Common Gateway Interface implementation rule (see clause 13.2.8) specifies how an ISO 13584-conformant implementation shall support these capabilities.

13.2.7 Common Gateway Interface access

When the library data supplier wants to enable the library end-user to switch from a document accessed through document navigation to data retrieval and selection by selecting an hypertext link contained in the document, the rules in the following subclauses shall be satisfied.

13.2.7.1 Common Gateway Interface access rule 1

To indicate that the library end-user should be set in database oriented navigation, and that the context of this database oriented navigation should be the current context of the database oriented navigation before the user starts document oriented navigation, the library data supplier shall define the **http_file** in such a way that the two strings: *HTTP-Version* and *lms-URI* are issued by the local Internet client of the library end-user. These two strings are expressed formally, in EBNF, as:

```
HTTP-Version = "HTTP/1.1"

lms-URI = "/cgi-bin/lms"
```

These two strings shall be separated by a linear white space lws:

```
lws = [<US-ASCII CR, carriage return (octet13)>
          <US-ASCII LF, linefeed (octet10)>]
          1*(<US-ASCII SP, space (octet32)>
          | <US-ASCII HT, horizontal-tab (octet9)>)
```

NOTE 1 The notational conventions used in ABNF are defined in clause 2 of IAB RFC 1808. They are summarised for convenience in the informative annex T.

NOTE 2 When this hyperlink is selected and the user did not initiate any database oriented navigation, the context of the database oriented navigation is the initial context of such a navigation.

13.2.7.2 Common Gateway Interface access rule 2

To indicate that the library end-user should be set in a database oriented navigation, and that the context of this database oriented navigation should be the context of selecting an instance in one particular class of which the selection results from the previous document-oriented navigation, the library data supplier shall define the **http_file** in such a way that the relative URL *class-URI* is issued by the local Internet client of the library end-user. This string is expressed formally, in EBNF, as:

```
class-URI = HTTP-Version lws "GET" lws "/cgi-bin/cl-sel?" class_id
```

NOTE The syntax of relative URL is defined in clause 2 of of IAB RFC 1808. The notational conventions used in ABNF are defined in clause 2 of IAB RFC 1808. Both are summarised for convenience in the informative annex T.

class_id shall be an URL-encoded string expressed formally, in EBNF, as:

```
class_id = "SUPPLIER" "=" supplier_code "&" "CLASS" "="
          class_dic_identifier
```

The following rules and restrictions shall apply:

— The *supplier_code* shall be the code of the library data supplier referenced by the **defined_by** attribute of the **class_BSU** that corresponds to the *class_dic_identifier* class.

— The *class_dic_identifier* shall be the value of the **dic_identifier** attribute of the class in which context the user shall be set, and this class shall be defined as an **item_class dictionary_definition**.

13.2.7.3 Common Gateway Interface access rule 3

To indicate that the library end-user should be set in a database-oriented navigation, and that some particular instance of some particular class shall be created by the LMS, the library data supplier shall define the **http-file** in such a way that the relative URL *instance-URI* is issued by the local Internet client of the library end-user. This string is expressed formally, in EBNF, as:

```
instances-URI = HTTP-Version lws "GET" lws
     "/cgi-bin/ins-sel?" instance_id
```

NOTE 1 The syntax of relative URL is defined in clause 2 of IAB RFC 1808. The notational conventions used in ABNF are defined in clause 2 of IAB RFC 1808. Both are summarised for convenience in the informative annex T.

instance_id shall be a URL-encoded string expressed formally, in EBNF, as:

```
instance_id = "SUPPLIER" "=" supplier_code "&" "CLASS" "="
     class_dic_identifier *( "&" property_value)

property_value = property_dic_identifier "=" value

value = simple_value | class_instance_value | "$"

class_instance_value = "(" instance_id ")"

simple_value = string_value |translated_string_value | integer_value |
real_value | Boolean_value
```

The following rules and restrictions shall apply:

— The *supplier_code* shall be the code of the library data supplier referenced by the **defined_by** attribute of the **class_BSU** that corresponds to the *class_dic_identifier* class.

— Each *class_dic_identifier* shall be the value of the **dic_identifier** attribute of a class defined by the preceding supplier.

— Each *class_dic_identifier* class shall be associated with an **item_class_extension** or an **explicit_item_class_extension** in the integrated user library.

— The **property_dic_identifier** shall be the value of the **dic_identifier** attribute of a property that corresponds to one of the **property_BSU**s returned by the **gm_identification_characteristics_list** function applied to the **class_BSU** that corresponds to the preceding *class_dic_identifier* class.

— Every **property_BSU** returned by the **gm_identification_characteristics_list** function applied to the **class_BSU** that corresponds to the preceding *class_dic_identifier* class shall correspond to exactly one *property_dic_identifier* in the list of *property_value* that immediately follows *class_dic_identifier*.

— Only *property_dic_identifier*s that correspond to **property_BSU**s belonging to the LIST of **property_BSU**s returned by the **optional_properties_list** function applied to the **class_BSU** that corresponds to the preceding *class_dic_identifier* class may have a *value* that is the "$" character.

— Each *value* that is a *translated_string_value* shall be the URL-encoded string of the value representation defined in ISO 10303-21 for the string that is the first one of the **string_values** attribute of the corresponding **translated_string_value**.

— Each *value* that is another type of *simple_value* shall be the URL-encoded string of the value representation defined for string, integer, real and Boolean in ISO 10303-21.

— The data type of the *value* of a *property_value* that is a *simple_value* shall be compatible with the **data_type** of the **property_BSU** that corresponds to the *property_dic_identifier* of this *property_value.*

NOTE 2 The **data_type** of a **property_BSU** is defined by the **domain** attribute of the **property_DET** associated with this **property_BSU**. These resource constructs are defined in the **ISO13584_IEC61360_dictionary_schema** documented in IEC61360-2 and quoted in an informative annex of ISO 13584-42.

NOTE 3 Compatibility between *simple_value* and the **data_type** of a **property_BSU** is defined by the **compatible_type_and_value** function of the **ISO13584_instance_resource_schema**, documented in clause 6 of this part of ISO 13584;

— The "CLASS" class referenced in the *class_instance_value* of a *property_value* shall correspond to a **class_BSU** that is compatible with the **class_BSU** that defines the **data_type** of the **property_BSU** that corresponds to the *property_dic_identifier* of this *property_value*.

NOTE 4 Compatibility between two **class_BSU** is defined by the **compatible_class_and_class** function documented in clause 6 of this part of ISO 13584.

13.2.8 Common Gateway Interface implementation rule

An ISO 13584 implementation that claims conformance to some library integrated information model or view exchange protocol that supports the **http_protocol** as an **external_file_protocol** shall recognise the three following relative URLs on the user local Internet server:

— `/cgi-bin/lms`

— `/cgi-bin/cl-sel`

— `/cgi-bin/ins-sel`

The interpretation of these relative URLs by the local Internet server shall have the effect to set the library end-user in database oriented navigation. The interpretation of the possible string that might follow a question mark ("?") after the two last relative URLs specified above is not mandatory. If this string is interpreted, the effect shall be to set the library end-user in the selection context specified in the previous subclause.

13.3 ISO13584_external_file_schema constant definitions

13.3.1 Compiler_version_length

The **compiler_version_length** is the maximum length of a **compiler_version_type**.

EXPRESS specification:

```
*)
CONSTANT
     compiler_version_length: INTEGER := 9;
(*
```

13.3.2 External_file_address_length

The **external_file_address_length** is the maximum length of an **external_file_address**.

NOTE 1 This length conforms to ISO 9075. This restriction intends to facilitate development of ISO 13584 LMSs that use existing relational database technology.

NOTE 2 This part of ISO 13584 does not specify the technology to be used for developing ISO 13584-conforming implementations.

EXPRESS specification:

```
*)
      external_file_address_length: INTEGER := 18;
(*
```

13.3.3 External_item_code_length

The **external_item_code_length** is the maximum length of an **external_item_code_type**.

EXPRESS specification:

```
*)
      external_item_code_length: INTEGER := 128;
(*
```

13.3.4 Http_file_name_length

The **http_file_name_length** is the maximum length of an **http_file_name_type**.

NOTE Validity of supplier-defined multifiles hypertext links requires that supplier-defined file names and supplier-defined directory names keep their values on the user file management system.

EXPRESS specification:

```
*)
      http_file_name_length: INTEGER := 128;
(*
```

13.3.5 Http_directory_name_length

The **http_directory_name_length** is the maximum length of an **http_directory_name_type**.

NOTE Validity of supplier-defined multifiles hypertext links requires that supplier-defined file names and supplier-defined directory names keep their values on the user file management system.

EXPRESS specification:

```
*)
      http_directory_name_length: INTEGER := 128;
END_CONSTANT;
(*
```

13.4 ISO13584_external_file_schema type definitions

13.4.1 External_file_address

An **external_file_address** is the physical address of an external file.

EXPRESS specification:

```
*)
TYPE external_file_address = identifier;
WHERE
     WR1: LENGTH(SELF) <= external_file_address_length;
     WR2: NOT(SELF LIKE '* *');
END_TYPE; -- external_file_address
(*
```

Formal propositions:

WR1: the length of file name shall be less or equal to **external_file_address_length**.

WR2: the name shall not contain any space.

13.4.2 External_item_code_type

The **external_item_code_type** is the code that identifies a **class_extension_external_item**.

EXPRESS specification:

```
*)
TYPE external_item_code_type = identifier;
WHERE
     WR1: LENGTH(SELF) <= external_item_code_length;
     WR2: NOT(SELF LIKE '* *');
END_TYPE; -- external_item_code_type
(*
```

Formal propositions:

WR1: the length of file name shall be less or equal to **external_item_code_length**.

WR2: the name shall not contain any space.

13.4.3 Http_file_name_type

The **http_file_name_type** is the type of the name of an **http_file**. This name is intended to be preserved in the user integrated library.

EXPRESS specification:

```
*)
TYPE http_file_name_type = identifier;
WHERE
     WR1: LENGTH(SELF) <= http_file_name_length;
     WR2: NOT(SELF LIKE '* *');
     WR3: NOT(SELF LIKE '*.*.*');
     WR4: ((NOT(SELF LIKE '*.*'))
          AND (LENGTH(SELF) <= http_file_name_length - 4))
```



```
          OR ((SELF LIKE '*.?')
          AND (LENGTH(SELF) <= http_file_name_length - 2))
          OR ((SELF LIKE '*.??')
          AND (LENGTH(SELF) <= http_file_name_length - 1))
          OR (SELF LIKE '*.???');
END_TYPE; -- http_file_name_type
(*
```

Formal propositions:

WR1: the length of file name shall not be greater than **http_file_name_length**.

WR2: the name shall not contain any space.

WR3: the name shall not contain more than one dot (".").

WR4: the name may contain a file extension that consists of one, two or three characters and a name that contains no more than eight characters.

13.4.4 Http_directory_name_type

The **http_directory_name_type** is the type of the name of an **http_class_directory**. This name is intended to be preserved in the user integrated library.

EXPRESS specification:

```
*)
TYPE http_directory_name_type = identifier;
WHERE
     WR1: LENGTH(SELF) <= http_directory_name_length;
     WR2: NOT(SELF LIKE '* *');
END_TYPE; -- http_directory_name_type
(*
```

Formal propositions:

WR1: the length of the directory name shall not be greater than **http_directory_name_length**.

WR2: the name shall not contain any space.

13.4.5 MIME_type

The **MIME_type** is an identifier registered as MIME type by IANA.

EXPRESS specification:

```
*)
TYPE MIME_type = identifier;
WHERE
     WR1: NOT(SELF LIKE '* *');
END_TYPE; -- MIME_type
(*
```


Formal propositions:

WR1: the name shall not contain any space.

Informal propositions:

IP1: the **MIME_type** value shall be an identifier registered as MIME type by IANA and its content shall be identical to the string contained in ftp://ftp.isi.edu/in-notes/iana/assignments/, after removing the space characters if there are some.

13.4.6 MIME_subtype

The **MIME_subtype** is an identifier registered as MIME subtype by IANA.

EXPRESS specification:

```
*)
TYPE MIME_subtype = identifier;
WHERE
     WR1: NOT(SELF LIKE '* *');
END_TYPE; -- MIME_subtype
(*
```

Formal propositions:

WR1: the name shall not contain any space.

Informal propositions:

IP1: the **MIME_subtype** value shall be an identifier registered as MIME subtype by IANA and it content shall be identical to the string contained in ftp://ftp.isi.edu/in-notes/iana/assignments/, after removing the space characters if there are some.

13.4.7 IAB_RFC

An **IAB_RFC** is an integer that identifies an RFC from the IAB.

EXPRESS specification:

```
*)
TYPE IAB_RFC = INTEGER;
WHERE
     WR1: SELF > 0;
END_TYPE; -- IAB_RFC
(*
```

Formal propositions:

WR1: the **IAB_RFC** value shall be positive.

Informal propositions:

IP1: the **IAB_RFC** value shall be the identifier of a RFC belonging to the Standard Track of the IAB.

13.4.8 Character_set_type

The **character_set_type** is the type of the character set encoding of a library external file that contains characters.

EXPRESS specification:

```
*)
TYPE character_set_type = identifier;
END_TYPE; -- character_set_type
(*
```

Informal propositions:

IP1: the **character_set_type** shall be the identifier of a character-set encoding registered in IAB RFC 1700 or, possibly, in an IAB RFC that updates IAB RFC 1700.

13.4.9 Content_encoding_type

The **content_encoding_type** is the type that specifies what sort of encoding transformation the content of a library external file was subjected to and hence what decoding operation must be used to restore it to its original form, and it specifies what the domain of the result is. The allowed encodings are those defined by IAB RFC 2045. They are referenced by the same name as the ones defined in IAB RFC 2045.

NOTE File encoding might prove necessary during the exchange of a supplier library, whether this exchange is done by means of some network or not. These files might be decoded during the integration of a supplier library into the user library. Thus, their **content_encoding_type** value might be modified during this integration.

EXPRESS specification:

```
*)
TYPE content_encoding_type = identifier;
WHERE
     WR1: (SELF = '7bit') OR (SELF = '8bit') OR (SELF = 'binary')
          OR (SELF = 'quoted-printable') OR (SELF = 'base64');
END_TYPE; -- content_encoding_type
(*
```

Formal propositions:

WR1: the encoding shall be one of the encodings defined in IAB RFC 2045.

13.4.10 Program_status

A **program_status** is the type that defines the status of a program provided as a library external file. Programs provided through external files can be either compiled or represented in a source code language intended to be either interpreted or compiled.

EXAMPLE 1 FORTRAN programs delivered as object code are examples of compiled programs.

EXAMPLE 2 Parametric models delivered as data model entities are examples of source code programs.

EXPRESS specification:

```
*)
TYPE program_status = ENUMERATION OF(source, compiled);
END_TYPE; -- program_status
(*
```

13.4.11 Program_reference_name_type

The **program_reference_name_type** is the type of a syntactical name of a program.

NOTE View exchange protocols that enable the use of **program_reference**s precise the role associated with the identifiers that define the syntactical names of programs.

EXPRESS specification:

```
*)
TYPE program_reference_name_type = identifier;
END_TYPE; -- program_reference_name_type
(*
```

13.4.12 Compiler_version_type

The **compiler_version_type** is the version associated with a compiler.

NOTE Only restricted patterns of identifiers are allowed.

EXPRESS specification:

```
*)
TYPE compiler_version_type = identifier;
WHERE
     WR1: LENGTH(SELF) <= compiler_version_length;
     WR2: control_compiler_version_format(SELF);
END_TYPE; -- compiler_version_type
(*
```

Formal propositions:

WR1: the length of the identifier representing the version shall be less than or equal to **version_len** that specifies the length of a version in the **ISO13584_IEC61360_dictionary_schema**.

NOTE The **ISO13584_IEC61360_dictionary_schema** is defined in IEC 61360-2. It is duplicated for convenience in informative annex D of Part ISO 13584-42.

WR2: the compiler version shall contain only digits, dots, or underscores.

13.4.13 Illustration_type

An **illustration_type** is the type that categorises the content of an **illustration**. It may be a schematic drawing, a realistic picture, or any other informative element that is not a static picture and whose format may be defined by a **data_protocol**.

EXAMPLE 1 A movie is an **illustration** content that is not a static picture.

EXAMPLE 2 A sound is an **illustration** content that is not a static picture.

NOTE The categorisation of the contents of the **illustration**s provided in a supplier library may be used, for instance, when creating "on the fly" an HTML view of the supplier library.

EXPRESS specification:

```
*)
TYPE illustration_type = ENUMERATION OF(
     schematic_drawing, realistic_picture, not_static_picture);
END_TYPE; -- illustration_type
(*
```

13.5 ISO13584_external_file_schema entity definitions: external_file_protocols

This clause defines the requirements for the **external_file_protocol**s. An **external_file_protocol** entity allows to specify how information provided as external files shall be processed. An **external_file_protocol** may be either a **program_protocol** or a **data_protocol**, and it may be either a **standard_protocol** or a **non_standard_protocol**.

NOTE 1 Several different **external_file_protocol**s can be used for the different types of files. Libraries modeled as **library_in_standard_format**s only use predefined **external_file_protocol**s specified by the library integrated information model and the set of standard view exchange protocols they reference. For general libraries, modeled as **library**'s, it is up to the library data supplier to decide to select any **external_file_protocol**s from international or national Standards (for instance STEP, SGML, SET, IGES or VDA-FS), from proprietary formats (for instance Postscript or EDIF) or from system-specific exchange formats (for instance DXF or CATIA® parametric programs).

NOTE 2 **library_in_standard_format** and **library** are defined in the **ISO13584_extended_dictionary_schema** documented in clause 10.

13.5.1 External_file_protocol

An **external_file_protocol** is a protocol that shall be used to process a library external file to process information provided as external files.

EXPRESS specification:

```
*)
ENTITY external_file_protocol
ABSTRACT SUPERTYPE OF(
     (ONEOF(standard_protocol, non_standard_protocol))
     ANDOR (ONEOF(program_protocol, data_protocol)));
     organisation: organization;
     country: OPTIONAL identifier;
     protocol_name: identifier;
     protocol_version: identifier;
     level: OPTIONAL identifier;
     designation: item_names;
```



```
      base_protocol: OPTIONAL program_protocol;
WHERE
      WR1: (NOT(SELF.protocol_name LIKE '* *'))
           AND (NOT(SELF.protocol_name LIKE '*.*'))
           AND (NOT(SELF.protocol_name LIKE '*-*'));
      WR2: NOT(SELF.protocol_version LIKE '* *');
END_ENTITY; -- external_file_protocol
(*
```

Attribute definitions:

organisation: the organisation that specified the protocol.

country: the country where the organisation that specifies the **external_file_protocol** has its headquarters.

protocol_name: the identifier specified for the **external_file_protocol** by the organisation that specified it, without any space, dot or hyphen character. Underscores are to be used as separators.

protocol_version: an identification of the version of the specified **external_file_protocol**.

level: the possible level associated with the specified **external_file_protocol** by the organisation that specified it.

designation: the **item_names** of the **external_file_protocol**.

NOTE 1 Each view exchange protocol shall specify the **external_file_protocol**s that are recognised by an implementation that claims conformance to this view exchange protocol.

NOTE 2 If more than one **level** is defined and used in a library, each level is specified by a different **external_file_protocol** entity.

base_protocol: the **program_protocol** to which the program libraries referred to are to be linked.

Formal propositions:

WR1: the **protocol_name** shall not contain neither space, nor dot nor hyphen.

WR2: the **protocol_version** shall not contain a space.

Informal propositions:

IP1: the **label** describing a country shall conform to the codes for representation of names of countries defined in ISO 3166.

IP2: the **country** attribute value shall not exist for the **external_file_protocol**s specified by International Standards.

IP3: the **country** attribute value shall exist for the **external_file_protocol**s that are not specified by International Standards. It shall conform to ISO 3166.

13.5.2 Standard_protocol

A **standard_protocol** is an **external_file_protocol** specified in a National, or International Standard.

EXPRESS specification:

```
*)
ENTITY standard_protocol
ABSTRACT SUPERTYPE OF(ONEOF(standard_simple_program_protocol,
     standard_data_protocol))
SUBTYPE OF(external_file_protocol);
END_ENTITY; -- standard_protocol
(*
```

Informal propositions:

IP1: the **protocol_name** identifier shall consist of the acronym of the standardisation organisation followed by the underscore ('_') character, followed by the number of the Standard that specified the **external_file_protocol** without any space, dot or hyphen character. Underscores shall be used as separators.

IP2: when the standard document that specifies the standard protocol does not explicitly specify a version number for this protocol, the **protocol_version** shall consist of the version number of the standard document without any leading space or leading zero; the version number of the first release of a standard being '1'.

NOTE The content of the strings defining the standard **external_file_protocol**s that are allowed for use by each library integrated information model or view exchange protocol are defined within the part of ISO 13584 that specifies the corresponding library integrated information model or view exchange protocol.

13.5.3 Non_standard_protocol

A **non_standard_protocol** is an **external_file_protocol** specified by an organisation that is neither a national nor an international standardisation organisation.

EXPRESS specification:

```
*)
ENTITY non_standard_protocol
ABSTRACT SUPERTYPE OF(ONEOF(non_standard_simple_program_protocol,
     non_standard_data_protocol))
SUBTYPE OF(external_file_protocol);
END_ENTITY; -- non_standard_protocol
(*
```

NOTE The content of the string defining a **non_standard_protocol** shall be defined by the company that specified this **external_file_protocol**.

13.5.4 Data_protocol

A **data_protocol** is an **external_file_protocol** that processes data. It can be either a **standard_data_protocol** or a **non_standard_data_protocol**.

EXPRESS specification:

```
*)
ENTITY data_protocol
ABSTRACT SUPERTYPE OF(ONEOF(standard_data_protocol,
     non_standard_data_protocol))
```



```
SUBTYPE OF(external_file_protocol);
END_ENTITY; -- data_protocol
(*
```

13.5.5 Program_protocol

A **program_protocol** is an **external_file_protocol** that processes programs, i.e., algorithms that shall be triggered to generate data. It can be either a **linked_interface_program_protocol** or a **simple_program_protocol**.

EXPRESS specification:

```
*)
ENTITY program_protocol
ABSTRACT SUPERTYPE OF(ONEOF(
      linked_interface_program_protocol,
      simple_program_protocol))
SUBTYPE OF(external_file_protocol);
END_ENTITY; -- program_protocol
(*
```

13.5.6 Simple_program_protocol

A **simple_program_protocol** is an **external_file_protocol** that processes programs.

NOTE 1 If it is required to deliver compiled programs, it is up to the library data supplier to define the **compiler_name** identifier and the value of **compiler_version** that enables portability of the object code.

NOTE 2 Exchange of compiled programs are subject to prior agreement between the library data supplier and the library user.

EXPRESS specification:

```
*)
ENTITY simple_program_protocol
ABSTRACT SUPERTYPE OF(ONEOF(standard_simple_program_protocol,
      non_standard_simple_program_protocol))
SUBTYPE OF(program_protocol);
      language: identifier;
      status: program_status;
      compiler_supplier: OPTIONAL organization;
      compiler_name: OPTIONAL identifier;
      compiler_version: OPTIONAL compiler_version_type;
WHERE
      WR1: ((SELF.status = source)
           AND (NOT(EXISTS(SELF.compiler_supplier)))
           AND (NOT(EXISTS(SELF.compiler_name)))
           AND (NOT(EXISTS(SELF.compiler_version))))
           OR ((SELF.status = compiled)
           AND (EXISTS(SELF.compiler_supplier))
           AND (EXISTS(SELF.compiler_name))
           AND (EXISTS(SELF.compiler_version)));
```



```
END_ENTITY; -- simple_program_protocol
(*
```

Attribute definitions:

language: an **identifier** that specifies the associated programming language.

status: the **status** of the program, if it is compiled or in source language.

compiler_supplier: the organisation who provides the compiler used.

compiler_name: the identifier of the compiler, as specified by the library data supplier.

compiler_version: the version of the compiler used, as specified by the library data supplier.

Formal propositions:

WR1: if the status of the program is **source**, there shall not exist a **compiler_name**, nor a **compiler_version** nor a **compiler_supplier**. If the **status** of the program is **compiled**, there shall exist a **compiler_name**, a **compiler_supplier** and a **compiler_version**.

13.5.7 Standard_simple_program_protocol

A **standard_simple_program_protocol** is a program protocol specified by a standardisation organisation.

EXPRESS specification:

```
*)
ENTITY standard_simple_program_protocol
SUBTYPE OF(standard_protocol, simple_program_protocol);
WHERE
     WR1: NOT EXISTS(SELF\external_file_protocol.base_protocol);
END_ENTITY; -- standard_simple_program_protocol
(*
```

Formal propositions:

WR1: the **base_protocol** shall not be defined.

13.5.8 Non_standard_simple_program_protocol

A **non_standard_simple_program_protocol** is a program protocol that is not specified by a standardisation organisation.

EXPRESS specification:

```
*)
ENTITY non_standard_simple_program_protocol
SUBTYPE OF(non_standard_protocol, simple_program_protocol);
WHERE
     WR1: NOT EXISTS(SELF\external_file_protocol.base_protocol);
```

```
END_ENTITY; -- non_standard_simple_program_protocol
(*
```

Formal propositions:

WR1: the **base_protocol** shall not be defined.

13.5.9 Linked_interface_program_protocol

A **linked_interface_program_protocol** entity is a **program_protocol** that consists of a base **program_protocol** that shall be linked with libraries of programs. This entity provides for defining a new **program_protocol** where the supplier programs may reference not only the functions of the base **program_protocol** but also the functions defined in these libraries of programs. These libraries are either delivered in the library exchange context, or referenced.

NOTE The status of the programs associated with a **linked_interface_program_protocol** is defined by the **simple_program_protocol** on which it is based.

EXPRESS specification:

```
*)
ENTITY linked_interface_program_protocol
SUBTYPE OF(program_protocol);
      link_libraries: SET [1:?] OF program_library_BSU;
WHERE
      WR1: QUERY(pl <* SELF.link_libraries
           | (SIZEOF(pl\basic_semantic_unit.referenced_by) > 0)
           AND (pl\basic_semantic_unit.referenced_by[1]
           \external_item.used_protocol <> SELF.base_protocol)) = [];
      WR2: NOT('ISO13584_EXTERNAL_FILE_SCHEMA.STANDARD_PROTOCOL'
           IN TYPEOF(SELF)) AND
           NOT('ISO13584_EXTERNAL_FILE_SCHEMA.NON_STANDARD_PROTOCOL'
           IN TYPEOF(SELF));
      WR3: EXISTS(base_protocol)
           AND ('ISO13584_EXTERNAL_FILE_SCHEMA.PROGRAM_PROTOCOL'
           IN TYPEOF(base_protocol));
END_ENTITY; -- linked_interface_program_protocol
(*
```

Attribute definitions:

link_libraries: the set of program libraries that need to be linked with the **base_protocol** to build up the referred **linked_interface_program_protocol**.

Formal propositions:

WR1: the **base_protocol** shall be the **program_protocol** to which each program library is associated in its **program_library_content** definition.

WR2: the **linked_interface_program_protocol** shall not be a **standard_protocol** nor a **non_standard_protocol**.

WR3: the **base_protocol** attribute shall be defined, and it shall be a **program_protocol**.

13.5.10 Standard_data_protocol

A **standard_data_protocol** entity is a data protocol specified by a standardisation organisation.

EXPRESS specification:

```
*)
ENTITY standard_data_protocol
SUBTYPE OF(data_protocol, standard_protocol);
WHERE
     WR1: NOT EXISTS(SELF\external_file_protocol.base_protocol);
END_ENTITY; -- standard_data_protocol
(*
```

Formal propositions:

WR1: the **base_protocol** shall not be defined.

13.5.11 Non_standard_data_protocol

A **non_standard_data_protocol** is a **data_protocol** specified by a private company.

EXPRESS specification:

```
*)
ENTITY non_standard_data_protocol
SUBTYPE OF(data_protocol, non_standard_protocol);
WHERE
     WR1: NOT EXISTS(SELF\external_file_protocol.base_protocol);
END_ENTITY; -- non_standard_data_protocol
(*
```

Formal propositions:

WR1: the **base_protocol** shall not be defined.

13.5.12 Http_protocol

An **http_protocol** is a **data_protocol** specified by a RFC registered as standard by the IAB.

EXPRESS specification:

```
*)
ENTITY http_protocol
SUBTYPE OF(standard_data_protocol);
     http_RFC: IAB_RFC;
END_ENTITY; -- http_protocol
(*
```

Attribute definitions:

http_RFC: the number of the RFC registered as standard by the IAB that specifies the used version of the http protocol.

Informal propositions:

IP1: the name of the protocol shall be the http protocol or one of its specialisation or updating as defined by the authorisation in charge the Internet protocols.

EXAMPLE The https protocol is a specialization of the http protocol.

NOTE The current authority in charge of the http protocol is the Internet Engineering Task Force.

13.6 ISO13584_external_file_schema entity definitions: dictionary external items

This subclause specifies the different items supplied by means of library external files and identified by a **basic_semantic_unit**. They can be either program libraries, or documents.

NOTE **Basic_semantic_unit** is defined in the common ISO/IEC dictionary schema documented in IEC 61360-2 and quoted in ISO 13584-42.

These **external_item**s are **dictionary_external_item**s: they are associated with a **dictionary_element** and are referred to by the BSU mechanism defined in the ISO/IEC common dictionary schema.

13.6.1 External_item

An **external_item** entity is any 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. This abstract resource shall be subtyped when an **external_item** is used in order to specify the meaning of this **external_item**.

EXPRESS specification:

```
*)
ENTITY external_item
ABSTRACT SUPERTYPE OF(ONEOF(dictionary_external_item,
     class_extension_external_item,
     property_value_external_item));
     used_protocol: external_file_protocol;
     content: external_content;
END_ENTITY; -- external_item
(*
```

Attribute definitions:

used_protocol: the **external_file_protocol** that specifies how the library external file(s) shall be processed.

content: the library external file(s) that represent the **external_item** content.

13.6.2 Dictionary_external_item

A **dictionary_external_item** entity is any **external_item** that is identified by a **basic_semantic_unit** and that may be referred to by the BSU mechanism defined in the ISO/IEC common dictionary schema. A **dictionary_external_item** may be updated alone. It has a **revision** that characterises the updating.

NOTE 1 A **basic_semantic_unit** also has a version number that is part of its identification.

EXPRESS specification:

```
*)
ENTITY dictionary_external_item
ABSTRACT SUPERTYPE OF(ONEOF(supplier_BSU_related_content,
     class_BSU_related_content))
SUBTYPE OF(content_item, external_item);
     revision: revision_type;
END_ENTITY; -- dictionary_external_item
(*
```

Attribute definitions:

revision: the **revision_type** that characterises the updating of the **dictionary_external_item**.

Informal propositions:

IP1: **revision** numbers shall be issued in ascending order for each value of the version of a **basic_semantic_unit**.

IP2: a new revision number of a **dictionary_external_item** shall be defined when a change in the **external_content** that describes this **dictionary_external_item** influences neither its meaning nor its use.

NOTE 2 This informal proposition states that when updating a library an old revision may be replaced by a new one without any consequence on the functional behaviour of the integrated library.

13.6.3 Supplier_BSU_related_content

A **supplier_BSU_related_content** entity is a **dictionary_external_item** whose **basic_semantic_unit** is a **supplier_related_BSU**.

EXPRESS specification:

```
*)
ENTITY supplier_BSU_related_content
ABSTRACT SUPERTYPE OF(program_library_content)
SUBTYPE OF(dictionary_external_item);
     SELF\content_item.dictionary_definition: supplier_related_BSU;
END_ENTITY; -- supplier_BSU_related_content
(*
```


Attribute definitions:

SELF\content_item.dictionary_definition: the **supplier_related_BSU** that identifies the **dictionary_external_content**.

13.6.4 Program_library_content

A **program_library_content** is the content of a program library. It refers to a **program_protocol** that specifies how it shall be processed. It is identified by a **program_library_BSU**.

EXPRESS specification:

```
*)
ENTITY program_library_content
SUBTYPE OF(supplier_BSU_related_content);
     SELF\content_item.dictionary_definition: program_library_BSU;
     SELF\external_item.used_protocol: program_protocol;
END_ENTITY; -- program_library_content
(*
```

Attribute definitions:

SELF\content_item.dictionary_definition: the **program_library_BSU** that identifies the **program_library_content**.

SELF\external_item.used_protocol: the **program_protocol** that specifies how the **program_library_content** shall be processed.

Informal propositions:

IP1: all the revisions that correspond to the same **program_library_BSU** version shall use the same **program_protocol** and shall contain a set of programs that provide the same referencable specifications. The only permitted changes are those that affect the bodies of these programs, or the programs that are used internally in the bodies of the referencable programs.

NOTE This informal proposition states that when updating a library, an old revision may be replaced by a new one without any consequence on the functional behaviour of the integrated library.

13.6.5 Class_BSU_related_content

A **class_BSU_related_content** entity is a **dictionary_external_item** whose **basic_semantic_unit** is a **class_related_BSU**.

EXPRESS specification:

```
*)
ENTITY class_BSU_related_content
ABSTRACT SUPERTYPE OF(document_content)
SUBTYPE OF(dictionary_external_item);
     SELF\content_item.dictionary_definition: class_related_BSU;
END_ENTITY; -- class_BSU_related_content
(*
```


Attribute definitions:

SELF\content_item.dictionary_definition: the **class_related_BSU** that identifies the **dictionary_external_content**.

13.6.6 Document_content

A **document_content** is the content of a document. It refers to a **data_protocol** that specifies how it shall be processed. It is identified by a **document_BSU**.

EXPRESS specification:

```
*)
ENTITY document_content
SUBTYPE OF(class_BSU_related_content);
     SELF\content_item.dictionary_definition: document_BSU;
     SELF\external_item.used_protocol: data_protocol;
END_ENTITY; -- document_content
(*
```

Attribute definitions:

SELF\content_item.dictionary_definition: the **document_BSU** that identifies the **document_content**.

SELF\external_item.used_protocol: the **data_protocol** that specifies how the **document_content** shall be processed.

Informal propositions:

IP1: all the revisions that correspond to the same **document_BSU** version shall have a content that corresponds to the same **document_element dictionary_definition**. The changes identified by different revisions can be typo corrections, addition of new translations or change of the protocol to be used to process the **document_content**.

13.7 ISO13584_external_file_schema entity definition: class extension external items

This subclause specifies the different items that are supplied by means of library external files and that are part of the extension of a class. They can be representations, programs or dialogue resources.

Programs, representations and dialogue resources are **class_extension_external_item**s. They are referred to directly by the **model_class_extension**s they belong to through their entity name. They shall be updated as a whole when a new release of the **model_class_extension** is delivered.

The following planning model outlines the structure of the different categories of **class_extension_external_item**s.

NOTE This planning model uses the EXPRESS-G graphical notation for the EXPRESS language, but, for clarification of the diagram, some of the relationships that are defined in the EXPRESS model are omitted, and some inter-schema references are not represented.

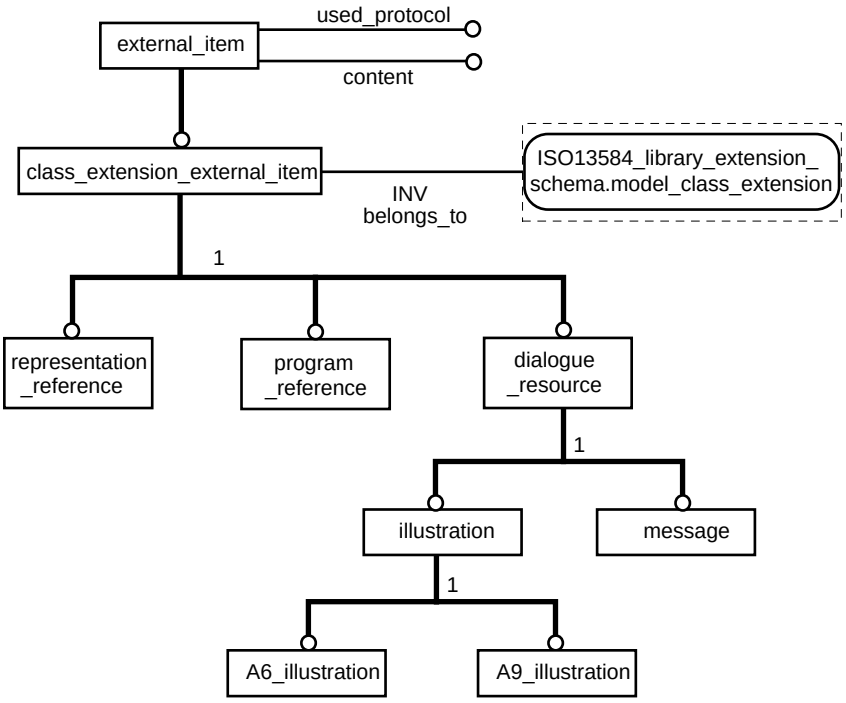


Figure 9 — Class_extension_external_items planning model

13.7.1 Class_extension_external_item

A **class_extension_external_item** entity is any **external_item** that is part of the content of a **model_class_extension**. A **class_extension_external_item** is not associated with a BSU. It has neither version nor revision. It belongs to only one **model_class_extension** and it is intended to be updated as a whole when a new release of the **model_class_extension** is delivered. It is associated with an **external_item_code_type** that shall be unique for the **model_class_extension** to which it belongs.

EXPRESS specification:

```
*)
ENTITY class_extension_external_item
ABSTRACT SUPERTYPE OF(ONEOF(dialogue_resource,
     representation_reference, program_reference))
SUBTYPE OF(external_item);
     code: external_item_code_type;
INVERSE
     belongs_to: model_class_extension FOR referenced_external_items;
UNIQUE
     UR1: code, belongs_to;
END_ENTITY; -- class_extension_external_item
(*
```

Attribute definitions:

code: the **external_item_code_type** that identifies the **class_extension_external_item** in its class.

belongs_to: the **model_class_extension** to which the **class_extension_external_item** belongs.

Formal propositions:

UR1: the **code** shall be unique for the **model_class_extension** to which the **class_extension_external_item** belongs.

13.7.2 Representation_reference

A **representation_reference** is a reference to a representation of a library item whose content is provided as part of library external file(s). In the library delivery file, the **representation_reference** enables to refer to this representation.

The **representation_reference** contains an optional attribute, called **representation_id**, that enables to specify one particular element of the **external_content** associated with **representation_reference**. The nature of the external description and the role of the **representation_id** shall be specified by the **data_protocol** referenced by the **representation_reference**.

EXAMPLE The target of a **representation_reference** may be a product **representation** defined in a file conformant with some ISO 10303 Application Protocol. In this case, the **representation_id** might be defined by the **data_protocol** that enables to reference such a file, as the **label** assigned to an EXPRESS entity instance of this file by an **external_referent_assignment** entity.

NOTE The information model of a **representation** is defined in ISO 10303-43. The information model of an **external_referent_assignment** and **label** are defined in ISO 10303-41.

EXPRESS specification:

```
*)
ENTITY representation_reference
SUBTYPE OF(class_extension_external_item);
     SELF\external_item.used_protocol: data_protocol;
     representation_id: OPTIONAL label;
END_ENTITY; -- representation_reference
(*
```

Attribute definitions:

SELF\external_item.used_protocol: the **data_protocol** that needs to be used to process the external file(s) that contains the referenced **representation**.

representation_id: the possible **label** that corresponds to the referenced representation in the **external_content** that corresponds to the **representation_reference program_reference**.

13.7.3 Program_reference

A **program_reference** entity is a reference to an algorithm whose contents is provided as library external file(s). This algorithm shall be triggered to generate one **representation** of an item of an **item_class**. A **program_reference** provides the information about the program name and parameters.

NOTE The allowed types for parameters and the allowed syntactical names for programs shall be specified in the view exchange protocols that use this resource construct.

EXPRESS specification:

```
*)
ENTITY program_reference
SUBTYPE OF(class_extension_external_item);
     SELF\external_item.used_protocol: program_protocol;
     syntactical_name: program_reference_name_type;
     in_parameters: LIST [0:?] OF property_or_data_type_BSU;
     out_parameters: LIST [0:?] OF property_or_data_type_BSU;
     inout_parameters: LIST [0:?] OF property_or_data_type_BSU;
END_ENTITY; -- program_reference
(*
```

Attribute definitions:

SELF\class_extension_external_item.used_protocol: the **program_protocol** that needs to be used to process the external file(s).

syntactical_name: the name by which the program shall be triggered.

in_parameters: the list of **property_or_data_type_BSU**s that specifies the types of the input parameters of the referenced program.

out_parameters: the list of **property_or_data_type_BSU**s that specifies the types of the output parameters of the referenced program.

inout_parameters: the list of **property_or_data_type_BSU**s that specifies the types of the inout parameters of the referenced program.

13.7.4 Dialogue_resource

A **dialogue_resource** is an informative item intended to be automatically displayed by the LMS in some given work contexts.

EXPRESS specification:

```
*)
ENTITY dialogue_resource
ABSTRACT SUPERTYPE OF(ONEOF(message, illustration))
SUBTYPE OF(class_extension_external_item);
     SELF\external_item.used_protocol: data_protocol;
END_ENTITY; -- dialogue_resource
(*
```

Attribute definitions:

SELF\class_extension_external_item.used_protocol: the **data_protocol** that needs to be used to process the external file.

13.7.5 Message

A **message** is a text that shall be short, typically up to 256 characters, and that shall be automatically displayed on the screen by the user library management system within a clearly defined work context.

A **message** is not associated with any particular window dimension. It may be provided in different languages and shall be processed through a certain **data_protocol**.

EXPRESS specification:

```
*)
ENTITY message
SUBTYPE OF(dialogue_resource);
END_ENTITY; -- message
(*
```

13.7.6 Illustration

An **illustration** is an informative item that is not a simple textual message. It may be a schematic drawing, a realistic picture, or any other informative element that is not a static picture and whose format may be defined by a **data_protocol**. It may be associated with an idea of window dimension that is intended to be filled when the **illustration** is displayed in a certain context. Two subtypes of **illustration**, the dimension of which are specified, exist.

NOTE 1 The role of an **illustration** in a supplier library is defined by the entity that references it. For example, it may be automatically displayed on the screen by the LMS within a clearly defined work context.

NOTE 2 An **illustration** is a specialisation of a **graphics** entity defined in ISO 13584-42. Another specialisation of a **graphics** entity is a **referenced_graphics** defined in the **ISO13584_extended_dictionary_schema**. The **illustration_is_not_a_referenced_graphics_rule** ensures that both subtypes are incompatible.

EXPRESS specification:

```
*)
ENTITY illustration
SUPERTYPE OF(ONEOF(A6_illustration, A9_illustration))
SUBTYPE OF(dialogue_resource, graphics);
     kind_of_content: illustration_type;
     width: OPTIONAL length_measure_with_unit;
     height: OPTIONAL length_measure_with_unit;
WHERE
     WR1: (NOT EXISTS(SELF.width) AND NOT EXISTS(SELF.height))
          OR (EXISTS(SELF.width) AND EXISTS(SELF.height));
END_ENTITY; -- illustration
(*
```

Attribute definitions:

kind_of_content: categorisation of the **illustration** content according to the taxonomy defined by the **illustration_type** data type.

width: the width of the window recommended by the library data supplier for viewing the **illustration**.

height: the height of the window recommended by the library data supplier for viewing the **illustration.**

Formal propositions:

WR1: either both width and height shall exist, or none of these attributes shall exist.

13.7.7 A6_illustration

An **A6_illustration** is an **illustration** of which the library data supplier ensures that it may be displayed in a window of size A6.

NOTE 1 This part of ISO 13584 requests some illustrations provided by a library data supplier to be **A6_illustration**

NOTE 2 The intended roles of illustrations of A6 size include the following:

— to display the meaning of properties of a general model class,

— to display the meaning of free model properties of a functional model class,

— to display the views defined by a functional models class.

NOTE 3 The width and height recommended by the library data supplier, as defined by the width and height attributes, for viewing the **illustration** may be different from size A6. The optimal size for viewing the illustration is defined by the width and height attributes. A window of size A6 is also acceptable for displaying it.

EXPRESS specification:

```
*)
ENTITY A6_illustration
SUBTYPE OF(illustration);
END_ENTITY; -- A6_illustration
(*
```

13.7.8 A9_illustration

An **A9_illustration** is an **illustration** of which the library data supplier ensures that it may be displayed in a window of size A9.

NOTE 1 This part of ISO 13584 requests some **illustration**s provided by a library data supplier to be **A9_illustration**

NOTE 2 One intended role of an illustration of A9 size is to represent the class in a menu driven hierarchical access.

NOTE 3 The width and height recommended by the library data supplier, as defined by the width and height attributes, for viewing the **illustration** may be different from size A9. The optimal size for viewing the **illustration** is defined by the width and height attributes. A window of size A9 is also acceptable for displaying it.

EXPRESS specification:

```
*)
ENTITY A9_illustration
SUBTYPE OF(illustration);
END_ENTITY; -- A9_illustration
(*
```

13.8 ISO13584_external_file_schema entity definition: property_value_external_item

This subclause specifies how to define a property value by means of library external files.

EXAMPLE the content of such external files might be icons that show the value of a property named "*simplified picture of a jack*" defined in some *jack* class

A **property_value_external_item** is an **external_item** that may constitute the value of any property the data type of which is defined as an **entity_instance_type**, the **type_name** attribute of which contains 'ISO13584_EXTERNAL_FILE_SCHEMA.PROPERTY_VALUE_EXTERNAL_ITEM'.

NOTE 1 **property_value_external_item** is defined as a **PLIB_entity_instance_value** and thus as a **primitive_value**in the **ISO13584_instance_resource_schema** documented in clause 6.

NOTE 2 Type checking for property values is ensured by **WR1** of the **property_value** entity documented in clause 6..

NOTE 3 **entity_instance_type** is defined in the **ISO13584_IEC61360_dictionary_schema** documented in ISO 13584-42:1998.

EXPRESS specification:

```
*)
ENTITY property_value_external_item
SUBTYPE OF(external_item);
END_ENTITY; -- property_value_external_item
(*
```

13.9 ISO13584_external_file_schema rule definition

13.9.1 Unique_http_file_name_per_supplier_element_rule rule

The **unique_http_file_name_per_supplier_element_rule** rule checks that all the **http_file**s associated with the different **supplier_related_BSU**s that correspond to the same **supplier_BSU** have different file names. If the **supplier_element** is not available, the rule is not violated.

NOTE All the **http_file**s associated with the different **supplier_related_BSU**s that correspond to the same **supplier_BSU** are intended to be stored in the same directory on the user local Internet server.

EXPRESS specification:

```
*)
RULE unique_http_file_name_per_supplier_element_rule FOR(
     supplier_BSU);
WHERE
     WR1: QUERY(sup_BSU <* supplier_BSU | (SIZEOF
          (sup_BSU.definition) = 1) AND
          (QUERY(fil_1 <* supplier_associated_http_files(sup_BSU)
          | QUERY(fil_2 <* supplier_associated_http_files(sup_BSU)
          | fil_1.http_file_name = fil_2.http_file_name)
          <> [fil_1]) <> []))
          = [];
END_RULE; -- unique_http_file_name_per_supplier_element_rule
(*
```

Formal propositions:

WR1: all the **http_file**s associated with the different **supplier_related_BSU**s that correspond to the same **supplier_BSU** shall have different file names.

13.9.2 Unique_http_directory_name_per_supplier_rule rule

The **unique_http_directory_name_per_supplier_rule** rule checks that all the **http_class_directory**'s associated with the same **supplier_BSU** have different names.

NOTE All the **http_class_directory**ies associated with the same **supplier_BSU** are intended to be subdirectories of the same directory on the user local Internet server.

EXPRESS specification:

```
*)
RULE unique_http_directory_name_per_supplier_rule FOR(
     http_class_directory);
WHERE
     WR1: QUERY(dir_1 <* http_class_directory
          | QUERY(dir_2 <* http_class_directory
          | (dir_1.name = dir_2.name) AND
          (dir_1.class.defined_by = dir_2.class.defined_by))
          <> [dir_1])
          = [];
END_RULE; -- unique_http_directory_name_per_supplier_rule
(*
```

Formal propositions:

WR1: all the **http_class_directory**'s associated with the same **supplier_BSU** shall have different names.

13.9.3 No_http_directory_for_supplier_related_file_rule rule

The **no_http_directory_for_supplier_related_file_rule** rule checks that all the **http_file**s belonging to a **supplier_BSU_related_content** are not associated with an **http_class_directory**.

NOTE The name of the directory intended to contain **http_file**s belonging to **supplier_BSU_related_content** should be implementation dependent.

EXAMPLE One particular implementation may use the supplier code of a supplier as the name of the directory intended to contain the **http_file**s belonging to **supplier_BSU_related_content** relating to this supplier. If the length of this name can not be supported on the user LMS, an implementation dependent abbreviation might be used.

EXPRESS specification:

```
*)
RULE no_http_directory_for_supplier_related_file_rule FOR(
     http_file, supplier_bsu_related_content);
WHERE
     WR1: QUERY(http_f <* http_file |
          (('ISO13584_EXTERNAL_FILE_SCHEMA' +
          '.SUPPLIER_BSU_RELATED_CONTENT'
          IN TYPEOF(http_f\external_file_unit.unit_of
          .content_of.content_of))
          AND EXISTS(http_f.http_directory))) = [];
END_RULE; -- no_http_directory_for_supplier_related_file_rule
```



```
(*
```

Formal propositions:

WR1: all the **http_file**s belonging to a **supplier_BSU_related_content** shall not be associated with an **http_class_directory**.

13.9.4 Http_directory_refers_to_bsu_related_class_rule rule

The **http_directory_refers_to_bsu_related_class_rule** rule checks that all the **http_file**s belonging to a **class_BSU_related_content** are associated with an **http_class_directory**.

NOTE The **name** of the **http_class_directory** where the LMS shall store the **http_file**s corresponding to each library class is defined by the library data supplier. This mechanism enable the library data supplier to define hypertext links between documents that remain valid once the documents are stored on the user site.

EXPRESS specification:

```
*)
RULE http_directory_refers_to_bsu_related_class_rule FOR(
     http_file, class_bsu_related_content);
WHERE
     WR1: QUERY(http_f <* http_file |
          (('ISO13584_EXTERNAL_FILE_SCHEMA' +
          '.CLASS_BSU_RELATED_CONTENT'
          IN TYPEOF(http_f\external_file_unit.unit_of
          .content_of.content_of))
          AND (http_f.http_directory.class <> http_f\
          external_file_unit.unit_of.content_of.content_of
          \content_item.dictionary_definition.name_scope))) = [];
END_RULE; -- http_directory_refers_to_bsu_related_class_rule
(*
```

Formal propositions:

WR1: all the **http_file**s belonging to a **class_BSU_related_content** shall be associated with an **http_class_directory**.

13.9.5 Http_directory_refers_to_class_extension_rule rule

The **http_directory_refers_to_class_extension_rule** rule checks that for each **http_file** belonging to a **class_extension_external_item**, the **class_BSU** referenced by this **class_extension_external_item** is the same as the **class_BSU** referenced by the **http_class_directory** intended to contain this **http_file**.

EXPRESS specification:

```
*)
RULE http_directory_refers_to_class_extension_rule FOR(
     http_file, class_extension_external_item);
WHERE
     WR1: QUERY(http_f <* http_file |
          ('ISO13584_EXTERNAL_FILE_SCHEMA' +
```



```
            '.CLASS_EXTENSION_EXTERNAL_ITEM'
            IN TYPEOF(http_f\external_file_unit.unit_of
            .content_of.content_of))
            AND (http_f.http_directory.class <> http_f\
            external_file_unit.unit_of.content_of.content_of
            \class_extension_external_item.belongs_to
            \content_item.dictionary_definition)) = [];
END_RULE; -- http_directory_refers_to_class_extension_rule
(*
```

Formal propositions:

WR1: for each **http_file** belonging to a **class_extension_external_item**, the **class_BSU** referenced by this **class_extension_external_item** shall be the same as the **class_BSU** referenced by the **http_class_directory** intended to contain this **http_file**.

13.9.6 Illustration_is_not_a_referenced_graphics_rule rule

The **illustration_is_not_a_referenced_graphics_rule** rule ensures that a **graphics** entity cannot be both a **referenced_graphics** and an **illustration.**

NOTE **graphics** is defined in the **ISO13584_IEC61360_dictionary_schema** and documented in ISO 13584-42. **referenced_graphics** is defined in the **ISO13584_extended_dictionary_schema**, documented in this part of ISO 13584. **illustration** is defined in the **ISO13584_exterenal_file_schema**, documented in this part of ISO 13584.

EXPRESS specification:

```
*)
RULE illustration_is_not_a_referenced_graphics_rule FOR(
     graphics);
WHERE
     WR1: QUERY(icon <* graphics |
          ('ISO13584_EXTENDED_DICTIONARY_SCHEMA.REFERENCED_GRAPHICS'
          IN TYPEOF(icon))
          AND ('ISO13584_EXTERNAL_FILE_SCHEMA.ILLUSTRATION'
          IN TYPEOF(icon))) = [];
END_RULE; -- illustration_is_not_a_referenced_graphics_rule
(*
```

Formal propositions:

WR1: a **graphics** entity cannot be both a **referenced_graphics** and an **illustration**.

13.10 ISO13584_external_file_schema entity definitions: external content

This subclause introduces the resource constructs for associating an **external_item** with the library external file(s) that defines its content. This content may be provided in different languages.

An **external_content** is the information model of the above content. It may or may not be translated and consists of **language_specific_content**.

Each **language_specific_content** describes the content in one specific language. This information may be provided by several **external_file_unit**s.

Each **external_file_unit** corresponds to one library external file. It is an **http_file** when the referenced **external_file_protocol** is the **http_protocol**.

The following planning model outlines the relationships between an **external_item** and the library external file(s) that define(s) its content.

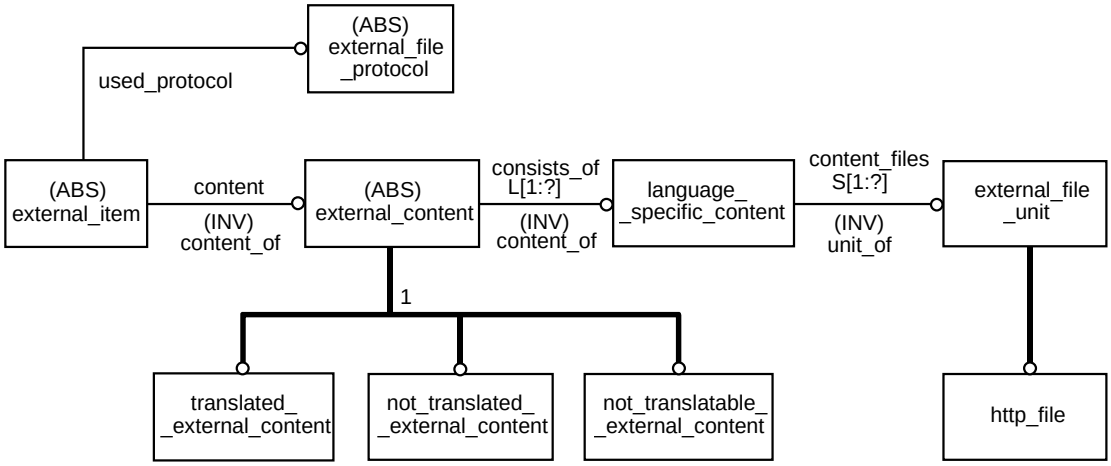


Figure 10 — External_content planning model

NOTE This planning model uses the EXPRESS-G graphical notation for the EXPRESS language, but, for clarification of the diagram, some of the relationships that are defined in the EXPRESS model are omitted, and inter-schema references are not represented.

13.10.1 External_content

An **external_content** entity is the content of an **external_item**.

EXPRESS specification:

```
*)
ENTITY external_content
ABSTRACT SUPERTYPE OF(ONEOF(
     translated_external_content,
     not_translated_external_content,
     not_translatable_external_content));
     consists_of: LIST[1:?] OF language_specific_content;
INVERSE
     content_of: external_item FOR content;
END_ENTITY; -- external_content
(*
```

Attribute definitions:

consists_of: the list of **language_specific_content**s that represent the content of the **external_item**.

NOTE The list order defined for the **external_content consists_of** attribute is only meaningful in the **translated_external_content** entity.

content_of: the **external_item** whose content is represented by the **external_content**.

13.10.2 Translated_external_content

A **translated_external_content** is the content of an **external_item**, provided in different languages.

EXPRESS specification:

```
*)
ENTITY translated_external_content
SUBTYPE OF(external_content);
      languages: present_translations;
WHERE
      WR1: SIZEOF(SELF\external_content.consists_of)
           = SIZEOF(SELF.languages.language_codes);
END_ENTITY; -- translated_external_content
(*
```

Attribute definitions:

languages: the different languages in which the **external_item** content is provided.

Formal propositions:

WR1: the number of existing **language_specific_content**s shall be equal to the number of translation languages.

Informal propositions:

IP1: the i-th **language_specific_content** specified by the **external_content consists_of** attribute provides the **external_item** in the i-th language specified by the **languages** attribute.

13.10.3 Not_translated_external_content

A **not_translated_external_content** is the content of an **external_item** that is provided in one particular language defined by a **global_language_assignment**.

NOTE The **global_language_assignment** entity is defined in the **language_resource_schema** documented in IEC 61360-2.

EXPRESS specification:

```
*)
ENTITY not_translated_external_content
SUBTYPE OF(external_content);
WHERE
      WR1: SIZEOF(SELF\external_content.consists_of) = 1;
END_ENTITY; -- not_translated_external_content
(*
```


Informal propositions:

IP1: the **language_specific_content** specified by the **external_content consists_of** attribute provides the **external_item** in the language specified by a **global_language_assignment** entity.

13.10.4 Not_translatable_external_content

A **not_translatable_external_content** is the content of an **external_item** that may be used in any particular language.

EXAMPLE A geometric **representation** without any textual information is a **not_translatable_external_content** that may be associated with a **representation_reference**

EXPRESS specification:

```
*)
ENTITY not_translatable_external_content
SUBTYPE OF(external_content);
WHERE
     WR1: SIZEOF(SELF\external_content.consists_of) = 1;
END_ENTITY; -- not_translatable_external_content
(*
```

Informal propositions:

IP1: the **language_specific_content** shall be meaningful in any particular language.

13.10.5 Language_specific_content

A **language_specific_content** is the content of an **external_item**, in a particular language. It may consist of several **external_file_unit**s with internal references, one of them being possibly the **main_file** file. When there exists a **main_file** file, all the references to the **external_item** stand for references to this file. All other **external_file_unit**s may only be accessed from the **main_file** file by the internal referencing mechanism specific to the **external_file_protocol** used.

EXAMPLE The **http_protocol** defines an internal referencing mechanism by means of hyper-text links. When this **external_file_protocol** is used for some **external_item**, wherever this **external_item** is referenced, the **main_file** file shall be displayed by the user LMS. Other **external_file_unit**s may only be accessed by the user and displayed by the user LMS through hyper-text links from the **main_file** file.

EXPRESS specification:

```
*)
ENTITY language_specific_content;
     content_files: SET [1:?] OF external_file_unit;
     main_file: OPTIONAL external_file_unit;
     character_encoding: OPTIONAL character_set_type;
INVERSE
     content_of: external_content FOR consists_of;
WHERE
     WR1: NOT EXISTS(main_file) OR (main_file IN content_files);
     WR2: EXISTS(main_file) XOR
          ('ISO13584_EXTERNAL_FILE_SCHEMA.PROGRAM_LIBRARY_CONTENT'
          IN TYPEOF(SELF.content_of.content_of));
```



```
      WR3: EXISTS(character_encoding) OR NOT
           ('ISO13584_EXTERNAL_FILE_SCHEMA.HTTP_PROTOCOL'
           IN TYPEOF(SELF.content_of.content_of.used_protocol));
END_ENTITY; -- language_specific_content
(*
```

Attribute definitions:

content_files: the **external_file_unit**s that define the content of the **external_item** in a particular language.

main_file: the **external_file_unit** that is to be processed (for instance, displayed) wherever the **external_item** is referenced to be processed during library usage.

character_encoding: the particular character encoding used in all the **external_file_unit**s of the **language_specific_content** that contain characters.

content_of: the **external_content** that is defined in one particular language by the **language_specific_content**.

Formal propositions:

WR1: the **main_file** shall be one of the **content_files**.

WR2: there shall exist a **main_file** but if the **external_item** is a **program_library_content**.

WR3: the **character_encoding** shall exist when the **http_protocol** is used.

Informal propositions:

IP1: the **character_encoding** shall exist when ever the **language_specific_content** contains characters and the **data_protocol** used enables several **character_encoding**s.

13.10.6 External_file_unit

An **external_file_unit** is a library external file.

EXPRESS specification:

```
*)
ENTITY external_file_unit
SUPERTYPE OF(http_file);
      file: external_file_address;
      content_encoding: OPTIONAL content_encoding_type;
INVERSE
      unit_of: language_specific_content FOR content_files;
WHERE
      WR1: (('ISO13584_EXTERNAL_FILE_SCHEMA.HTTP_PROTOCOL'
           IN TYPEOF(SELF.unit_of.content_of.content_of.used_protocol))
           AND ('ISO13584_EXTERNAL_FILE_SCHEMA.HTTP_FILE'
           IN TYPEOF(SELF)))
           XOR NOT
```



```
          (('ISO13584_EXTERNAL_FILE_SCHEMA.HTTP_PROTOCOL'
          IN TYPEOF(SELF.unit_of.content_of.content_of.used_protocol))
          OR ('ISO13584_EXTERNAL_FILE_SCHEMA.HTTP_FILE'
          IN TYPEOF(SELF)));
END_ENTITY; -- external_file_unit
(*
```

Attribute definitions:

file: the library external file represented by the **external_file_unit**.

content_encoding: if present, the encoding transformation performed on the content of the library external file represented by the **external_file_unit**.

NOTE The allowed encoding transformation are those defined by IAB RFC 2045.

unit_of: the **language_specific_content** to which the **external_file_unit** belongs.

Formal propositions:

WR1: the **external_file_unit** shall not correspond to an **external_item** associated with the **http_protocol**, otherwise it shall be an **http_file** subtype.

13.10.7 Http_file

An **http_file** is a library external file that is a MIME-like file and that may contain references to other Internet resources by means of http URLs. An **http_file** is associated with the **http_protocol**. It references in particular, an **http_file_name_type**, that specifies the name that shall be associated to this file on the user local Internet server, and when this file relates to a class, an **http_class_directory** that specifies the name of the directory where this file shall be stored on the user local Internet server.

NOTE 1 When an **http_file** relates to a **supplier_BSU_related_document**, the name of its directory on the user local Internet server is not specified; therefore it cannot be accessed by library data supplier-defined hyper-text links.

EXPRESS specification:

```
*)
ENTITY http_file
SUBTYPE OF(external_file_unit);
     mime: MIME_type;
     exchange_format: MIME_subtype;
     format_RFC: OPTIONAL IAB_RFC;
     http_file_name: http_file_name_type;
     http_directory: OPTIONAL http_class_directory;
     remote_access: OPTIONAL absolute_URL_type;
UNIQUE
     UR1: http_file_name, http_directory;
WHERE
     WR1: EXISTS(http_directory) XOR
          ('ISO13584_EXTERNAL_FILE_SCHEMA.SUPPLIER_BSU_RELATED_CONTENT'
          IN TYPEOF(SELF.unit_of.content_of.content_of));
     WR1: NOT EXISTS(http_directory) XOR
     ('ISO13584_EXTERNAL_FILE_SCHEMA.CLASS_EXTENSION_EXTERNAL_ITEM'
```



```
            IN TYPEOF(SELF.unit_of.content_of.content_of));
END_ENTITY; -- http_file
(*
```

Attribute definitions:

mime: the MIME type of the http file.

exchange_format: the MIME subtype of the http file.

format_RFC: the possible IAB RFC that defines the MIME subtype.

EXAMPLE 1 IAB RFC 1866 defines the HTML MIME text subtype.

EXAMPLE 2 IAB RFC 2376 [6] is an informational protocol that defines the XML MIME text subtype and the XML MIME application subtype.

http_file_name: the file name to be assigned to the http file on the local Internet server.

http_directory: the optional directory to be assigned to the http file on the local Internet server.

remote_access: the possible absolute URL where the http file may be found on an Internet site.

Formal propositions:

UR1: the **http_file_name** shall be unique within an **http_class_directory**.

WR1: the **http_class_directory** shall exist but if the **http_file** relates to a **supplier_BSU_related_content**.

WR1: the **http_class_directory** shall not exist but if the **http_file** relates to a **class_extension_external_item**.

Formal propositions:

IP1: only **mime** MIME type and **exchange_format** MIME subtype values that are registered by IANA are allowed for use by this part of ISO 13584.

NOTE 2 Apart private agreement between the sender and the receiver, this part of ISO 13584 strongly recommends to restrict to a small number of formats that are:

— mature;

— stable;

— unambiguously characterised by MIME Content-Type, i.e., type and subtype;

— publicly available or associated with public domain Internet-available readers.

NOTE 3 Format of MIME-like files whose Content-Type corresponds to specifications that are not publicly available, and that are not associated with public domain Internet-available readers are forbidden for any non extended conformance class of any of the three library integrated information models defined in this part of ISO 13584. This constraint is stated in annex E, I and M.

13.10.8 Http_class_directory

An **http_class_directory** entity specifies the name of the directory where an **http_file** shall be stored on the user local Internet server. Each **http_class_directory** refers to one **class** and each class that is associated with **http_file**s is referenced by exactly one **http_class_directory**.

EXPRESS specification:

```
*)
ENTITY http_class_directory;
      name: http_directory_name_type;
      class: class_BSU;
UNIQUE
      UR1: class;
END_ENTITY; -- http_class_directory
(*
```

Attribute definitions:

name: the name of the directory where an **http_file** shall be stored on the user local Internet server.

class: the referred class of the **http_class_directory**.

Formal propositions:

UR1: there shall be at most one **http_class_directory** per class.

13.11 ISO13584_external_file_schema function definitions

13.11.1 Supplier_associated_http_files

The **supplier_associated_http_files** function computes all the **http_file**s associated with the different **supplier_related_BSU**s that correspond to the same **supplier_BSU**. If the **supplier_element** is not available, the function returns the empty set.

NOTE All the **http_file**s associated with the different **supplier_related_BSU**s that correspond to the same **supplier_BSU** are intended to be stored in the same directory on the user local Internet server.

EXPRESS specification:

```
*)
FUNCTION supplier_associated_http_files(sup_BSU: supplier_BSU):
      SET OF http_file;

LOCAL
      sup: supplier_element;
      files: SET OF http_file := [];
END_LOCAL;

IF SIZEOF(sup_BSU.definition) > 0
      THEN sup := sup_BSU.definition[1];
ELSE
      RETURN(files);
```

```
END_IF;

REPEAT i := 1 TO SIZEOF(sup.associated_items);
-- supplier_BSU_relationship
     REPEAT j := 1 TO SIZEOF(sup.associated_items[i].related_tokens);
     --supplier_related_BSU
          REPEAT k := 1 TO SIZEOF(sup.associated_items[i]
                    .related_tokens[j].referenced_by); --content_item

               IF ('ISO13584_EXTERNAL_FILE_SCHEMA.EXTERNAL_ITEM'IN
                    TYPEOF(sup.associated_items[i]
                    .related_tokens[j].referenced_by[k]))
               THEN
                    REPEAT l := 1 TO SIZEOF(sup.associated_items[i]
                         .related_tokens[j].referenced_by[k]
                         \external_item.content.consists_of);
                         --language_specific_content

                         REPEAT m := 1 TO SIZEOF(
                              sup.associated_items[i]
                              .related_tokens[j].referenced_by[k]
                              \external_item.content
                              .consists_of[l].content_files);
                              -- external_file_unit
                              IF
                              ('ISO13584_EXTERNAL_FILE_SCHEMA.HTTP_FILE'
                                   IN TYPEOF(sup.associated_items[i]
                                   .related_tokens[j].referenced_by[k]
                                   \external_item.content
                                   .consists_of[l].content_files[m]))
                              THEN
                                   files := files +
                                   sup.associated_items[i]
                                   .related_tokens[j].referenced_by[k]
                                   \external_item.content
                                   .consists_of[l].content_files[m];
                              END_IF; -- http_file
                         END_REPEAT; -- m
                    END_REPEAT; -- l
               END_IF; -- external_item
          END_REPEAT; -- k
     END_REPEAT; -- j
END_REPEAT; -- i

RETURN(files);

END_FUNCTION; -- supplier_associated_http_files
(*
```


13.11.2 Control_compiler_version_format

The **control_compiler_version_format** returns TRUE if the **the_compiler_version** string contains only digits, dots, or underscores. Otherwise, it returns FALSE.

EXPRESS specification:

```
*)
FUNCTION control_compiler_version_format(the_compiler_version: STRING):
     BOOLEAN;
LOCAL
     result: BOOLEAN := TRUE;
END_LOCAL;

REPEAT i := 1 TO LENGTH(the_compiler_version);
     IF (NOT((the_compiler_version[i] LIKE '#')
          OR (the_compiler_version[i] LIKE '.')
          OR (the_compiler_version[i] LIKE '_')))
     THEN
          result := FALSE;
     END_IF;
END_REPEAT;
     RETURN(result);
END_FUNCTION; -- control_compiler_version_format
(*

*)
END_SCHEMA; -- ISO13584_external_file_schema
(*
```

14 ISO13584_method_schema

This clause defines the requirements for the **ISO13584_method_schema**. The following EXPRESS declaration introduces the **ISO13584_method_schema** block and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA ISO13584_method_schema;

REFERENCE FROM ISO13584_IEC61360_dictionary_schema
     (all_class_descriptions_reachable,
     class_BSU,
     content_item,
     definition_available_implies,
     list_to_set,
     property_BSU);

REFERENCE FROM ISO13584_library_expressions_schema
     (class_instance_constructor,
     class_instance_expression,
```

```
      collects_assigned_properties,
      collects_referenced_library_expressions,
      compatible_variable_and_library_expression,
      library_expression,
      library_variable,
      property_assignment);

  REFERENCE FROM ISO13584_variable_semantics_schema
      (property_semantics,
      property_semantics_or_path,
      self_property_value_semantics);

  REFERENCE FROM ISO13584_domain_resource_schema
      (collects_variables,
      functional_domain_restriction,
      used_variables_in_domain);

  REFERENCE FROM ISO13584_extended_dictionary_schema
      (abstract_functional_model_class,
      applicable_properties,
      applicable_tables,
      data_type_class_of,
      data_type_typeof,
      data_type_type_name,
      functional_model_class,
      functional_view_class,
      functional_view_v_c_v,
      view_control_variable_range);

  REFERENCE FROM ISO13584_generic_expressions_schema
      (generic_variable,
      used_variables);

  REFERENCE FROM ISO13584_expressions_schema
      (boolean_expression,
      numeric_expression,
      string_expression);

  REFERENCE FROM ISO13584_library_content_schema
      (exists_super,
      functional_model_class_extension,
      method_variables,
      model_class_extension,
      provided_properties_list,
      provided_properties_or_method_variables,
      super);

  REFERENCE FROM ISO13584_external_file_schema
      (external_file_protocol);

  (*
```


NOTE The schemas referenced above can be found in the following documents:

ISO13584_IEC61360_dictionary_schema	IEC 61360-2

(which is duplicated for convenience in informative annex D of ISO 13584-42),

ISO13584_expressions_schema	ISO 13584-20,
ISO13584_generic_expressions_schema	ISO 13584-20,
ISO13584_library_expressions_schema	This part of ISO 13584,
ISO13584_variable_semantics_schema	This part of ISO 13584,
ISO13584_domain_resource_schema	This part of ISO 13584,
ISO13584_extended_dictionary_schema	This part of ISO 13584,
ISO13584_library_content_schema	This part of ISO 13584,
ISO13584_external_file_schema	This part of ISO 13584.

14.1 Introduction to the ISO13584_method_schema

In the ISO 13584 standard series, part views are generated by functional model classes. Functional model classes described by **explicit_functional_model_class_extension** explicitly describe the various item views. Functional model classes described by **functional_model_class_extension** implicitly describe item views by means of methods. Such methods are triggered by LMS for creating views. The scope of the **ISO13584_method_schema** is to allow the description of library methods that enable the generation of item views, both for atomic items and for assembled items. In the latter case, methods may create representations of assembled items from the representations of its constituent components. The "bolt + nut + optional washer" assembly is an example of an assembled items. The resources provided in this schema enable the specification of a method that creates a "bolt + nut + optional washer" representation by triggering relevant methods on the bolt, the nut and, possibly, the washer.

The **ISO13584_method_schema** models:

— the methods that enable the creation of views of atomic items,

— the methods that enable the creation of views of assembled items by triggering methods on the constituent parts.

The **ISO13584_method_schema** does not model:

— the methods that enable the creation of views of parts that contains list-structured properties,

— the methods that contain repetitive or recursive control structures.

14.2 Fundamental concepts and assumptions for the ISO13584_method_schema

The following assumptions apply to the portions of this schema that deal with methods:

— supplier defined methods are only intended to produce functional views.

— functional views are **representation**s defined in their own representation context called the Object View Coordinate (OVC).

NOTE 1 The OVC is not necessarily a **geometric_representation_context**.

— the role of a method is to generate **representation**s that constitutes its created functional view, either from library-stored ISO 10303-conformant **representation**s, or from parametric programs. The view generation process is modelled in a procedural approach as an ordered list of statements to ensure that view generation is fully deterministic.

— when a method is triggered by the user, through the LMS, an unspecified positioning process is run by the system to allow the user to specify the transformation from the OVC representation context of the view onto the representation context of the modelling system. At the end of a user-triggered method, the functional view is created in the user modelling system and the LMS returns to an empty state.

NOTE 2 The positioning process is not necessarily geometric. It may consist for instance in relating the created view to an overall structure already existing in the user modelling system.

— a method consists of two entities: a **method_specif** entity, that specifies the different functional views a method is able to generate and the information requirements of the method; a **method_body** entity, that specifies, as an ordered list of statements, the method algorithm. Each statement of the list may be guarded or not. The guarded statement allows the introduction of an *if then else* construct.

— the statements that may comprise a method are:

  a) null statements;

  b) modelling statements that specify transformations of the OVC: these transformations are applied to all the **representation_items** created in the view after such a transformation has been specified;

NOTE 3 Such modelling statements are only meaningful when the OVC is a **geometric_representation_context**.

  c) predefined representation call statements that specify a call to a pre-existing implicit or explicit representation;

  d) assignment statement that assigns values to variables;

  e) subobject view statements that allow a method to trigger other methods on a subobject to compose an assembled item. The view created by the subobject method is mapped onto the representation context of the embedding view by applying the current transformation of the OVC to the **representation_items** created by the subobject method.

— a method may only generate functional views belonging to the same representation category, identified by a **class_BSU** entity.

— a method may generate several functional views belonging to the same representation category.

— a functional view may possibly contain a set of properties defined in its **functional_view_class dictionary_element**, and, as a subtype of a **representation**, it inherits an **items** attribute. The (possible defined) properties may be assigned a value. The inherited **items** attribute may only be filled by externally defined **representation**, by externally defined program associated with an **external_file_protocol**, or by internally defined **representation**.

— when the **items** attribute of a functional view is filled by an externally-defined or internally-defined **representation**,

  a) the OVC of the view is the **representation_context** of the representation,

  b) the items of the view is the content of the items attribute of the representation.

— methods are triggered through message passing. A message contains the following information:

  a) the required functional view, represented by the corresponding **class_BSU**;

  b) a list of view control variable values that specify precisely the required functional view.

— when triggered, a method is associated with one functional view: the functional view specified by the triggering message. This view is called the *open_view* of the running method.

— when it is performed, a method is always associated with one instance of a functional model class that may contain, amongst its properties, copies of properties of one instance of the **item_class**

the instance of the functional model class is a view of. If the instance of the **item_class** models an assembled part, the method may send messages to the subobjects of this assembled part or may instantiate functional models of these subobjects.

NOTE 4 The properties of an **item_class** instance whose values shall be copied into properties of a functional model class instance are modelled by the **required_item_characteristics** attribute of a **functional_model_class_extension** entity.

— when a method is triggered from within another method, a new *open_view* is associated with this method. The current transformation of the OVC of the *open_view* associated with the embedding method specifies the transformation that shall be performed on the representation context of the *open_view* associated with the embedded method to ensure adequate mapping of the created view within the representation context of the embedding view. This view is closed when the method returns.

— a message may be sent either to a general model, instance of an **item_class**, or to a functional model, instance of a **functional_model_class**.

— when an instance of an **item_class** that composes an assembled part is sent a message from a method that creates a view of its embedding part, the LMS searches, within the functional model classes that are is-view-of this **item_class**, a functional model class whose instances support this message. This class is instantiated and the created instance is sent the message. Hence a new functional view is generated.

— when an instance of a **functional_model_class** is sent a message, this instance shall support the method specified by the message. The **functional_model_class** instance is specified by a **class_instance_constructor** that completely characterises the instance. The **class_instance_constructor** is evaluated and the created instance is sent the message. The method is then triggered and a new functional view is generated.

14.3 ISO13584_method_schema type definitions

The following sections introduce the type definitions used in the **ISO13584_method_schema**.

14.3.1 Accessible_variable_for_method

An **accessible_variable_for_method** is a **library_variable** that may be referenced from a method. It can be a **library_variable** that represents **self_variable_semantics** of the instance that supports the method, a **library_variable** that represents **open_view_variable_semantics** of the *open_view* created by the method, or a **library_variable** that represents an iterator intended to be used in a table query. The latter case of **library_variable** is associated with a **column_traversal_variable_semantics**.

NOTE 1 In ISO 13584, each **library_variable** is associated, through an environment to a **variable_semantics**; conversely, each variable semantics is represented by no more than one variable. This is documented in the **two_fold_representation_rule** rule of the **ISO13584_library_expressions_schema**.

NOTE 2 All the properties whose values are represented in a **functional_model_class_extension** (as returned by the **provided_properties_or_method_variables** function) may be associated with a **library_variable** to be accessed by a method. But only the variables that are first declared in the **declaration** attribute of the method may be referred to in the method statements.

NOTE 3 The properties whose values are represented in a **functional_model_class_extension** (as returned by the **provided_properties_or_method_variables** function) may contain properties that are **imported_properties_from_view** (for instance, when view control variable values are stored within a table), or, in the case of a **fm_class_view_of**, properties that are **imported_properties_from_item**. Nevertheless, these properties belong to the SELF instance, i.e., the functional model instance that supports the method, and they are referenced by **self_property_value_semantics**.

NOTE 4 All the properties of the view created by the method may be associated with a variable to be accessed by this method. But only the variables that are declared in the **declaration** attribute of the method may be referred to in the method statements.

NOTE 5 All the view control variable values that were part of the triggering message of a method shall be assigned to the corresponding open view. Therefore these values are already bounded with the variables associated to the corresponding **open_view_property_value_semantics** when the **method** is triggered.

NOTE 6 When a property of a **functional_model_class_extension** is an **imported_properties_from_view**, during the running of the method, two **variable_semantics** are identified by the same **property_BSU**. The first is a **self_variable_semantics** that represents the corresponding property of the SELF instance. The second is an **open_view_property_semantics** that represents the corresponding property of the current open view. They shall be associated with two different variables. Their values may be different.

EXPRESS specification:

```
*)
TYPE accessible_variable_for_method = library_variable;
WHERE
     WR1:(('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
          +'SELF_VARIABLE_SEMANTICS')
          IN TYPEOF(SELF\generic_variable.interpretation
          .semantics))
          OR (('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
          +'OPEN_VIEW_VARIABLE_SEMANTICS')
          IN TYPEOF(SELF\generic_variable.interpretation
          .semantics))
          OR (('ISO13584_TABLE_RESOURCE_SCHEMA.'
          +'COLUMN_TRAVERSAL_VARIABLE_SEMANTICS')
          IN TYPEOF(SELF\generic_variable.interpretation
          .semantics));
END_TYPE; -- accessible_variable_for_method
(*
```

Formal propositions:

WR1: an **accessible_variable_for_method** shall be associated with a **self_variable_semantics**, with an **open_view_variable_semantics**, or with a **column_traversal_variable_semantics** intended to be used as an iterator for querying tables.

14.3.2 Assignment_allowed_variable

An **assignment_allowed_variable** is a variable that represents either the value of a property of the SELF instance, or a property of the current *open view.* Such a variable may be assigned a value in a method body.

EXPRESS specification:

```
*)
TYPE assignment_allowed_variable = library_variable;
WHERE
     WR1:(('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
          +'SELF_PROPERTY_VALUE_SEMANTICS')
```



```
            IN TYPEOF(SELF\generic_variable.interpretation
            .semantics))
            OR (('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
            +'OPEN_VIEW_PROPERTY_VALUE_SEMANTICS')
            IN TYPEOF(SELF\generic_variable.interpretation
            .semantics));
END_TYPE; -- assignment_allowed_variable
(*
```

Formal propositions:

WR1: an **assignment_allowed_variable** shall be associated either with a **self_property_value_semantics** or with an **open_view_property_value_semantics**.

14.3.3 Control_allowed_variable

A **control_allowed_variable** is a variable that may be used in a method for control purpose. For that purpose, **translatable** values are not allowed.

EXAMPLE If a variable is associated with a **self_property_short_name_semantics** the value of this variable depends on the language selected by the interpretation function to return this name (see clause 10.7.4.2).

EXPRESS specification:

```
*)
TYPE control_allowed_variable = library_variable;
WHERE
     WR1: (('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
          +'SELF_PROPERTY_VALUE_SEMANTICS')
          IN TYPEOF(SELF\generic_variable.interpretation
          .semantics))
          OR (('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
          +'OPEN_VIEW_PROPERTY_VALUE_SEMANTICS')
          IN TYPEOF(SELF\generic_variable.interpretation
          .semantics))
          OR (('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
          +'SELF_PROPERTY_CODE_SEMANTICS')
          IN TYPEOF(SELF\generic_variable.interpretation
          .semantics))
          OR (('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
          +'SELF_PROPERTY_VERSION_SEMANTICS')
          IN TYPEOF(SELF\generic_variable.interpretation
          .semantics))
          OR (('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
          +'SELF_PROPERTY_CLASS_CODE_SEMANTICS')
          IN TYPEOF(SELF\generic_variable.interpretation
          .semantics))
          OR (('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
          +'SELF_PROPERTY_CLASS_VERSION_SEMANTICS')
          IN TYPEOF(SELF\generic_variable.interpretation
          .semantics))
          OR (('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
          +'SELF_PROPERTY_CLASS_SUPPLIER_CODE_SEMANTICS')
```



```
            IN TYPEOF(SELF\generic_variable.interpretation
            .semantics))
            OR (('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
            +'SELF_CLASS_CODE_SEMANTICS')
            IN TYPEOF(SELF\generic_variable.interpretation
            .semantics))
            OR (('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
            +'SELF_CLASS_VERSION_SEMANTICS')
            IN TYPEOF(SELF\generic_variable.interpretation
            .semantics))
            OR (('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
            +'SELF_CLASS_SUPPLIER_CODE_SEMANTICS')
            IN TYPEOF(SELF\generic_variable.interpretation
            .semantics));
END_TYPE; -- control_allowed_variable
(*
```

Formal propositions:

WR1: a **control_allowed_variable** shall not be associated with a **variable_semantics** that represents a translated string.

14.4 ISO13584_method_schema entity definitions

The following clause describes the **ISO13584_method_schema** entities.

14.4.1 Method

A **method** is an operation that is defined in a **functional_model_class_extension** to be associated with instances of this class. Its role is to create a view.

EXPRESS specification:

```
*)
ENTITY method;
      specification: method_specif;
      body: method_body;
      representation_interface: OPTIONAL external_file_protocol;
INVERSE
      its_class: functional_model_class_extension
            FOR provided_methods;
WHERE
      WR1: (EXISTS(SELF.representation_interface))
            AND (SELF.representation_interface
            IN SELF.its_class\model_class_extension.used_protocols)
            OR (NOT(EXISTS(SELF.representation_interface)));
END_ENTITY; -- method
(*
```


Attribute definitions:

specification: the specification of the method.

body: the body of the method.

representation_interface: the protocol that may be required to run the method.

EXAMPLE If the method only sends to the user modelling system a library-stored ISO 10303-203-conformant **representation**s, the **representation_interface** shall contain a **standard_data_protocol** that references some conformance class of ISO 10303-203.

NOTE The content of a **standard_data_protocol** that enables reference to a conformance class of an Application Protocol of ISO 10303 is specified in ISO 13584-102[1].

its_class: the class to which the method belongs.

Formal propositions:

WR1: if an **external_file_protocol** is used by the **method**, it shall belong to the **used_protocols** set of the **model_class_extension** that contains the method.

14.4.2 Method_specif

A **method_specif** entity encapsulates the content of the method. It describes the views created by the method, the properties of the functional model that shall have values when the method is run, and, in the case of assemblies, the subobject classes that may be referred to by the method.

EXPRESS specification:

```
*)
ENTITY method_specif;
     created_view: class_BSU;
     v_c_v_range: SET [0:?] OF view_control_variable_range;
     model_needed_properties: SET [0:?] OF property_BSU;
     referred_subobject_models: SET [0:?] OF class_BSU;
INVERSE
     specifies: method FOR specification;
WHERE
     WR1: NOT all_class_descriptions_reachable(
          SELF.specifies.its_class.dictionary_definition)
          OR (SELF.model_needed_properties
          <= list_to_set(provided_properties_list(
          SELF.specifies.its_class.dictionary_definition)));
     WR2: same_view_model_method(SELF);
     WR3: SIZEOF(QUERY(models <* SELF.referred_subobject_models |
          definition_available_implies(models,
          ('ISO13584_IEC61360_DICTIONARY_SCHEMA.ITEM_CLASS'
          IN TYPEOF(models.definition[1])) OR (
          ('ISO13584_EXTENDED_DICTIONARY_SCHEMA'
          + '.FUNCTIONAL_MODEL_CLASS' IN TYPEOF(models.definition[1])))
          ))) = SIZEOF(SELF.referred_subobject_models);
```

[1] To be published


```
END_ENTITY; -- method_specif
(*
```

Attribute definitions:

created_view: the view created by the **method**.

v_c_v_range: the set of values of the **view_control_variable** for which the model is able to create a view.

model_needed_properties: the properties of the instance that supports the method that need to have a value to run the method.

NOTE 1 The **model_needed_properties** may have a value that is the EXPRESS indeterminable value ("?") if the property was defined as optional in the **functional_model_class_extension**.

NOTE 2 The role of the **model_needed_properties** is to specify:

— which **selectable_properties** shall be provided by the user in order, for the method, to be run, and

— in the case of a partially defined item, which **required_properties** (from the general model) shall have values in order, for the method, to be run.

NOTE 3 The **model_needed_properties** and the *open_view* view control variables are the only values that shall be initialized by the LMS when triggering a method.

referred_subobject_models: the possible classes referred to in the **method_body**.

specifies: the **method** whose specification is defined.

Formal propositions:

WR1: the **model_needed_properties** properties shall belong to the result of the **provided_properties_list** for the **functional_model_class_extension** to which the method belongs.

NOTE 4 As documented in this where rule, method variables shall not be represented in the **model_needed_properties** attribute.

WR2: the **created_view** shall be the view created by the **functional_model_class**.

WR3: if data are available, then **IP1** holds.

Informal propositions:

IP1: the **referred_subobject_models** shall be defined either as an **item_class** or as a **functional_model_class**.

14.4.3 Method_body

A **method_body** is a set of variables, that specifies the **accessible_variable_for_method**s that are accessed from the **method**, and a list of statements, that specifies the deterministic process the **method** shall perform.

NOTE Only those view properties intended to be accessed by the method body need to be represented by variables in the **declaration** set.

Performing a method is a two phase process:

a) In the elaboration phase:

   1) the current view, as specified by the **method** triggering message, is created, and

   2) the required values of the current views' view control variables are assigned, as values, to the corresponding view properties whether they are referred to by means of **open_view_property_value_semantics**, and

   3) the context of the **method**, that consists of the **declaration** set of **accessible_variable_for_method**s, is created by the LMS, and

   4) values are elaborated, possibly by triggering some derivation functions, for the variables in the **declaration** set that correspond to **model_needed_properties**,

b) In the running phase, the **method** statements are performed. Only the **accessible_variable_for_method**s belonging to the context of the **method** may be referred to as variables from the **method** statements.

EXPRESS specification:

```
*)
ENTITY method_body;
     declaration: SET [1:?] OF accessible_variable_for_method;
     view_generation: LIST [1:?] OF method_statement;
INVERSE
     describes: method FOR body;
WHERE
     WR1: QUERY(prop <*
          SELF.describes.specification.model_needed_properties |
          SIZEOF(QUERY(v <* SELF.declaration |
          ('ISO13584_VARIABLE_SEMANTICS_SCHEMA.PROPERTY_SEMANTICS'
          IN TYPEOF(v\generic_variable.interpretation.semantics))
          AND (v\generic_variable.interpretation.semantics\
          property_semantics.the_property :=: prop))) <> 1) = [];
     WR2: QUERY(v <* SELF.declaration |
          ('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
          + 'SELF_PROPERTY_SEMANTICS' IN TYPEOF (
          v\generic_variable.interpretation.semantics)) AND
          NOT(v\generic_variable.interpretation
          .semantics\property_semantics.the_property
          IN provided_properties_or_method_variables (
          SELF.describes.its_class.dictionary_definition)))
          = [];
     WR3: QUERY(v <* SELF.declaration |
          ('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
          + 'OPEN_VIEW_PROPERTY_SEMANTICS' IN TYPEOF(
          v\generic_variable.interpretation.semantics))
          AND NOT applicable_properties(
          SELF.describes.specification.created_view,
          [v\generic_variable.interpretation.semantics\
          property_semantics.the_property])) = [];
     WR4: QUERY(v <* SELF.declaration |
          ('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
          + 'SELF_PROPERTY_SEMANTICS' IN TYPEOF
```



```
            (v\generic_variable.interpretation.semantics))
            AND NOT checks_applicable_properties_in_path(
            v\generic_variable.interpretation.semantics)) = [];
       WR5: QUERY(v <* SELF.declaration |
            ('ISO13584_VARIABLE_SEMANTICS_SCHEMA.'
            + 'SELF_PROPERTY_SEMANTICS' IN TYPEOF
            (v\generic_variable.interpretation.semantics))
            AND NOT checks_classes_in_path(v\generic_variable
            .interpretation.semantics,
            SELF.describes.specification.referred_subobject_models))
            = [];
  END_ENTITY; -- method_body
  (*
```

Attribute definitions:

declaration: the SET of **accessible_variable_for_method**s that are to be created in the **method** context before performing the statements of the **method**.

view_generation: the sequence of statements the **method** shall perform.

describes: the method whose content is defined by the **method_body**.

Formal propositions:

WR1: all the **model_needed_properties** properties specified in the **method_specif** shall be referenced by a **accessible_variable_for_method** in the **declaration** set of the **method_body**.

WR2: the properties of the SELF instance referred in by a **variable** in the **declaration** attribute shall belong to the properties represented in the **functional_model_class_extension** instance as returned by the **provided_properties_or_method_variables** function.

WR3: the properties of the current *open view* referred to by an **accessible_variable_for_method** shall be **applicable_properties** for the functional view class to which the current *open view* belongs.

WR4: all the properties referred to in a **sub_property_path** of a **self_property_semantics** shall be **applicable_properties** for the class to which they belong.

WR5: all the properties referred to in a **sub_property_path** of a **self_property_semantics** that are of **class_instance_type** shall belong to the **referred_subobject_models** of the **method_specif**.

14.4.4 Method_statement

A **method_statement** corresponds to each operation a **method** shall perform. It contains a list of **guarded_statement**s.

Performing a **method_statement** consists of evaluating each **guard** that guards each **simple_statement**. If some of them evaluate to TRUE, any **simple_statement** whose guard evaluates to TRUE is performed. If all the guards evaluate to FALSE, no **simple_statement** shall be performed.

NOTE If only one **simple_statement** is always performed, only one **guarded_statement** shall be specified. Its **guard** is the **boolean_literal** TRUE, as defined by ISO 13584-20.

EXPRESS specification:

```
*)
ENTITY method_statement;
     statements: LIST [1:?] OF guarded_statement;
INVERSE
     defines: method_body FOR view_generation;
END_ENTITY; -- method_statement
(*
```

Attribute definitions:

statements: the list of statements of which one (or more) is to be performed if its corresponding **guard** evaluates to TRUE.

defines: the **method_body** to which the **method_statement** belongs.

14.4.5 Guarded_statement

A **guarded_statement** is a statement that is associated with a **boolean_expression** defining the **guard**.

NOTE The **guard** may be a **boolean_literal** whose **the_value** attribute is TRUE. Such a **guard** always evaluates to TRUE.

EXPRESS specification:

```
*)
ENTITY guarded_statement;
     guard: boolean_expression;
     statement: simple_statement;
INVERSE
     item_of: method_statement FOR statements;
WHERE
     WR1: QUERY(elt <* used_variables(SELF.guard) |
          NOT('ISO13584_METHOD_SCHEMA.CONTROL_ALLOWED_VARIABLE' IN
          TYPEOF(elt))) = [];
     WR2: QUERY(elt <* used_variables(SELF.guard) |
          NOT(elt IN SELF.item_of.defines.declaration)) = [];
END_ENTITY; -- guarded_statement
(*
```

Attribute definitions:

guard: the **boolean_expression** that specifies whether the statement may be performed.

statement: the statement that is performed when the **guard** evaluates to TRUE. In case of several **guard**s evaluating to TRUE, the **statement** is selected by the system.

item_of: the **method_statement** that contains the **guarded_statement**.

Formal propositions:

WR1: all the variables involved in the **boolean_expression** that constitutes the **guard** shall be **control_allowed_variables**.

WR2: all the **control_allowed_variable**s involved in the **boolean_expression** that constitutes the **guard** shall belong to the context of the method (i.e., the **declaration** set of the **method_body**).

14.4.6 Simple_statement

A **simple_statement** is the basic statement that composes a **method_body**. It can either be a null statement, a modelling statement, a call to a representation, an assignment, or a subobject view statement.

EXPRESS specification:

```
*)
ENTITY simple_statement
ABSTRACT SUPERTYPE OF(ONEOF(
     null_statement,
     modelling_statement,
     predefined_representation_call_statement,
     assignment_statement,
     sub_object_view_statement));
INVERSE
     referenced_by: guarded_statement FOR statement;
END_ENTITY; -- simple_statement
(*
```

Attribute definitions:

referenced_by: the **guarded_statement** that references the **simple_statement**.

14.4.7 Null_statement

A **null_statement** specifies that no action shall be performed.

EXPRESS specification:

```
*)
ENTITY null_statement
SUBTYPE OF(simple_statement);
END_ENTITY; -- null_statement
(*
```

14.4.8 Modelling statement

A **modelling_statement** allows to modify the OVC of the current open view. **Modelling_statement**s enable the definition of a set structure for the **representation_item**s that constitute the **items** of the view, and, for views whose representation context is a **geometric_representation_context**, to define the geometric positioning of the **geometric_representation_items**.

EXPRESS specification:

```
*)
ENTITY modelling_statement
ABSTRACT SUPERTYPE OF(ONEOF(
      set_reference_lcs,
      begin_set,
      close_set,
      set_2d_relative_view_level))
SUBTYPE OF(simple_statement);
END_ENTITY; -- modelling_statement
(*
```

14.4.9 Set_reference_lcs

For OVCs that are in a **geometric_representation_context**, the **set_reference_lcs** entity specifies that the reference local coordinate system (LCS) that defines the positioning of the next entities created shall be moved relative to the current reference coordinate system of the OVC. The new position of the reference coordinate system defines the mapping target on which the space coordinates of the next entities created are to be mapped.

The order of the attributes of a **set_reference_lcs** defines the order of the transformations that shall be performed to define the new reference coordinate system, with first a rotation around the x axis and last a translation along the z axis.

The **length_unit** and **planar_angle_unit** are those defined in the **global_unit_assigned_context** that defines the measure units for all the functional views created by the **functional_model_class_extension** to which the method belongs. If no **global_unit_assigned_context** is provided in the **functional_model_class_extension** the method belongs to, the default value for **length_measure** is millimetre and for **planar_angle_measure** it is degree.

EXAMPLE When a method that provides a functional view of a nut is triggered from within a method associated with a functional model that is is-view-of a "bolt + nut" assembly, the OVC reference coordinate system shall be changed in order, for the nut representation, to be well positioned in the "bolt + nut" OVC. Subsequently, the triggering method shall move the reference coordinate system of the "bolt + nut" view. Then, the OVC of the nut is implicitly mapped onto the current reference coordinate system of the embedding view.

NOTE Changes in reference coordinate system are local to a method. When a method returns, the reference coordinate system of the view of the embedding method is reset by the system at the value it had in the possible embedding view.

EXPRESS specification:

```
*)
ENTITY set_reference_lcs
SUBTYPE OF(modelling_statement);
      x_rotation: numeric_expression;
      y_rotation: numeric_expression;
      z_rotation: numeric_expression;
      x_translation: numeric_expression;
      y_translation: numeric_expression;
      z_translation: numeric_expression;
WHERE
      WR1: QUERY(elt <* used_variables(SELF.y_rotation) |
           NOT('ISO13584_METHOD_SCHEMA.CONTROL_ALLOWED_VARIABLE'
```



```
            IN TYPEOF(elt))) +
            QUERY(elt <* used_variables(SELF.z_rotation) |
            NOT('ISO13584_METHOD_SCHEMA.CONTROL_ALLOWED_VARIABLE'
            IN TYPEOF(elt))) +
            QUERY(elt <* used_variables(SELF.x_translation) |
            NOT('ISO13584_METHOD_SCHEMA.CONTROL_ALLOWED_VARIABLE'
            IN TYPEOF(elt))) +
            QUERY(elt <* used_variables(SELF.y_translation) |
            NOT('ISO13584_METHOD_SCHEMA.CONTROL_ALLOWED_VARIABLE'
            IN TYPEOF(elt))) +
            QUERY(elt <* used_variables(SELF.z_translation) |
            NOT('ISO13584_METHOD_SCHEMA.CONTROL_ALLOWED_VARIABLE'
            IN TYPEOF(elt))) +
            QUERY(elt <* used_variables(SELF.x_rotation) |
            NOT('ISO13584_METHOD_SCHEMA.CONTROL_ALLOWED_VARIABLE'
            IN TYPEOF(elt)))
            = [];
      WR2: QUERY(elt <* used_variables(SELF.x_rotation) |
            NOT(elt IN SELF\simple_statement.referenced_by
            .item_of.defines.declaration)) +
            QUERY(elt <* used_variables(SELF.y_rotation) |
            NOT(elt IN SELF\simple_statement.referenced_by
            .item_of.defines.declaration)) +
            QUERY(elt <* used_variables(SELF.z_rotation) |
            NOT(elt IN SELF\simple_statement.referenced_by
            .item_of.defines.declaration)) +
            QUERY(elt <* used_variables(SELF.x_translation) |
            NOT(elt IN SELF\simple_statement.referenced_by
            .item_of.defines.declaration)) +
            QUERY(elt <* used_variables(SELF.y_translation) |
            NOT(elt IN SELF\simple_statement.referenced_by
            .item_of.defines.declaration)) +
            QUERY(elt <* used_variables(SELF.z_translation) |
            NOT(elt IN SELF\simple_statement.referenced_by
            .item_of.defines.declaration))
            = [];
END_ENTITY; -- set_reference_lcs
(*
```

Attribute definitions:

x_rotation: rotation, in **planar_angle_measure**, of the new LCS relative to the x axis of the current LCS.

y_rotation: rotation, in **planar_angle_measure**, of the new LCS relative to the y axis of the current LCS.

z_rotation: rotation, in **planar_angle_measure**, of the new LCS relative to the z axis of the current LCS.

x_translation: translation, in **length_measure**, of the new LCS relative to the x axis of the current LCS.

y_translation: translation, in **length_measure**, of the new LCS relative to the y axis of the current LCS.

z_translation: translation, in **length_measure**, of the new LCS relative to the z axis of the current LCS.

Formal propositions:

WR1: all the variables involved in the **numeric_expression**s shall be **control_allowed_variables**.

WR2: all the **control_allowed_variable**s involved in the **numeric_expression**s shall belong to the context of the method (i.e., the **declaration** set of the **method_body**).

14.4.10 Begin_set

The **begin_set** entity declares that a new set shall be open inside the modelling system, or inside the current open set inside the modelling system. After this entity is performed, all the representation elements created inside the *open_view* belong to this set.

EXAMPLE In a method that produces a "bolt + nut" solid geometry functional view through two different programs referring, respectively to bolt and nut, this entity enables to structure the set of geometric entities of the assembly into two sets. One contains the nut geometry. The other contains the bolt geometry. The modelling system may then allow the user to access this set structure.

NOTE 1 All the sets opened by a method are local to the view created by the method. When a method returns upon completion, all the sets opened by the method are closed and the set structure is reset by the system at the state it had in the possible embedding view.

NOTE 2 View exchange protocol shall specify the required set structure of the view they define.

EXPRESS specification:

```
*)
ENTITY begin_set
SUBTYPE OF(modelling_statement);
     set_name: string_expression;
WHERE
     WR1: QUERY(elt <* used_variables(SELF.set_name)
          | NOT(elt IN SELF\simple_statement.referenced_by.item_of
          .defines.declaration)) = [];
END_ENTITY; -- begin_set
(*
```

Attribute definitions:

set_name: the label of the new open set in the current open set.

Formal propositions:

WR1: all the **variable**s involved in the **string_expression** shall belong to the context of the method (i.e., the **declaration** set of the **method_body**).

Informal propositions:

IP1: No other set with the **set_name** name shall exist within the current open set.

14.4.11 Close_set

The **close_set** entity declares that one current open set shall be closed. It declares the name of the set that shall be closed, and this name shall correspond to a set that was open by the current method. If this set contains some other open sets, they are closed recursively.

EXPRESS specification:

```
*)
ENTITY close_set
SUBTYPE OF(modelling_statement);
      set_name: string_expression;
WHERE
      WR1: QUERY(elt <* used_variables(SELF.set_name) |
           NOT(elt IN SELF\simple_statement.referenced_by
           .item_of.defines.declaration)) = [];
END_ENTITY; -- close_set
(*
```

Attribute definitions:

set_name: the name of the last open set to be closed.

Formal propositions:

WR1: all the **variable**s involved in the **string_expression** shall be **accessible_variable_for_method** belonging to the context of the method (i.e., the **declaration** set of the **method_body**).

Informal propositions:

IP1: the **set_name** shall be one of the sets opened by the current method and still not closed.

14.4.12 Set_2d_relative_view_level

The **set_2d_relative_view_level** entity declares that the virtual altitude where the next entities are to be created inside the OVC shall be changed relatively to the current virtual altitude. This entity is only allowed when the current open view is founded in a geometric representation context and when it is two-dimensional. If this function is supported by the representation transmission interface this allows virtually hidden lines to be removed.

EXAMPLE In a method that produces a "bolt + nut" geometric 2D view by triggering method on bolt and nut, if each subobject view consists of opaque **fill_area** and if the virtual altitude of the nut view is greater than the virtual altitude of the bolt view, the hidden lines of the bolt may be automatically removed.

NOTE 1 Change in relative view level are local to the view created by a method. When a method returns upon completion, the initial value of this relative view level shall be reset by the system.

NOTE 2 View exchange protocols shall specify whether the hidden line elimination process is allowed.

EXPRESS specification:

```
*)
ENTITY set_2d_relative_view_level
```



```
SUBTYPE OF(modelling_statement);
     offset: numeric_expression;
WHERE
     WR1: QUERY(elt <* used_variables(SELF.offset) |
          NOT('ISO13584_METHOD_SCHEMA.CONTROL_ALLOWED_VARIABLE'
          IN TYPEOF(elt))) = [];
     WR2: QUERY(elt <* used_variables(SELF.offset) |
          NOT(elt IN SELF\simple_statement.referenced_by
          .item_of.defines.declaration)) = [];
END_ENTITY; -- set_2d_relative_view_level
(*
```

Attribute definitions:

offset: the real algebraic value to be added to the current virtual altitude.

Formal propositions:

WR1: all the variables involved in the **real_numeric_expression** shall be **control_allowed_variables**.

WR2: all the **control_allowed_variable**s involved in the **real_numeric_expression** shall belong to the context of the method (i.e., the **declaration** set of the **method_body**).

14.4.13 Predefined_representation_call_statement

A functional view is a subtype of an ISO 10303-43 **representation**. Therefore, it inherits an **items** attribute that is a set of **representation_items**.

The **items** attribute of a functional view may not be referenced explicitly (it has no **property_BSU**). It may only be filled from a method by referencing some predefined descriptions that describe the **representation_item**s that constitute its set of representation elements. This predefined description may be either provided as an external file, in which case the description shall conform to the specification of the **representation_interface** of the **method**, or it may be provided as an ISO 10303-43 **representation** that is included in the ISO 13584 library file exchange context. In the latter, the referenced view exchange protocol shall specify the used ISO 10303 Application Protocol.

NOTE The view exchange protocol associated with a functional model class is specified in the **referenced_view_exchange_protocol** attribute of a **model_class_extension**.

When a view is created by a method that was triggered by another method, this view is mapped within the **representation_context** of the embedding view when the embedded method returns.

The **predefined_representation_call_statement** entity enables the specification of the predefined description that shall be processed to fill the **items** attribute of the current open view. An external description may be either a **program_reference**, that has some formal parameters, or a **representation_reference**, that has no formal parameter.

EXAMPLE 1 The geometry of a bolt may be described through an external ISO 10303-203 explicit data model. It is a **representation_reference** predefined description. It shall be processed through an ISO 10303-203 post processor. In the functional model class that provides the geometric representation of the bolt family, each referenced **representation_reference** describes a geometric **representation** of one bolt of the family.

EXAMPLE 2 The geometry of all the bolts of a bolt **component_class** may be described by a parametric program according to the specifications of the ISO 13584-31 Geometric Programming Interface. It is a **program_reference** predefined description. It contains some formal parameters that may be the length and the diameter of the bolt. It shall be processed through the ISO 13584 Part 31 Geometric Programming Interface.

EXPRESS specification:

```
*)
ENTITY predefined_representation_call_statement
ABSTRACT SUPERTYPE OF(ONEOF(
      send_representation_statement,
      send_representation_reference_statement,
      call_program_statement))
SUBTYPE OF(simple_statement);
END_ENTITY; -- predefined_representation_call_statement
(*
```

14.4.14 Send_representation_statement

A **send_representation_statement** entity specifies the ISO 10303-43 **representation** whose **items** attribute describes the **representation_item**s that shall be mapped within the **representation_context** of the current open view. These **representation_item**s are mapped onto the current OVC using the current transformation, as defined by previous **modelling_statement**s, within the current open set and, if appropriate, at the current virtual altitude.

NOTE 1 ISO 13584-24 does not specify whether the result of this mapping is an ISO 10303-43 **mapped_item**, or it is a set of **representation_item**s defined by applying the transformation to each item.

NOTE 2 Reference to the **representation** to be mapped is defined as a **functional_domain_restriction** that may evaluates to various **representation**s according to the values of **library_variable**s belonging to the context of the method.

NOTE 3 When the view is created by an explicit **representation**, different **representation**s are, in general, associated with the different values of the **required_properties**.

EXPRESS specification:

```
*)
ENTITY send_representation_statement
SUBTYPE OF(predefined_representation_call_statement);
      corresponding_method_variable: assignment_allowed_variable;
      representation_to_be_processed: functional_domain_restriction;
WHERE
      WR1: SELF.representation_to_be_processed.defines
           = [SELF.corresponding_method_variable\generic_variable.
           interpretation.semantics];
      WR2: SELF.corresponding_method_variable
           IN SELF.referenced_by.item_of.defines.declaration;
      WR3: collects_variables(SELF.representation_to_be_processed
           .assumes) <= SELF.referenced_by.item_of.defines.declaration;
      WR4: definition_available_implies(
           SELF.corresponding_method_variable\generic_variable
           .interpretation.semantics\property_semantics.the_property,
           (('ISO13584_IEC61360_DICTIONARY_SCHEMA.ENTITY_INSTANCE_TYPE'
           IN data_type_typeof(SELF.corresponding_method_variable
           \generic_variable.interpretation.semantics
           \property_semantics.the_property))
           AND ('REPRESENTATION_SCHEMA.REPRESENTATION' IN
           data_type_type_name(
```



```
            SELF.corresponding_method_variable\generic_variable
            .interpretation.semantics\property_semantics.the_property)))
            OR (data_type_typeof(SELF.corresponding_method_variable
            \generic_variable.interpretation.semantics\property_semantics
            .the_property) = []));
       WR5: NOT all_class_descriptions_reachable
            (SELF.referenced_by.item_of.defines.describes.its_class
            .dictionary_definition) OR
            ((SELF.corresponding_method_variable\generic_variable
            .interpretation.semantics\property_semantics.the_property) IN
            method_variables(SELF.referenced_by.item_of.defines
            .describes.its_class.dictionary_definition));
       WR6: applicable_tables(SELF.referenced_by.item_of
            .defines.describes.its_class.dictionary_definition,
            SELF.representation_to_be_processed.base_tables);
       WR7: used_variables_in_domain(
            SELF.representation_to_be_processed) <=
            SELF.referenced_by.item_of.defines.declaration;
  END_ENTITY; -- send_representation_statement
  (*
```

Attribute definitions:

corresponding_method_variable: the property variable declared in the context of the **method** that is assigned the entity name of the **representation** to be processed.

representation_to_be_processed: the **functional_domain_restriction** that specifies the entity name of the **representation** to be processed.

NOTE 4 When the **simple_domain** is a **null_defined_value**, no value shall be assigned to the **corresponding_method_variable** and no **representation** shall be processed.

Formal propositions:

WR1: the **functional_domain_restriction** shall define the **corresponding_method_variable**.

WR2: the **corresponding_method_variable** shall be part of the context of the **method**.

WR3: the variables associated with the **assumes** attribute of the **functional_domain_restriction** shall be part of the context of the **method**.

WR4: if data are available, then **IP1** holds.

WR5: the **corresponding_method_variable** shall be defined in the **model_class_extension** as **method_variables**.

WR6: all the tables possibly involved in the **functional_domain_restriction** shall be applicable tables for the class to which the method belongs.

WR7: the variables used in the **functional_domain_restriction** shall be part of the context of the **method**.

Informal propositions:

IP1: the data type of the **property_semantics** associated with the **corresponding_method_variable** shall be an **entity_instance_type** that shall contain a **representation**.

14.4.15 Send_representation_reference_statement

The **send_representation_reference_statement** entity specifies the **class_extension_external_item** that contains an ISO 10303-43 **representation** whose **items** attribute describes the **representation_items** that shall be processed through the **representation_interface** of the **method** and that shall be mapped within the **representation_context** of the open view. These **representation_items** are mapped onto the current OVC using the current transformation, as defined by previous **modelling_statement**s, within the current open set and, if appropriate, at the current virtual altitude. The **representation_reference** to be processed is specified like a **representation** in the **send_representation_statement**.

NOTE 1 The only difference between **representation**s and **representation_reference**s is that the former is represented within the library delivery file, when the latter is represented in a separate file that is a library external file. The view exchange protocols that use ISO 10303 conformant **representation**s to define the **items** representations within library functional model classes shall specify whether these **representation**s shall be internal or external to the library delivery file.

EXPRESS specification:

```
*)
ENTITY send_representation_reference_statement
SUBTYPE OF(predefined_representation_call_statement);
     corresponding_method_variable: assignment_allowed_variable;
     representation_reference_to_be_processed:
          functional_domain_restriction;
WHERE
     WR1: SELF.representation_reference_to_be_processed.defines
          = [SELF.corresponding_method_variable\generic_variable
          .interpretation.semantics];
     WR2: SELF.corresponding_method_variable
          IN SELF.referenced_by.item_of.defines.declaration;
     WR3: collects_variables(
          SELF.representation_reference_to_be_processed.assumes)
          <= SELF.referenced_by.item_of.defines.declaration;
     WR4: definition_available_implies(
          SELF.corresponding_method_variable\generic_variable
          .interpretation.semantics\property_semantics.the_property,
          (('ISO13584_IEC61360_DICTIONARY_SCHEMA.ENTITY_INSTANCE_TYPE'
          IN data_type_typeof(SELF.corresponding_method_variable
          \generic_variable.interpretation.semantics
          \property_semantics.the_property))
          AND ('ISO13584_EXTERNAL_FILE_SCHEMA.REPRESENTATION_REFERENCE'
          IN data_type_type_name(SELF.corresponding_method_variable
          \generic_variable.interpretation.semantics
          \property_semantics.the_property))) OR
          (data_type_typeof(SELF.corresponding_method_variable
          \generic_variable.interpretation.semantics
          \property_semantics.the_property) = []));
```



```
        WR5: NOT all_class_descriptions_reachable
             (SELF.referenced_by.item_of.defines.describes.its_class
             .dictionary_definition) OR
             ((SELF.corresponding_method_variable\generic_variable
             .interpretation.semantics\property_semantics.the_property) IN
             method_variables(SELF.referenced_by.item_of.defines
             .describes.its_class.dictionary_definition));
        WR6: applicable_tables(SELF.referenced_by.item_of
             .defines.describes.its_class.dictionary_definition,
             SELF.representation_reference_to_be_processed.base_tables);
        WR7: used_variables_in_domain(
             SELF.representation_reference_to_be_processed) <=
             SELF.referenced_by.item_of.defines.declaration;
    END_ENTITY; -- send_representation_reference_statement
    (*
```

Attribute definitions:

corresponding_method_variable: the property declared in the context of the **method** that is assigned the entity name of the **representation_reference** to be processed.

representation_reference_to_be_processed: the **functional_domain_restriction** that specifies the entity name of the **representation_reference** to be processed.

NOTE 2 When the **simple_domain** is a **null_defined_value** no value shall be assigned to the **corresponding_method_variable** and no **representation_reference** shall be processed.

Formal propositions:

WR1: the **functional_domain_restriction** shall define the **corresponding_method_variable**.

WR2: the **corresponding_method_variable** shall be part of the context of the **method**.

WR3: the variables associated with the **assumes** attribute of the **functional_domain_restriction** shall be part of the context of the **method**.

WR4: if data are available, then **IP1** holds.

WR5: the **corresponding_method_variable** shall be defined in the **model_class_extension** as **method_variables**.

WR6: all the tables possibly involved in the **functional_domain_restriction** shall be applicable tables for the class to which the method belongs.

WR7: the variables used in the **functional_domain_restriction** shall be part of the context of the **method**.

Informal propositions:

IP1: the data type of the **property_semantics** associated with the **corresponding_method_variable** shall be an **entity_instance_type** that shall contain a **representation_reference**.

IP2: the **representation_reference** to which the **functional_domain_restriction** evaluates shall reference, in its **used_protocol** attribute, the **representation_interface** of the **method** to which the statement belongs.

14.4.16 Call_program_statement

A **call_program_statement** specifies the **class_extension_external_item** that contains the external program that shall be processed through the **representation_interface** of the **method** to generate the **representation_items** that shall be mapped within the **representation_context** of the current open view.

NOTE 1 In this context, program means any entity associated with input parameters and that shall be triggered by some means.

EXAMPLE 1 Such a program may be a parametric data model, a VHDL specification or a FORTRAN program that references the ISO 13584-31 interface.

These **representation_items** are mapped onto the current *open_view* using the current transformation, as defined by previous **modelling_statement**s, within the current open set and, if appropriate, at the current virtual altitude.

NOTE 2 This International Standard does not specify whether the result of this mapping is an ISO 10303-43 **mapped_item**, or a set of **representation_item**s defined by applying the transformation to each item.

The reference to the file to be processed is specified as a **representation_reference** in a **send_representation_reference_statement**.

The referenced program has input parameters whose actual values are specified through **library_expressions**. It may have output and/or inout parameters whose corresponding actual parameters are specified as **assignment_allowed_variable**s. Output or inout parameters shall be associated, through an environment, either to **self_property_semantics** that are method variables or to **open_view_property_semantics**.

When a view exchange protocol enables the use of **program_reference**s, it shall specify the types of the allowed input, output and inout parameters, and the mapping between the parameter representation in the host language of the program and the library value representation as specified by the **ISO13584_instance_resource_schema**.

EXAMPLE 2 If a view exchange protocol enables the use of JAVA programs in a functional model, and if this view exchange protocol enables a **position** as output parameter, a mapping shall be specified between a **position** entity instance and a structured data type in JAVA.

NOTE 3 **position** is defined in ISO 10303-42: 1994.

EXPRESS specification:

```
*)
ENTITY call_program_statement
SUBTYPE OF(predefined_representation_call_statement);
      corresponding_method_variable: assignment_allowed_variable;
      program_reference_to_be_processed:
           functional_domain_restriction;
      input_parameters: LIST [0:?] OF library_expression;
      output_parameters: LIST [0:?] OF assignment_allowed_variable;
      inout_parameters: LIST [0:?] OF assignment_allowed_variable;
WHERE
      WR1: SELF.program_reference_to_be_processed.defines
           = [SELF.corresponding_method_variable\generic_variable
           .interpretation.semantics];
      WR2: SELF.corresponding_method_variable
```



```
          IN SELF.referenced_by.item_of.defines.declaration;
     WR3: collects_variables(
          SELF.program_reference_to_be_processed.assumes)
          <= SELF.referenced_by.item_of.defines.declaration;
     WR4: definition_available_implies(
          SELF.corresponding_method_variable\generic_variable
          .interpretation.semantics\property_semantics.the_property,
          (('ISO13584_IEC61360_DICTIONARY_SCHEMA.ENTITY_INSTANCE_TYPE'
          IN data_type_typeof(SELF.corresponding_method_variable
          \generic_variable.interpretation.semantics
          \property_semantics.the_property))
          AND ('ISO13584_EXTERNAL_FILE_SCHEMA.PROGRAM_REFERENCE' IN
          data_type_type_name(SELF.corresponding_method_variable
          \generic_variable.interpretation.semantics
          \property_semantics.the_property)))
          OR (data_type_typeof(SELF.corresponding_method_variable
          \generic_variable.interpretation.semantics
          \property_semantics.the_property) = []));
     WR5: NOT all_class_descriptions_reachable
          (SELF.referenced_by.item_of.defines.describes.its_class
          .dictionary_definition) OR
          ((SELF.corresponding_method_variable\generic_variable
          .interpretation.semantics\property_semantics.the_property) IN
          method_variables(SELF.referenced_by.item_of.defines
          .describes.its_class.dictionary_definition));
     WR6: applicable_tables(SELF.referenced_by.item_of
          .defines.describes.its_class.dictionary_definition,
          SELF.program_reference_to_be_processed.base_tables);
     WR7: QUERY(expr <* SELF.input_parameters
          | QUERY(v <* used_variables(expr)
          | NOT(v IN SELF\simple_statement.referenced_by
          .item_of.defines.declaration)) <> []) = [];
     WR8: QUERY(v <* SELF.output_parameters
          | NOT(v IN SELF\simple_statement.referenced_by
          .item_of.defines.declaration)) = [];
     WR9: QUERY(v <* SELF.inout_parameters
          | NOT(v IN SELF\simple_statement.referenced_by
          .item_of.defines.declaration)) = [];
     WR10: used_variables_in_domain(
          SELF.program_reference_to_be_processed)
          <= SELF.referenced_by.item_of.defines.declaration;
END_ENTITY; -- call_program_statement
(*
```

Attribute definitions:

corresponding_method_variable: the property declared in the context of the **method** that is assigned the entity name of the **program_reference** to be processed.

program_reference_to_be_processed: the **functional_domain_restriction** that specifies the entity name of the **program_reference** to be processed.

NOTE When the **simple_domain** is a **null_defined_value** no value shall be assigned to the **corresponding_method_variable** and no **program_reference** shall be processed.

input_parameters: list of the **expressions** to be provided to the **program** as input parameters.

output_parameters: list of the **assignment_allowed_variable**s that are identified to the output parameters of the **program**.

inout_parameters: list of the **assignment_allowed_variable**s that are identified to the inout parameters of the **program**.

Formal propositions:

WR1: the **functional_domain_restriction** shall define the **corresponding_method_variable**.

WR2: the **corresponding_method_variable** shall be part of the context of the **method**.

WR3: the variables associated with the **assumes** attribute of the **functional_domain_restriction** shall be part of the context of the **method**.

WR4: if data are available, then **IP1** holds.

WR5: the **corresponding_method_variable** shall be defined in the **functional_model_class_extension** as **method_variables**.

WR6: all the tables possibly involved in the **functional_domain_restriction** shall be applicable tables for the class to which the method belongs.

WR7: the variables used in the **input_parameters** LIST of expressions shall belong to the context of the method.

WR8: the **output_parameters** variables shall belong to the context of the method.

WR9: the **inout_parameters** variables shall belong to the context of the method.

WR10: the variables used in the **functional_domain_restriction** shall be part of the context of the **method**.

Informal propositions:

IP1: the data type of the **property_semantics** associated with the **corresponding_method_variable** shall be an **entity_instance_type** that shall contain a **program_reference**.

IP2: the **representation_reference** to which the **functional_domain_restriction** evaluates shall reference, in its **used_protocol** attribute, the **representation_interface** of the **method** to which the statement belongs.

IP3: the number and types of the **input_parameters**, **output_parameters** and **inout_parameters** shall be compatible with the number and types defined by the **in_parameters**, **out_parameters** and **inout_parameters** of the **program_reference** to which the **functional_domain_restriction** evaluates.

14.4.17 Assignment_statement

The **assignment_statement** entity enables the specification of what value shall be assigned to an **assignment_allowed_variable**.

NOTE 1 This entity allows the specification of what value shall be assigned either to a **self_property_value_semantics** or to an **open_view_property_value_semantics**.

NOTE 2 Only the view properties that are identified by a **property_BSU**s may be represented as **open_view_property_value_semantics** and, therefore, may be assigned a value using an **assignment_statement**.

NOTE 3 The **items** attribute is a view properties inherited from the ISO 10303 **representation** of which the view is subtype. This view properties is not identified by any **property_BSU.** It may only be assigned a value implicitly using a **predefined_representation_call_statement**.

NOTE 4 When a method is run, the view control variables of the current open view shall be set by the LMS to the values they have in the message that triggered the method. These variables shall not be assigned different values by the method body.

EXPRESS specification:

```
*)
ENTITY assignment_statement
SUBTYPE OF(simple_statement);
      assigned_variable: assignment_allowed_variable;
      assigned_value: library_expression;
WHERE
      WR1: SELF.assigned_variable IN
           SELF\simple_statement.referenced_by.item_of.defines
           .declaration;
      WR2: compatible_variable_and_library_expression(
           SELF.assigned_variable, SELF.assigned_value);
      WR3: QUERY(v <* used_variables(
           SELF\assignment_statement.assigned_value)
           | NOT(v IN SELF\simple_statement.referenced_by
           .item_of.defines.declaration)) = [];
      WR4: NOT(SELF.assigned_variable\generic_variable
           .interpretation.semantics\property_semantics.the_property
           IN provided_properties_list(
           SELF\simple_statement.referenced_by.item_of.defines
           .describes.its_class.dictionary_definition));
END_ENTITY; -- assignment_statement
(*
```

Attribute definitions:

assigned_variable: the **assignment_allowed_variable** to which a value is assigned.

assigned_value: the **library_expression** the value of which is assigned to the **library_variable**.

Formal propositions:

WR1: **the assigned_variable** shall belong to the context of the **method**.

WR2: the data types of the **assigned_value library_expression**s shall be compatible with the **library_variable**s to which they are assigned.

WR3: all the **variable**s involved in the **assigned_value** expression shall belong to the context of the method.

WR4: **the assigned_variable** shall not be a selectable, required or derived property of the functional model class.

14.4.18 Sub_object_view_statement

The **sub_object_view_statement** allows the triggering of a **method** that refers, directly or indirectly, to another class instance, the subobject. This instance may be either an already existing **item_class** instance, that is a component of an assembled item of which the current functional model class instance is able to create a view, or a created **functional_model_class** instance that is specified by a **class_instance_constructor**.

In the first case, the already existing instance is referenced by a **self_property_value_semantics** of the functional model class instance that is imported from the assembled item the functional model is-view-of. It shall be an instance of an **item_class**, with or without **item_class_extension**. In this case, the LMS shall search, following the is-view-of lattice represented in the user library, a functional model class able to provide the view specified by the **created_view** attribute for the subobject instance.

In the second case, the created instance shall be an instance of a **functional_model_class_extension** that is able to create some part of the view created by its triggering instance. Due to the **complete_identification_for_instance_rule** rule, the functional model class instance is completely and correctly identified by the **class_instance_constructor**. In this case the **functional_model_class_extension** instance shall be able to provide the view specified by the **created_view** with the view control variables specified by the **v_c_v_values** attributes.

NOTE The **class_instance_constructor** is defined in the **ISO13584_library_expression_schema** and the **complete_identification_for_instance_rule** is defined in the **ISO13584_library_content_schema**. They are both referenced from the **ISO13584_library_content_schema** to ensure that the **complete_identification_for_instance_rule** holds for all the instances created by a method.

EXPRESS specification:

```
*)
ENTITY sub_object_view_statement
ABSTRACT SUPERTYPE OF(ONEOF(referenced_sub_item_view_statement,
     constructed_sub_model_view_statement))
SUBTYPE OF(simple_statement);
     created_view: class_BSU;
     v_c_v_values: SET [0:?] OF property_assignment;
WHERE
     WR1: definition_available_implies(SELF.created_view,
          'ISO13584_EXTENDED_DICTIONARY_SCHEMA.FUNCTIONAL_VIEW_CLASS'
          IN TYPEOF(SELF.created_view.definition[1]));
     WR2: QUERY(e <* collects_referenced_library_expressions(
          SELF.v_c_v_values) | QUERY(v <* used_variables(e)
          | NOT(v IN SELF\simple_statement.referenced_by
          .item_of.defines.declaration)) <> []) = [];
END_ENTITY; -- sub_object_view_statement
(*
```

Attribute definitions:

created_view: the **class_BSU** that defines the view to be created by the triggered **method**.

v_c_v_values: the **property_assignment**s that define the values of the view control variables that specify the view.

Formal propositions:

WR1: if data are available, then **IP1** holds.

WR2: all the variables used in **v_c_v_values** shall belong to the context of the method.

Informal propositions:

IP1: the **created_view** shall correspond to a functional view.

IP2: all the **library_expression**s in **v_c_v_values** shall evaluate to values belonging to the range of the view control variables.

14.4.19 Referenced_sub_item_view_statement

The **referenced_sub_item_view_statement** allows the triggering of a **method** that creates a specified functional view of a component, the referenced subitem, of an assembled item. A functional model class, that depends on the content of the user library, is first instantiated, and then, a **method** of that instance is triggered that creates the specified functional view.

The functional model class to be instantiated is defined by:

— the **sub_object variable_semantics** attribute whose value is a **dic_class_instance** that identifies a pre-existing instance of an **item_class** (the component of the assembled item), of which a functional view is required, and

— the required functional view, specified by the **created_view** and **v_c_v_values** inherited attributes.

NOTE The identity of the component of the assembled item may be for instance imported from the assembled item by the **required_item_characteristics** attributes of the functional model class instance that contains the triggering method.

The LMS shall search, following the is-view-of lattice represented in the user library, a **functional_model_class_extension** able to provide the required view for the **sub_object** instance. Then this class is instantiated, its possible **required_item_characteristics** are initiated from the **sub_object** properties and the method is triggered. The selected **functional_model_class_extension** shall not contain any **free_model_properties**.

EXPRESS specification:

```
*)
ENTITY referenced_sub_item_view_statement
SUBTYPE OF(sub_object_view_statement);
      sub_object: self_property_value_semantics;
WHERE
      WR1: definition_available_implies(SELF.sub_object
           \property_semantics.the_property,
           (data_type_typeof(SELF.sub_object
           \property_semantics.the_property) = [])
           OR ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_INSTANCE_TYPE'
           IN data_type_typeof(SELF.sub_object
```



```
          \property_semantics.the_property)));
      WR2: self_property_value_semantics_is_item_class(SELF);
      WR3: definition_available_implies(SELF.sub_object
          \property_semantics.the_property,
          (data_type_class_of(SELF.sub_object
          \property_semantics.the_property) = []))
          OR definition_available_implies(data_type_class_of(
          SELF.sub_object\property_semantics.the_property)[1],
          data_type_class_of(
          SELF.sub_object\property_semantics.the_property)[1] IN
          SELF\simple_statement.referenced_by.item_of.defines
          .describes.specification.referred_subobject_models);
END_ENTITY; -- referenced_sub_item_view_statement
(*
```

Attribute definitions:

sub_object: the property of the SELF instance that references the subitem.

Formal propositions:

WR1: if data are available, then **IP1** holds.

WR2: the class of the value of the **self_property_value_semantics** shall be an **item_class**.

WR3: if data are available, then **IP2** holds.

Informal propositions:

IP1: the value of the **self_property_value_semantics** shall correspond to a class instance.

IP2: the **SELF.sub_item.the_property\dic_class_instance.class_def** class shall belong to the **referred_subobject_models** of the method.

IP3: there shall exist, on the receiving system, a **functional_model_class_extension** able to create the required view for the **sub_object** instance.

14.4.20 Constructed_sub_model_view_statement

The **constructed_sub_model_view_statement** allows the triggering of a **method** belonging to a created instance of a **functional_model_class_extension** that is specified by a **class_instance_constructor**. Due to the **complete_identification_for_instance_rule** rule, the instance is completely and correctly identified by the **class_instance_constructor**.

The **functional_model_class_extension** shall be able to provide the view specified by the inherited **created_view** and **v_c_v_values** attributes.

EXPRESS specification:

```
*)
ENTITY constructed_sub_model_view_statement
SUBTYPE OF(sub_object_view_statement);
```



```
      sub_model: class_instance_constructor;
WHERE
      WR1: SELF.sub_model\class_instance_expression.expr_type IN
           SELF\simple_statement.referenced_by.item_of.defines
           .describes.specification.referred_subobject_models;
      WR2: definition_available_implies(
           SELF.sub_model\class_instance_expression.expr_type,
           'ISO13584_EXTENDED_DICTIONARY_SCHEMA'
           +'.FUNCTIONAL_MODEL_CLASS' IN
           TYPEOF(SELF.sub_model\class_instance_expression.expr_type
           .definition[1]));
      WR3: definition_available_implies(
           SELF.sub_model\class_instance_expression.expr_type,
           SIZEOF(SELF.sub_model\class_instance_expression.expr_type
           .referenced_by) = 1);
      WR4: definition_available_implies(
           SELF.sub_model\class_instance_expression.expr_type,
           SELF.sub_model\class_instance_expression.expr_type
           = SELF\sub_object_view_statement.created_view);
      WR5: QUERY(v <* used_variables(SELF.sub_model)
           | NOT(v IN SELF\simple_statement.referenced_by
           .item_of.defines.declaration)) = [];
END_ENTITY; -- constructed_sub_model_view_statement
(*
```

Attribute definitions:

sub_model: the **class_instance_constructor** that evaluates to the functional model instance subobject.

Formal propositions:

WR1: the **SELF.sub_model\class_expression.expression_type** class shall belong to the **referred_subobject_models** of the method.

WR2: if data are available, then **IP1** holds.

WR3: if data are available, then **IP2** holds.

WR4: if data are available, then **IP3** holds.

WR5: all the variables used in **SELF.sub_model class_instance_constructor** shall belong to the context of the method.

Informal propositions:

IP1: the **SELF.sub_model\class_expression.expression_type** class shall be a **functional_model_class**.

IP2: the **SELF.sub_model\class_expression.expression_type functional_model_class** shall be associated with a content (i.e., a **functional_model_class_extension**).

IP3: the **SELF.sub_model** shall be able to create the **SELF**.**created_view**.

IP4: the **SELF.sub_model** shall be able to create the functional view that corresponds to the evaluated values of the view control variables.

14.5 ISO13584_method_schema rules definitions

14.5.1 Created_view_v_c_v_rule rule

The **created_view_v_c_v_rule** rule checks that for each **method_specif**, all the properties referenced by its **v_c_v_range** attribute are view control variables of the functional view class referenced by its **created_view** attribute.

NOTE This constraint may only be checked when functional view classes are defined in the same exchange context.

EXPRESS specification:

```
*)
RULE created_view_v_c_v_rule FOR(method_specif,
     functional_view_class);
WHERE
     WR1: QUERY(meth <* method_specif |
          all_class_descriptions_reachable(meth.created_view)
          AND (QUERY(temp <* meth.v_c_v_range | NOT(
          temp.parameter_type IN functional_view_v_c_v(
          meth.created_view))) <> [])) = [];
END_RULE; -- created_view_v_c_v_rule
(*
```

Formal propositions:

WR1: all the properties referenced by the **v_c_v_range** attribute of a **method_specif** shall be view control variables of the functional view class referenced by the **created_view** attribute of this **method_specif**.

14.5.2 V_c_v_values_set_and_created_view_v_c_v_set_equality_rule rule

The **v_c_v_values_set_and_created_view_v_c_v_set_equality_rule** rule checks that for each **sub_object_view_statement**, all the properties referenced by its **v_c_v_values** attribute are view control variables of the functional view class referenced by its **created_view** attribute.

NOTE 1 This constraint may only be checked when functional view classes are defined in the same exchange context.

NOTE 2 If some view control variables of the **created_view** functional view are not needed to specify the view, they shall be associated with an indeterminate (?) **library_expression**.

EXPRESS specification:

```
*)
RULE v_c_v_values_set_and_created_view_v_c_v_set_equality_rule FOR(
     sub_object_view_statement, functional_view_class);
WHERE
     WR1: QUERY(sub <* sub_object_view_statement |
          all_class_descriptions_reachable(sub.created_view)
```



```
          AND
          (collects_assigned_properties(sub.v_c_v_values) <>
          functional_view_v_c_v(sub.created_view))) = [];
END_RULE; -- v_c_v_values_set_and_created_view_v_c_v_set_equality_rule
(*
```

Formal propositions:

WR1: for each **sub_object_view_statement**, all the properties referenced by its **v_c_v_values** attribute shall be view control variables of the functional view class referenced by its **created_view** attribute.

14.5.3 No_v_c_v_in_assigned_variables_set_rule rule

The **no_v_c_v_in_assigned_variables_set_rule** rule checks that for each **assignment_statement**, the **assigned_variable** is not a view control variable of the current open view.

NOTE This constraint may only be checked when functional view classes are defined in the same exchange context.

EXPRESS specification:

```
*)
RULE no_v_c_v_in_assigned_variables_set_rule FOR(
     assignment_statement, functional_view_class);
WHERE
     WR1: QUERY(ass <* assignment_statement |
          all_class_descriptions_reachable(ass\simple_statement.
          referenced_by.item_of.defines.describes
          .specification.created_view)
          AND
          (ass.assigned_variable\generic_variable
          .interpretation.semantics\property_semantics.the_property IN
          functional_view_v_c_v(ass\simple_statement.referenced_by
          .item_of.defines.describes.specification.created_view)))
          = [];
END_RULE; -- no_v_c_v_in_assigned_variables_set_rule
(*
```

Formal propositions:

WR1: for each **assignment_statement**, the **assigned_variable** shall not be a view control variable of the current open view.

14.6 ISO13584_method_schema function definitions

This section describes the functions in the **ISO13584_method_schema**.

14.6.1 Checks_classes_in_path function

The **checks_classes_in_path** function checks that all the **class_BSU**s that define the final domain of the properties referred to in a **property_semantics** path belong to an AGGREGATE of **class_BSU**s defined by the **cl** parameter.

The function returns a LOGICAL that is TRUE when all the properties of the path whose domain is a class have their domain defined by **class_BSU**s belonging to **cl**, and FALSE when they have not. It returns UNKNOWN when at least one **dictionary_element** is not available.

EXPRESS specification:

```
*)
FUNCTION checks_classes_in_path(v: property_semantics_or_path;
          cl: SET OF class_BSU): LOGICAL;

LOCAL
     prop: property_BSU;
     temp: SET [0:1] OF class_BSU;
END_LOCAL;

prop := v.the_property;

IF data_type_typeof(prop) = []
THEN -- domain is unknown
     RETURN(UNKNOWN);
ELSE -- domain is known
     IF('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_INSTANCE_TYPE' IN
          data_type_typeof(prop))
     THEN
          temp := data_type_class_of(prop);
          IF NOT(temp[1] IN cl)
          THEN
               RETURN(FALSE);
          ELSE -- domain in cl, check forward
               IF EXISTS(v.its_own_property)
               THEN
                    RETURN(checks_classes_in_path(
                         v.its_own_property, cl));
               ELSE
                    RETURN(TRUE); -- all path checked
               END_IF;
          END_IF;
     ELSE
          RETURN(TRUE); -- domain is not a class
     END_IF;
END_IF;

END_FUNCTION; -- checks_classes_in_path
(*
```

14.6.2 Checks_applicable_properties_in_path function

The **checks_applicable_properties_in_path** function checks that all the **its_own_property** properties used in the path of a **property_semantics_or_path** are **applicable_properties** for the class to which they belong.

The function returns a LOGICAL that is TRUE when all the properties of the path are **applicable_properties** and FALSE when they are not. It returns UNKNOWN when at least one **dictionary_element** is not available.

EXPRESS specification:

```
*)
FUNCTION checks_applicable_properties_in_path(
     v: property_semantics_or_path): LOGICAL;

LOCAL
     prop: property_BSU;
     temp: SET[0:1] OF class_BSU;
END_LOCAL;

prop := v.the_property;

IF data_type_typeof(prop) = []
THEN --domain is unknown
     RETURN(UNKNOWN);
ELSE --domain is known
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_INSTANCE_TYPE' IN
          data_type_typeof(prop))
     THEN
          IF EXISTS(v.its_own_property)
          THEN
               temp := data_type_class_of(prop);
               RETURN(applicable_properties(
                    temp[1], [v.its_own_property
                    .the_property]) AND
                    checks_applicable_properties_in_path(
                    v.its_own_property));
          ELSE
               RETURN(TRUE); --all paths checked
          END_IF;
     ELSE
          RETURN(TRUE); -- no sub_property
     END_IF;
END_IF;

END_FUNCTION; --checks_applicable_properties_in_path
(*
```

14.6.3 same_view_model_method

The **same_view_model_method** function checks that the **cl** class defined by its **class_extension** does not refers to a **class** associated with a **class_extension** as its superclass.

If the **class** associated with **cl** cannot be computed, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION same_view_model_method(meth: method_specif): LOGICAL;

IF SIZEOF(meth.specifies.its_class
     \content_item.dictionary_definition.definition) = 0
THEN
     RETURN(UNKNOWN);
END_IF;

RETURN(
     meth.specifies.its_class
     \content_item.dictionary_definition.definition[1]
     \abstract_functional_model_class.created_view :=:
     meth.created_view);

END_FUNCTION; -- same_view_model_method
(*
```

14.6.4 self_property_value_semantics_is_item_class

The **self_property_value_semantics_is_item_class** function checks that the class of the value of the **sub_object self_property_value_semantics** of an **it referenced_sub_item_view_statement** is an **item_class**.

If the **class** associated with the **sub_object self_property_value_semantics** of the **it referenced_sub_item_view_statement** cannot be computed, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION self_property_value_semantics_is_item_class(
     it: referenced_sub_item_view_statement): LOGICAL;

IF SIZEOF(it.sub_object
     \property_semantics.the_property.definition) = 0
THEN
     RETURN(UNKNOWN);
END_IF;

IF (data_type_class_of(it.sub_object
     \property_semantics.the_property) = [])
THEN
     RETURN(UNKNOWN);
END_IF;

IF SIZEOF(data_type_class_of(it.sub_object
     \property_semantics.the_property)[1].definition) = 0
THEN
     RETURN(UNKNOWN);
END_IF;
```



```
RETURN(('ISO13584_IEC61360_DICTIONARY_SCHEMA.ITEM_CLASS' IN
     TYPEOF(data_type_class_of(it.sub_object
     \property_semantics.the_property)[1].definition[1])));

END_FUNCTION; -- self_property_value_semantics_is_item_class
(*
*)
END_SCHEMA; -- ISO13584_method_schema
(*
```

15 Conformance requirements

An integrated library may consist of general model classes, functional model classes and functional view classes.

This part of ISO 13584 does not provide for the simultaneous exchange of these three kinds of classes.

NOTE 1 Such an exchange is allowed by library integrated information model 25 specified in ISO 13584-25:—[1].

Three library integrated information models are defined in this part of ISO 13584. They provide respectively for the:

— exchange of general model classes, defining the first library integrated information model (LIIM 24-1) and whose EXPRESS schema is the **ISO13584_g_m_iim_schema**;

— exchange of functional model classes, defining the second library integrated information model (LIIM 24-2) and whose EXPRESS schema is the **ISO13584_f_m_iim_schema**;

— exchange of functional view classes, defining the third library integrated information model (LIIM 24-3) and whose EXPRESS schema is the **ISO13584_f_v_iim_schema**.

Each schema 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 to support all entities, types, and associated constraints defined as part of that class. Support for a particular conformance class requires support of all the options specified in this class.

A conformance class may also define standard data that are instances of some entity data types defined in one schema of this part of ISO 13584. Claiming conformance to this class requires that such instances are recognised in an exchange context.

NOTE 2 Two kinds of standard data are intended to be specified as part of the conformance classes of the library integrated information models and of the view exchange protocols defined in ISO 13584: **external_file_protocols**, and **data_exchange_specification_identification**s.

A conformance class may finally define some dictionary entries, and the standard **basic_semantic_unit**s that shall be used to refer to these entries. Claiming conformance to this class requires that these dictionary entries are stored in the dictionary of the system and may be referenced by the standard **basic_semantic_unit**s.

EXAMPLE 1 ISO 13584-101 defines the *basic_geometry* functional view class, and the content of the **class_BSU** that shall be used to reference it.

To conform to ISO 13584, a system shall state the following:

[1] To be published.

— the conformance classes it supports for the library integrated information model LIIM 24-1,

— if any, the conformance classes it supports for the library integrated information model LIIM 24-2,

— if any, the conformance classes it supports for the library integrated information model LIIM 24-3,

— if any, the view exchange protocols from the view exchange protocol series of parts of ISO 13584 it supports, together with their supported conformance class,

— if any, the **external_file_protocol**s it supports that are not already specified as standard data in any of the supported conformance classes of the supported library integrated information models and view exchange protocols.

Claiming conformance to some conformance classes of the first three library integrated information models includes satisfying the requirements stated in the corresponding integrated schema, support of all the entities and associated constructs defined as part of these conformance classes, and recognition of all the standard data and dictionary entries possibly associated with these conformance classes.

Claming conformance to some conformance classes of a view exchange protocol means satisfying the requirements stated in the corresponding constraint schema(s), if any, recognition of all the standard data and dictionary entries possibly associated with these conformance classes, and, when such a standard data is an **external_file_protocol** entity instance, capability to process the library external files whose structure is defined by this **external_file_protocol** entity instance.

NOTE 3 A view echange protocol may specify how it should be referenced from a library delivery file by means of an EXPRESS schema that consists only of constraints. These constraints are fulfilled by any library delivery file that references this view echange protocol in any of its conformance class.

EXAMPLE 2 Annex C specifies how to build the complete information model of a library delivery file that reference **ISO13584_g_m_iim_schema** and a view exchange protocol "V1".

Conformance to different view exchange protocols are orthogonal. An implementation shall be able to compile a library exchange context that refers to a supported conformance class of a library integrated information model, and to several view exchange protocols, all of them in supported conformance classes.

EXAMPLE 3 ISO 13584-101 defines the dictionary entries that corresponds to the *basic_geometry* functional view class and defines the ISO 13584-31 **program_protocol**. A library exchange context may reference both the library integrated information model LIIM 24-2 and the view exchange protocols defined by ISO 13584-101. The first reference defines the structure of the library delivery file, and of the library external files that contain documents or dialogue resources. The second reference defines the structure of the library external files that contain FORTRAN programs intended to generate the *basic_geometry* representation of library parts.

Finally, claiming support of some **external_file_protocol**(s) not already specified as standard data in one of the supported view exchange protocol conformance classes means capability to process the library external files whose structure is defined by these **external_file_protocol**(s), if these library external files are referenced from supported entities.

EXAMPLE 4 An ISO 13584-conformant implementation may claim to support the **program_protocol** that corresponds to CATIA® FORTRAN programs, and conformance to some conformance class of View Exchange Protocol ISO 13584-101. This implementation shall be able to support functional model classes conformant with this conformance class even if some **program_reference**s reference **external_file_protocol**s that characterizes CATIA® FORTRAN programs.

NOTE 4 When an **external_file_protocol** not defined as standard data in some View Exchange Protocol is used in a library exchange context, it is assumed that there exists an agreement between the library data supplier and the library user about the description of this protocol through an **external_file_protocol** entity.

16 Exchange of general model classes: library integrated information model 24-1

Conformance to the library integrated information model 24-1 includes satisfying the requirements stated in the **ISO13584_g_m_iim_schema** presented in annex C, the requirements of the implementation method(s) supported, the relevant requirements of the normative references and the support of the standard data defined in annex E.

An implementation shall support at least the following implementation method: ISO10303-21. Requirements with respect to implementation methods are specified in annex F.

The **ISO13584_g_m_iim_schema** provides for a number of options that may be supported by an implementation. These options have been grouped into conformance classes. Thirteen conformance classes are defined. Conformance to the library integrated information model 24-1 requires, at a minimum, conformance to class 0. Options are defined by each class and may be selected by an implementation. Conformance to a particular conformance class requires that all the **ISO13584_g_m_iim_schema** entities, types and associated constraints defined as part of the class, be supported, together with the standard data associated with the class.

The numbering schema of the conformance classes is as follows:

— the digit specifies the dictionary data and possibly library data that shall be supported. The conformance classes identified by digit 0, 1 and 2 correspond to implementations that support only **dictionary_element**s.

— the conformance class identifier may contain the letter E. The corresponding conformance classes are called extended conformance classes. These conformance classes support descriptions that use supplier-defined protocols. Implementations conformant with an extended conformance class shall be able to process any entity belonging to the set of supported entities, even when they reference a protocol that is not part of the standard data but that is value-equal to an **external_file_protocol** they claim to support.

NOTE 1 The attribute values for the **external_file_protocol** entities that do not belong to the standard data defined in this part of ISO 13584 (annexes E, I and M) or to the standard data defined in one part of the view exchange protocol series of part of ISO 13584 are subject to prior agreement between the sender and the receiver. They are outside the scope of this International Standard.

NOTE 2 The only files that may be referenced as **http_file**s in conformance classes '1' to '6' of library integrated information model 24-1 are files whose MIME type and subtype:

— either corresponding to specifications that are publicly available, or

— that are associated with public domain Internet-available readers.

Reference to **http_file**s corresponding to other MIME types and subtypes may only be done in extended conformance classes. This is documented as an informal proposition in the annex E.

Conformance class are characterised as follows:

— class 0: minimal **dictionary_element**s for atomic items;

— class 1: **dictionary_element**s for atomic items;

— class 1E: **dictionary_element**s for atomic items with supplier-defined external file protocols;

— class 2: **dictionary_element**s for assembled items;

— class 2E: **dictionary_element**s for assembled items with supplier-defined external file protocols;

— class 3: **dictionary_element**s and simple library specifications for atomic items;

— class 3E: **dictionary_element**s and simple library specifications for atomic items with supplier-defined external file protocols;

— class 4: **dictionary_element**s and relational-algebra-based library specifications for atomic items;

— class 4E: **dictionary_element**s and relational algebra-based library specifications for atomic items with supplier-defined external file protocols;

— class 5: **dictionary_element** and simple library specifications for assembled items;

— class 5E: **dictionary_element** and simple library specifications for assembled items with supplier-defined external file protocols;

— class 6: **dictionary_element** and relational algebra-based library specifications for assembled items;

— class 6E: **dictionary_element** and relational algebra-based library specifications for assembled items with supplier-defined external file protocols.

Table 1 — Conformance options of library integrated information model 24-1

Capabilities	**Dictionary_elements**			**Library specification (class extension)**			**Supplier defined protocol**
Conformance class	atomic items: common dictionary schema	atomic items with graphics and referenced document	assembled items	atomic items with tables, (no relational algebra)	assembled items with tables, (no relational algebra)	relational algebra	
0	x						
1	x	x					
1-e	x	x					x
2	x	x	x				
2-e	x	x	x				x
3	x	x		x			
3-e	x	x		x			x
4	x	x		x		x	
4-e	x	x		x		x	x
5	x	x	x	x	x		
5-e	x	x	x	x	x		x
6	x	x	x	x	x	x	
6-e	x	x	x	x	x	x	x

Table 1 shows the supported capabilities of the different conformance classes of library integrated information model 24-1.

16.1 ISO13584_g_m_iim_schema short listing

This clause specifies the EXPRESS schema that uses elements either from the integrated resource series of ISO 10303 or from the logical resource or description methodology series of parts of ISO 13584 to define the requirements of library integrated information model LIIM24-1.

NOTE 1 The integrated resource series of ISO 10303 are part 10303-4x and 10303-1xx. The logical resource series of parts of ISO 13584 are ISO 13584-2x and the description methodology series of parts of ISO 13584 are ISO 13584-4x.

Requirements stated in the referenced schemas apply exclusively to those items that are used from these schemas.

The expanded EXPRESS listing is presented in annex C.

NOTE 2 Some of the entities and related constructs USEd in this schema do not belong to any of the conformance classes defined in this part of ISO 13584. They are intended to be used by view exchange protocols.

NOTE 3 This schema is the information model of supplier libraries. The information model of integrated libraries is outside the scope of this International Standard.

EXPRESS specification:

```
*)
SCHEMA ISO13584_g_m_iim_schema;

USE FROM ISO13584_generic_expressions_schema
     (environment);

USE FROM ISO13584_expressions_schema
     (abs_function,
     acos_function,
     and_expression,
     asin_function,
     atan_function,
     boolean_literal,
     boolean_variable,
     comparison_equal,
     comparison_greater,
     comparison_greater_equal,
     comparison_less,
     comparison_less_equal,
     comparison_not_equal,
     concat_expression,
     cos_function,
     div_expression,
     equals_expression,
     exp_function,
     format_function,
     index_expression,
     interval_expression,
     int_literal,
     int_numeric_variable,
     int_value_function,
     length_function,
     like_expression,
     log_function,
     log2_function,
     log10_function,
     maximum_function,
```



```
        minimum_function,
        minus_expression,
        minus_function,
        mod_expression,
        mult_expression,
        not_expression,
        odd_function,
        or_expression,
        plus_expression,
        power_expression,
        real_literal,
        real_numeric_variable,
        sin_function,
        slash_expression,
        square_root_function,
        string_literal,
        string_variable,
        substring_expression,
        tan_function,
        value_function,
        xor_expression);

    USE FROM ISO13584_IEC61360_dictionary_schema
        (axis1_placement_type,
        axis2_placement_2d_type,
        axis2_placement_3d_type,
        basic_semantic_unit,
        boolean_type,
        class_BSU,
        class_instance_type,
        class_value_assignment,
        component_class,
        condition_DET,
        data_type_BSU,
        data_type_element,
        dates,
        dependent_P_DET,
        dic_unit,
        dic_value,
        entity_instance_type,
        identified_document,
        integer_type,
        int_currency_type,
        int_measure_type,
        int_type,
        item_class,
        item_names,
        label_with_language,
        level_type,
        material_class,
        mathematical_string,
```



```
      named_type,
      non_dependent_P_DET,
      non_quantitative_code_type,
      non_quantitative_int_type,
      non_si_unit,
      number_type,
      placement_type,
      property_BSU,
      real_currency_type,
      real_measure_type,
      real_type,
      string_type,
      supplier_BSU,
      supplier_element,
      value_code_type,
      value_domain);

  USE FROM ISO13584_IEC61360_language_resource_schema
      (global_language_assignment,
      present_translations,
      translated_label,
      translated_text);

  USE FROM ISO13584_instance_resource_schema
      (Null_value,
      Primitive_value,
      Null_or_primitive_value,
      Simple_value,
      Null_or_simple_value,
      Number_value,
      Null_or_number_value,
      Integer_value,
      Null_or_integer_value,
      Real_value,
      Null_or_real_value,
      Boolean_value,
      Null_or_boolean_value,
      Translatable_string_value,
      Translated_string_value,
      String_value,
      Null_or_translatable_string_value,
      Complex_value,
      Null_or_complex_value,
      Entity_instance_value,
      Null_or_entity_instance_value,
      Defined_entity_instance_value,
      Controlled_entity_instance_value,
      STEP_entity_instance_value,
      PLIB_entity_instance_value,
      Property_or_data_type_BSU,
      Level_spec_value,
      Null_or_level_spec_value,
```

```
      Int_level_spec_value,
      Null_or_int_level_spec_value,
      Real_level_spec_value,
      Null_or_real_level_spec_value,
      Property_value,
      Context_dependent_property_value,
      dic_class_instance,
      null_or_dic_class_instance,
      dic_component_instance,
      dic_feature_instance,
      dic_material_instance,
      lib_component_instance,
      lib_feature_instance,
      lib_material_instance);

  USE FROM ISO13584_library_expressions_schema
      (binary_class_instance_constructor,
      class_instance_literal,
      class_instance_variable,
      entity_instance_literal,
      entity_instance_variable,
      exists_value,
      instance_comparison_equal,
      int_level_spec_literal,
      int_level_spec_variable,
      multiple_arity_class_instance_constructor,
      property_assignment,
      real_level_spec_literal,
      real_level_spec_variable,
      unary_class_instance_constructor);

  USE FROM ISO13584_table_resource_schema
      (boolean_column,
      class_instance_column,
      column_traversal_variable_semantics,
      difference_table_expression,
      entity_instance_column,
      integer_column,
      intersect_table_expression,
      int_level_spec_column,
      in_RDB_table_boolean_expression,
      multiple_arity_cartesian_product,
      natural_join_expression,
      projection_expression,
      RDB_table_extension,
      RDB_table_specification,
      RDB_table_variable,
      real_column,
      real_level_spec_column,
      select_expression,
      string_column,
```



```
      table_extension,
      table_literal,
      table_specification,
      table_variable,
      union_table_expression);

  USE FROM ISO13584_variable_semantics_schema
      (property_semantics_or_path,
      self_class_code_semantics,
      self_class_preferred_name_semantics,
      self_class_short_name_semantics,
      self_class_supplier_code_semantics,
      self_class_version_semantics,
      self_property_class_code_semantics,
      self_property_class_version_semantics,
      self_property_code_semantics,
      self_property_preferred_name_semantics,
      self_property_short_name_semantics,
      self_property_value_semantics,
      self_property_version_semantics,
      sub_property_path);

  USE FROM ISO13584_domain_resource_schema
      (constant_range_defined_domain,
      domain_restriction,
      functional_domain_restriction,
      guarded_functional_domain,
      guarded_simple_domain,
      library_expression_defined_value,
      null_defined_value,
      others,
      predicate_defined_domain,
      subclass_defined_domain,
      table_defined_domain,
      table_defined_value,
      type_defined_domain,
      variable_range_defined_domain);

  USE FROM ISO13584_extended_dictionary_schema
      (a_posteriori_case_of,
      class_document_relationship,
      class_table_relationship,
      component_class_case_of,
      dictionary_identification,
      dictionary,
      dictionary_in_standard_format,
      document_BSU,
      document_element,
      feature_class,
      feature_class_case_of,
      geometric_representation_context_type,
      item_class_case_of,
```

```
        library_iim_identification,
        material_class_case_of,
        RDB_table_content,
        RDB_table_element,
        representation_type,
        table_BSU,
        table_content,
        table_element,
        view_exchange_protocol_identification);

    USE FROM ISO13584_library_content_schema
        (explicit_item_class_extension,
        item_class_extension,
        library,
        library_in_standard_format,
        opt_or_mand_property_BSU);

    USE FROM ISO13584_external_file_schema
        (A6_illustration,
        A9_illustration,
        document_content,
        external_file_unit,
        http_class_directory,
        http_directory_name_type,
        http_file,
        http_protocol,
        illustration,
        language_specific_content,
        message,
        non_standard_data_protocol,
        not_translatable_external_content,
        not_translated_external_content,
        standard_data_protocol,
        translated_external_content,
        property_value_external_item);

    USE FROM measure_schema
        (amount_of_substance_measure,
        amount_of_substance_unit,
        area_measure,
        area_unit,
        context_dependent_measure,
        context_dependent_unit,
        conversion_based_unit,
        count_measure,
        derived_unit,
        derived_unit_element,
        descriptive_measure,
        dimensional_exponents,
        electric_current_measure,
        electric_current_unit,
```



```
      global_unit_assigned_context,
      length_measure,
      length_measure_with_unit,
      length_unit,
      luminous_intensity_measure,
      luminous_intensity_unit,
      mass_measure,
      mass_unit,
      measure_value,
      measure_with_unit,
      named_unit,
      numeric_measure,
      parameter_value,
      plane_angle_measure,
      plane_angle_unit,
      positive_length_measure,
      positive_plane_angle_measure,
      positive_ratio_measure,
      ratio_measure,
      ratio_unit,
      si_unit,
      solid_angle_measure,
      solid_angle_unit,
      thermodynamic_temperature_measure,
      thermodynamic_temperature_unit,
      time_measure,
      time_unit,
      volume_measure,
      volume_unit);

  USE FROM person_organization_schema
      (address,
      organization,
      person,
      person_and_organization,
      personal_address,
      organizational_address );

  USE FROM date_time_schema
      (date,
      date_and_time,
      local_time,
      calendar_date,
      ordinal_date,
      week_of_year_and_day_date);

  USE FROM application_context_schema
      (application_context,
      application_context_element,
      application_protocol_definition);

  (*
```

16.2 ISO13584_g_m_iim_schema global rule definitions

The following rules define the requirements for the **ISO13584_g_m_iim_schema** schema.

16.2.1 At_most_one_dictionary_rule rule

The **at_most_one_dictionary_rule** rule states that a library exchange context is associated to no more than one **dictionary**.

EXPRESS specification:

```
*)
RULE at_most_one_dictionary_rule FOR(dictionary);
WHERE
     WR1: SIZEOF(dictionary) <= 1;
END_RULE; -- at_most_one_library_rule
(*
```

Formal propositions:

WR1: there is no more than one **dictionary** defined in a library exchange context.

16.2.2 Class_associated_items_rule rule

The **class_associated_items_rule** rule states that each **class** has no more than two **associated_items** that are **class_table_relationship** and/or **class_document_relationship**.

EXPRESS specification:

```
*)
RULE class_associated_items_rule FOR(class);
WHERE
     WR1: QUERY(temp <* class | (SIZEOF(temp.associated_items) > 2)
          OR ((SIZEOF(temp.associated_items) = 1)
          AND NOT(('ISO13584_G_M_IIM_SCHEMA'
          +'.CLASS_TABLE_RELATIONSHIP' IN
          TYPEOF(temp.associated_items[1]))
          OR ('ISO13584_G_M_IIM_SCHEMA'
          +'.CLASS_DOCUMENT_RELATIONSHIP'
          IN TYPEOF(temp.associated_items[1]))))
          OR ((SIZEOF(temp.associated_items) = 2)
          AND NOT((('ISO13584_G_M_IIM_SCHEMA'
          + '.CLASS_TABLE_RELATIONSHIP'
          IN TYPEOF(temp.associated_items[1]))
          AND ('ISO13584_G_M_IIM_SCHEMA'
          +'.CLASS_DOCUMENT_RELATIONSHIP'
          IN TYPEOF(temp.associated_items[2])))
          OR (('ISO13584_G_M_IIM_SCHEMA'
          +'.CLASS_TABLE_RELATIONSHIP'
          IN TYPEOF(temp.associated_items[2]))
          AND ('ISO13584_G_M_IIM_SCHEMA'
```



```
          + '.CLASS_DOCUMENT_RELATIONSHIP'
          IN TYPEOF(temp.associated_items[1])))))
          = [];
END_RULE; -- class_associated_items_rule
(*
```

Formal propositions:

WR1: each **class** has no more than two **associated_items** that are **class_table_relationship** and/or **class_document_relationship**.

```
*)
END_SCHEMA; -- ISO13584_g_m_iim_schema
(*
```

16.3 Conformance class requirements

16.3.1 Conformance class 0

Conformance class 0 addresses those implementations that are intended to support the common requirements stated in the ISO/IEC dictionary schema. An implementation of conformance class 0 of library integrated information model 24-1 shall support the following entities and related constructs.

```
FROM ISO13584_IEC61360_dictionary_schema
   (boolean_type,
   class_BSU,
   class_value_assignment,
   component_class,
   condition_DET,
   data_type_element,
   dependent_P_DET,
   dic_unit,
   dic_value,
   identified_document,
   int_currency_type,
   int_measure_type,
   int_type,
   item_class,
   item_names,
   label_with_language,
   level_type,
   material_class,
   mathematical_string,
   named_type,
   non_dependent_P_DET,
   non_quantitative_code_type,
   non_quantitative_int_type,
   non_si_unit,
   number_type,
   property_BSU,
   real_currency_type,
   real_measure_type,
   real_type,
```



```
    string_type,
    supplier_BSU,
    supplier_element,
    syn_name_type,
    value_domain);

  FROM ISO13584_IEC61360_language_resource_schema
    (global_language_assignment,
    present_translations,
    translated_label,
    translated_text);

  FROM measure_schema
    (amount_of_substance_unit,
    area_unit,
    context_dependent_unit,
    conversion_based_unit,
    derived_unit,
    derived_unit_element,
    dimensional_exponents,
    electric_current_unit,
    length_measure_with_unit,
    length_unit,
    luminous_intensity_unit,
    mass_unit,
    measure_value,
    measure_with_unit,
    named_unit,
    plane_angle_unit,
    ratio_unit,
    si_unit,
    solid_angle_unit,
    thermodynamic_temperature_unit,
    time_unit,
    volume_unit);

    FROM person_organization_schema
    (address,
    organization,
    person,
    person_and_organization,
    personal_address,
    organizational_address );

    FROM date_time_schema
    (date,
    date_and_time,
    local_time,
    calendar_date,
    ordinal_date,
    week_of_year_and_day_date);
```


16.3.2 Conformance class 1

Conformance class 1 addresses those implementations that support **dictionary_element**s of atomic items that may involve the use of the is-case-of relationship and that may be associated with documents and graphics. An implementation of conformance class 1 of the library integrated information model 24-1 shall support the standard data defined in annex E. It shall also support the following entities and related constructs.

```
FROM ISO13584_IEC61360_dictionary_schema
   (boolean_type,
   class_BSU,
   class_value_assignment,
   component_class,
   condition_DET,
   data_type_BSU,
   dates,
   dic_unit,
   dic_value,
   identified_document,
   int_currency_type,
   int_measure_type,
   int_type,
   item_class,
   item_names,
   label_with_language,
   level,
   level_type,
   material_class,
   mathematical_string,
   named_type,
   non_dependent_P_DET,
   non_quantitative_code_type,
   non_quantitative_int_type,
   non_si_unit,
   number_type,
   property_BSU,
   real_currency_type,
   real_measure_type,
   real_type,
   string_type,
   supplier_BSU,
   supplier_element,
   syn_name_type,
   value_code_type,
   value_domain,
   value_format_type);

FROM ISO13584_IEC61360_language_resource_schema
   (global_language_assignment,
   language_code,
   present_translations,
```



```
    translated_label,
    translated_text);

  FROM ISO13584_extended_dictionary_schema
    (a_posteriori_case_of,
    class_document_relationship,
    component_class_case_of,
    dictionary_identification,
    dictionary_in_standard_format,
    document_BSU,
    document_element,
    document_element_with_http_access,
    document_element_with_translated_http_access,
    feature_class,
    feature_class_case_of,
    geometric_representation_context_type,
    item_class_case_of,
    library_iim_identification,
    material_class_case_of,
    referenced_document,
    referenced_graphics);

  FROM ISO13584_external_file_schema
    (A6_illustration,
    A9_illustration,
    document_content,
    external_file_unit,
    http_class_directory,
    http_file,
    http_protocol,
    illustration,
    language_specific_content,
    not_translatable_external_content,
    not_translated_external_content,
    standard_data_protocol,
    translated_external_content,
    property_value_external_item);

  FROM measure_schema
    (amount_of_substance_unit,
    area_unit,
    context_dependent_unit,
    conversion_based_unit,
    derived_unit,
    derived_unit_element,
    dimensional_exponents,
    electric_current_unit,
    length_measure_with_unit,
    length_unit,
    luminous_intensity_unit,
    mass_unit,
```



```
    measure_value,
    measure_with_unit,
    named_unit,
    plane_angle_unit,
    ratio_unit,
    si_unit,
    solid_angle_unit,
    thermodynamic_temperature_unit,
    time_unit,
    volume_unit);

FROM person_organization_schema
    (address,
    organization,
    person,
    person_and_organization,
    personal_address,
    organizational_address );

FROM date_time_schema
    (date,
    date_and_time,
    local_time,
    calendar_date,
    ordinal_date,
    week_of_year_and_day_date);

FROM application_context_schema
    (application_context,
    application_context_element,
    application_protocol_definition);
```

16.3.3 Conformance class 1E

Conformance class 1E addresses those implementations that support conformance class 1 and may support some supplier-defined external file protocols.

NOTE 1 The target of all the conformance classes that include 'E' in their identifiers are the consortiums that consist of one or several library data suppliers, and one or several library data users, who agree on some specific exchange format for some of the external files referenced from a library delivery file.

NOTE 2 Besides the entities and related constructs defined for conformance class 1, they shall also support the following entities and related constructs.

```
FROM ISO13584_extended_dictionary_schema
    (dictionary);

FROM ISO13584_external_file_schema
    (non_standard_data_protocol);
```

16.3.4 Conformance class 2

Conformance class 2 addresses those implementations that support **dictionary_element**s of assembled items that may involve the use of the is-case-of relationship and that may be associated with documents and graphics. An implementation of conformance class 2 of the library integrated

information model 24-1 shall support the standard data defined in annex I. It shall also support the entities and related constructs defined for conformance class 1 more the following entities and related constructs.

```
FROM ISO13584_IEC61360_dictionary_schema
   (class_instance_type);
```

16.3.5 Conformance class 2E

Conformance class 2E addresses those implementations that support conformance class 2 and may support some supplier-defined external file protocols.

Besides the entities and related constructs defined for conformance class 2, they shall also support the following entities and related constructs.

```
FROM ISO13584_extended_dictionary_schema
   (dictionary);

FROM ISO13584_external_file_schema
   (non_standard_data_protocol);
```

16.3.6 Conformance class 3

Conformance class 3 addresses those implementations that support **dictionary_element**s and simple library specifications of atomic items. Conformance class 3 implementations are only required to support a subset of the numeric, Boolean and string operators defined by the **ISO13584_expressions_schema**, and to support tables. They are not required to support any relational-algebra operator, nor properties whose data type is a **level_type**, nor access to the various names of the classes and properties.

NOTE It is considered that conformance class 3 should cover the requirements of most of the existing nonstandardised part libraries systems. A system supporting conformance class 3 might be for instance implemented on a file management system with some basic object oriented engine.

Besides the requirements already stated for conformance class 1, an implementation of conformance class 3 shall also support the following entities and associated constructs.

```
FROM ISO13584_generic_expressions_schema
   (environment);

FROM ISO13584_expressions_schema
   (and_expression,
   boolean_literal,
   boolean_variable,
   comparison_equal,
   comparison_greater,
   comparison_greater_equal,
   comparison_less,
   comparison_less_equal,
   comparison_not_equal,
   concat_expression,
   div_expression,
   equals_expression,
   format_function,
   int_literal,
```



```
    int_numeric_variable,
    maximum_function,
    minimum_function,
    minus_expression,
    minus_function,
    mult_expression,
    not_expression,
    or_expression,
    plus_expression,
    real_literal,
    real_numeric_variable,
    slash_expression,
    string_literal,
    string_variable,
    substring_expression,
    xor_expression);

FROM ISO13584_instance_resource_schema
    (Null_value,
    Primitive_value,
    Null_or_primitive_value,
    Simple_value,
    Null_or_simple_value,
    Number_value,
    Null_or_number_value,
    Integer_value,
    Null_or_integer_value,
    Real_value,
    Null_or_real_value,
    Boolean_value,
    Null_or_boolean_value,
    Translatable_string_value,
    Translated_string_value,
    String_value,
    Null_or_translatable_string_value);

FROM ISO13584_table_resource_schema
    (boolean_column,
    column_traversal_variable_semantics,
    integer_column,
    real_column,
    string_column,
    table_literal);

FROM ISO13584_variable_semantics_schema
    (self_property_value_semantics);

FROM ISO13584_domain_resource_schema
    (constant_range_defined_domain,
    domain_restriction,
    functional_domain_restriction,
    library_expression_defined_value,
```

```
    guarded_functional_domain,
    guarded_simple_domain,
    others,
    predicate_defined_domain,
    simple_domain,
    simple_functional_domain,
    table_defined_domain,
    table_defined_value,
    type_defined_domain,
    variable_range_defined_domain);

FROM ISO13584_extended_dictionary_schema
    (class_table_relationship,
    RDB_table_element,
    RDB_table_content,
    table_BSU);

FROM ISO13584_library_content_schema
    (explicit_item_class_extension,
    item_class_extension,
    library_in_standard_format,
    opt_or_mand_property_BSU);

FROM ISO13584_external_file_schema
    (message);
```

16.3.7 Conformance class 3E

Conformance class 3E addresses those implementations that support conformance class 3 and may support some supplier-defined external file protocols.

Besides the entities and related constructs defined for conformance class 3, they shall also support the following entities and related constructs.

```
FROM ISO13584_extended_dictionary_schema
    (dictionary);

FROM ISO13584_library_content_schema
    (library);

FROM ISO13584_external_file_schema
    (non_standard_data_protocol);
```

16.3.8 Conformance class 4

Conformance class 4 addresses those implementations that support **dictionary_element**s and library specifications of atomic items that involve all the operators defined in the **ISO13584_expressions_schema** and that support relational algebra for table definition, properties whose data type is a **level_type**, and access to the various names of the classes and properties.

NOTE The target systems for conformance class 4 are DBMS-based systems, where the database is either relational (with an object oriented engine) or object oriented.

Besides the requirements already stated for conformance class 3, an implementation of conformance class 4 shall also support the following entities and associated constructs.

```
FROM ISO13584_expressions_schema
   (abs_function,
   acos_function,
   asin_function,
   atan_function,
   cos_function,
   exp_function,
   index_expression,
   interval_expression,
   int_value_function,
   length_function,
   like_expression,
   log_function,
   log2_function,
   log10_function,
   mod_expression,
   odd_function,
   power_expression,
   sin_function,
   SQL_mappable_defined_function,
   square_root_function,
   tan_function,
   value_function);

FROM ISO13584_instance_resource_schema
   (int_level_spec_value,
   real_level_spec_value);

FROM ISO13584_library_expressions_schema
   (int_level_spec_literal,
   int_level_spec_variable,
   real_level_spec_literal,
   real_level_spec_variable);

FROM ISO13584_table_resource_schema
   (difference_table_expression,
   entity_instance_column,
   intersect_table_expression,
   int_level_spec_column,
   in_RDB_table_boolean_expression,
   multiple_arity_cartesian_product,
   natural_join_expression,
   projection_expression,
   real_level_spec_column,
   select_expression,
   union_table_expression);

FROM ISO13584_variable_semantics_schema
   (self_class_code_semantics,
```

```
    self_class_preferred_name_semantics,
    self_class_short_name_semantics,
    self_class_supplier_code_semantics,
    self_class_version_semantics,
    self_property_class_code_semantics,
    self_property_class_supplier_code_semantics,
    self_property_class_version_semantics,
    self_property_code_semantics,
    self_property_preferred_name_semantics,
    self_property_short_name_semantics,
    self_property_version_semantics);

FROM ISO13584_extended_dictionary_schema
    (table_content,
    table_element);
```

16.3.9 Conformance class 4E

Conformance class 4E addresses those implementations that support conformance class 4 and may support some supplier-defined external file protocol.

Besides the entities and related constructs defined for conformance class 4, they shall also support the following entities and related constructs;

```
FROM ISO13584_extended_dictionary_schema
    (dictionary);

FROM ISO13584_library_content_schema
    (library);

FROM ISO13584_external_file_schema
    (non_standard_data_protocol);
```

16.3.10 Conformance class 5

Conformance class 5 addresses those implementations that support **dictionary_element**s and simple library specifications of assembled items. Conformance class 5 implementations are only required to support a subset of the numeric, Boolean and string operators defined by the **ISO13584_expressions_schema**, and to support tables. They are not required to support any relational-algebra operator, nor properties whose data type is a **level_type**, nor access to the various names of the classes and properties.

NOTE The target systems for conformance class 5 are relational DBMS-based system.

Besides the entities and related constructs defined for conformance class 3 and for conformance class 2, they shall also support the following entities and related constructs.

```
FROM ISO13584_instance_resource_schema
    (Complex_value,
    Null_or_complex_value,
    Entity_instance_value,
    Null_or_entity_instance_value,
    Controlled_entity_instance_value,
```



```
    STEP_entity_instance_value,
    PLIB_entity_instance_value,
    Property_or_data_type_BSU,
    Level_spec_value,
    Null_or_level_spec_value,
    Int_level_spec_value,
    Null_or_int_level_spec_value,
    Real_level_spec_value,
    Null_or_real_level_spec_value,
    Property_value,
    Context_dependent_property_value,
    dic_class_instance,
    null_or_dic_class_instance,
    dic_component_instance,
    dic_feature_instance,
    dic_material_instance,
    lib_component_instance,
    lib_feature_instance,
    lib_material_instance);

FROM ISO13584_library_expressions_schema
    (binary_class_instance_constructor,
    class_instance_literal,
    class_instance_variable,
    entity_instance_literal,
    entity_instance_variable,
    exists_value,
    instance_comparison_equal,
    multiple_arity_class_instance_constructor,
    property_assignment,
    unary_class_instance_constructor);

FROM ISO13584_table_resource_schema
    (class_instance_column,
    RDB_table_variable);

FROM ISO13584_variable_semantics_schema
    (sub_property_path);

FROM ISO13584_domain_resource_schema
    (null_defined_value,
    subclass_defined_domain);
```

16.3.11 Conformance class 5E

Conformance class 5E addresses those implementations that support conformance class 5 and may support some supplier-defined external file protocols.

Besides the entities and related constructs defined for conformance class 5, they shall also support the following entities and related constructs;

```
FROM ISO13584_extended_dictionary_schema
    (dictionary);
```



```
FROM ISO13584_library_content_schema
   (library);

FROM ISO13584_external_file_schema
   (non_standard_data_protocol);
```

16.3.12 Conformance class 6

Conformance class 6 addresses those implementations that support **dictionary_element**s and library specifications of assembled items that involve all the operators defined in the **ISO13584_expressions_schema** and relational algebra for table definition.

NOTE The target systems for conformance class 6 are Object-Oriented DBMS-based system with full object-oriented capabilities.

Besides the entities and related constructs defined for conformance class 5, they shall also support the following entities and related constructs.

```
FROM ISO13584_expressions_schema
   (abs_function,
   acos_function,
   asin_function,
   atan_function,
   cos_function,
   exp_function,
   index_expression,
   interval_expression,
   int_value_function,
   length_function,
   like_expression,
   log_function,
   log2_function,
   log10_function,
   mod_expression,
   odd_function,
   power_expression,
   sin_function,
   SQL_mappable_defined_function,
   square_root_function,
   tan_function,
   value_function);

FROM ISO13584_instance_resource_schema
   (int_level_spec_value,
   real_level_spec_value);

FROM ISO13584_library_expressions_schema
   (int_level_spec_literal,
   int_level_spec_variable,
   real_level_spec_literal,
   real_level_spec_variable);
```



```
FROM ISO13584_table_resource_schema
   (difference_table_expression,
   intersect_table_expression,
   int_level_spec_column,
   in_RDB_table_boolean_expression,
   multiple_arity_cartesian_product,
   natural_join_expression,
   projection_expression,
   real_level_spec_column,
   select_expression,
   union_table_expression,
   table_variable);

FROM ISO13584_variable_semantics_schema
   (self_class_code_semantics,
   self_class_preferred_name_semantics,
   self_class_short_name_semantics,
   self_class_supplier_code_semantics,
   self_class_version_semantics,
   self_property_class_code_semantics,
   self_property_class_version_semantics,
   self_property_code_semantics,
   self_property_preferred_name_semantics,
   self_property_short_name_semantics,
   self_property_version_semantics);

FROM ISO13584_extended_dictionary_schema
   (table_content,
   table_element);
```

16.3.13 Conformance class 6E

Conformance class 6E addresses those implementations that support conformance class 6 and may support some supplier-defined external file protocols.

Besides the entities and related constructs defined for conformance class 6, they shall also support the following entities and related constructs;

```
FROM ISO13584_extended_dictionary_schema
   (dictionary);

FROM ISO13584_library_content_schema
   (library);

FROM ISO13584_external_file_schema
   (non_standard_data_protocol);
```

17 Exchange of functional model classes: library integrated information model 24-2

Conformance to the LIIM 24-2 includes satisfying the requirements stated in the **ISO13584_f_m_iim_schema** presented in annex G, the requirements of the implementation

method(s) supported, the relevant requirements of the normative references and the support of the standard data defined in annex I.

An implementation shall support at least the following implementation method: ISO10303-21. Requirement with respect to implementation methods are specified in annex J.

The **ISO13584_f_m_iim_schema** provides for a number of options that may be supported by an implementation. These options have been grouped into conformance classes. Ten conformance classes are defined. Conformance to the LIIM 24-2 requires, at a minimum, conformance to class 1. Options are defined by each class and may be selected by an implementation. Conformance to a particular conformance class requires that all the **ISO13584_f_m_iim_schema** entities, types and associated constraints defined as part of the class, be supported, together with the standard data associated with the class.

The numbering schema of the conformance classes is as follows:

— the digit specifies the dictionary elements and the library specifications (i.e., class extension description) that shall be supported. The conformance classes identified by digit 1 and 2 correspond to implementations that support only dictionaries elements.

— the conformance class identifier may contain the letter E. The corresponding conformance classes are called extended conformance classes. These conformance classes support descriptions that use supplier-defined external file protocols. Implementation conformant with an extended conformance class shall be able to process any entity belonging to the set of supported entities, even when they reference an external file protocol that is not part of the standard data but that is value-equal to an **external_file_protocol** they claim to support.

NOTE The attribute values for the **external_file_protocol** entities that do not belong to the standard data defined in this part of ISO 13584 (annexes E, I and M) or to the standard data defined in one part of the view exchange protocol series of part of ISO 13584 are subject to prior agreement between the sender and the receiver. They are outside the scope of ISO 13584.

Conformance class are characterised as follows:

— class 1: **dictionary_element**s for atomic functional models;

— class 1E: **dictionary_element**s for atomic functional models with supplier-defined external file protocols;

— class 2: **dictionary_element**s for functional models that reference subobjects;

— class 2E: **dictionary_element**s for functional models that reference subobjects with supplier-defined external file protocols;

— class 3: **dictionary_element**s and simple library specifications for atomic functional models;

— class 3E: **dictionary_element**s and simple library specifications for atomic functional models with supplier-defined external file protocols;

— class 4: **dictionary_element**s and relational-algebra-based library specifications for atomic functional models;

— class 4E: **dictionary_element**s and relational algebra-based library specifications for atomic functional models with supplier-defined external file protocols;

— class 5: **dictionary_element**s and simple library specifications for functional models that reference subobjects;

— class 5E: **dictionary_element**s and simple library specifications for functional models that reference subobjects with supplier-defined external file protocols;

— class 6: **dictionary_element**s and relational algebra-based library specifications for functional models that reference subobjects;

— class 6E: **dictionary_element**s and relational algebra-based library specifications for functional models that reference subobjects with supplier-defined external file protocols.

Table 2, below, shows the inclusion structure of the supported capabilities of the different conformance classes of LIIM 24-2.

Table 2 — Conformance options of library integrated information model 24-2

Capabilities	**Dictionary elements**		**Library specification (class extension)**			**Supplier defined protocol**
Conformance class	atomic items with graphics and referenced document	assembled items	atomic items with tables, (no relational algebra)	assembled items with tables, (no relational algebra)	relational algebra	
1	x					
1-e	x					x
2	x	x				
2-e	x	x				x
3	x		x			
3-e	x		x			x
4	x		x		x	
4-e	x		x		x	x
5	x	x	x	x		
5-e	x	x	x	x		x
6	x	x	x	x	x	
6-e	x	x	x	x	x	x

17.1 ISO13584_f_m_iim_schema short listing

This clause specifies the EXPRESS schema that uses elements either from the integrated resources of ISO 10303 (ISO 10303-4x) or from the logical resource or description methodology series of parts of ISO 13584 (ISO 13584-2x and ISO 13584-4x) to define the requirements of LIIM 24-2.

Requirements stated in the referenced schemas apply exclusively to those items that are used from these schemas.

The expanded EXPRESS listing of this schema, is presented in annex G.

NOTE 1 Some of the entities and related constructs USEd in this schema do not belong to any of the conformance classes defined in this part of ISO 13584. They are intended to be possibly allowed by some part of the view exchange protocol series of part of ISO 13584.

NOTE 2 This schema is the information model of supplier libraries. The information model of integrated libraries is outside the scope of ISO 13584.

EXPRESS Specification:

```
*)
SCHEMA ISO13584_f_m_iim_schema;

USE FROM ISO13584_generic_expressions_schema
     (environment);

USE FROM ISO13584_expressions_schema
     (abs_function,
     acos_function,
     and_expression,
     asin_function,
     atan_function,
     boolean_literal,
     boolean_variable,
     comparison_equal,
     comparison_greater,
     comparison_greater_equal,
     comparison_less,
     comparison_less_equal,
     comparison_not_equal,
     concat_expression,
     cos_function,
     div_expression,
     equals_expression,
     exp_function,
     format_function,
     index_expression,
     interval_expression,
     int_literal,
     int_numeric_variable,
     int_value_function,
     length_function,
     like_expression,
     log_function,
     log2_function,
     log10_function,
     maximum_function,
     minimum_function,
     minus_expression,
     minus_function,
     mod_expression,
     mult_expression,
     not_expression,
     odd_function,
     or_expression,
     plus_expression,
     power_expression,
     real_literal,
     real_numeric_variable,
```



```
      sin_function,
      slash_expression,
      square_root_function,
      string_literal,
      string_variable,
      substring_expression,
      tan_function,
      value_function,
      xor_expression);

  USE FROM ISO13584_IEC61360_dictionary_schema
      (axis1_placement_type,
      axis2_placement_2d_type,
      axis2_placement_3d_type,
      basic_semantic_unit,
      boolean_type,
      class_BSU,
      class_instance_type,
      data_type_BSU,
      data_type_element,
      dates,
      dic_unit,
      dic_value,
      entity_instance_type,
      identified_document,
      integer_type,
      int_currency_type,
      int_measure_type,
      int_type,
      item_names,
      label_with_language,
      level_type,
      mathematical_string,
      named_type,
      non_quantitative_code_type,
      non_quantitative_int_type,
      non_si_unit,
      number_type,
      placement_type,
      property_BSU,
      real_currency_type,
      real_measure_type,
      real_type,
      string_type,
      supplier_BSU,
      supplier_element,
      value_code_type,
      value_domain);

  USE FROM ISO13584_IEC61360_language_resource_schema
      (global_language_assignment,
      present_translations,
```

```
        translated_label,
        translated_text);

    USE FROM ISO13584_instance_resource_schema
        (Null_value,
        Primitive_value,
        Null_or_primitive_value,
        Simple_value,
        Null_or_simple_value,
        Number_value,
        Null_or_number_value,
        Integer_value,
        Null_or_integer_value,
        Real_value,
        Null_or_real_value,
        Boolean_value,
        Null_or_boolean_value,
        Translatable_string_value,
        Translated_string_value,
        String_value,
        Null_or_translatable_string_value,
        Complex_value,
        Null_or_complex_value,
        Entity_instance_value,
        Null_or_entity_instance_value,
        Defined_entity_instance_value,
        Controlled_entity_instance_value,
        STEP_entity_instance_value,
        PLIB_entity_instance_value,
        Property_or_data_type_BSU,
        Level_spec_value,
        Null_or_level_spec_value,
        Int_level_spec_value,
        Null_or_int_level_spec_value,
        Real_level_spec_value,
        Null_or_real_level_spec_value,
        Property_value,
        Context_dependent_property_value,
        dic_class_instance,
        null_or_dic_class_instance,
        dic_f_model_instance,
        lib_f_model_instance);

    USE FROM ISO13584_library_expressions_schema
        (binary_class_instance_constructor,
        class_instance_literal,
        class_instance_variable,
        entity_instance_literal,
        entity_instance_variable,
        exists_value,
        instance_comparison_equal,
```



```
      int_level_spec_literal,
      int_level_spec_variable,
      multiple_arity_class_instance_constructor,
      property_assignment,
      real_level_spec_literal,
      real_level_spec_variable,
      unary_class_instance_constructor);

  USE FROM ISO13584_table_resource_schema
      (boolean_column,
      class_instance_column,
      column_traversal_variable_semantics,
      difference_table_expression,
      entity_instance_column,
      integer_column,
      intersect_table_expression,
      int_level_spec_column,
      in_RDB_table_boolean_expression,
      multiple_arity_cartesian_product,
      natural_join_expression,
      projection_expression,
      RDB_table_extension,
      RDB_table_specification,
      RDB_table_variable,
      real_column,
      real_level_spec_column,
      select_expression,
      string_column,
      table_extension,
      table_literal,
      table_specification,
      table_variable,
      union_table_expression);

  USE FROM ISO13584_variable_semantics_schema
      (open_view_property_value_semantics,
      self_class_code_semantics,
      self_class_preferred_name_semantics,
      self_class_short_name_semantics,
      self_class_supplier_code_semantics,
      self_class_version_semantics,
      self_property_class_code_semantics,
      self_property_class_version_semantics,
      self_property_code_semantics,
      self_property_preferred_name_semantics,
      self_property_short_name_semantics,
      self_property_value_semantics,
      self_property_version_semantics,
      sub_property_path);

  USE FROM ISO13584_domain_resource_schema
      (constant_range_defined_domain,
```

```
      domain_restriction,
      functional_domain_restriction,
      library_expression_defined_value,
      guarded_functional_domain,
      guarded_simple_domain,
      null_defined_value,
      others,
      predicate_defined_domain,
      subclass_defined_domain,
      table_defined_domain,
      table_defined_value,
      type_defined_domain,
      variable_range_defined_domain);

  USE FROM ISO13584_extended_dictionary_schema
      (a_posteriori_view_of,
      class_document_relationship,
      class_table_relationship,
      dictionary_identification,
      dictionary,
      dictionary_in_standard_format,
      document_BSU,
      document_element,
      fm_class_view_of,
      functional_model_class,
      geometric_representation_context_type,
      library_iim_identification,
      program_library_BSU,
      program_library_element,
      program_reference_type,
      RDB_table_content,
      RDB_table_element,
      representation_P_DET,
      representation_type,
      representation_reference_type,
      supplier_program_library_relationship,
      table_BSU,
      table_content,
      table_element,
      view_control_variable_range,
      view_exchange_protocol_identification);

  USE FROM ISO13584_library_content_schema
      (explicit_functional_model_class_extension,
      functional_model_class_extension,
      library,
      library_in_standard_format,
      opt_or_mand_property_BSU);

  USE FROM ISO13584_external_file_schema
      (A6_illustration,
```



```
      A9_illustration,
      document_content,
      external_file_unit,
      http_class_directory,
      http_file,
      http_protocol,
      IAB_RFC,
      illustration,
      language_specific_content,
      linked_interface_program_protocol,
      message,
      non_standard_data_protocol,
      non_standard_simple_program_protocol,
      not_translatable_external_content,
      not_translated_external_content,
      program_library_content,
      program_reference,
      representation_reference,
      standard_data_protocol,
      standard_simple_program_protocol,
      translated_external_content,
      property_value_external_item);

  USE FROM ISO13584_method_schema
      (assignment_statement,
      begin_set,
      call_program_statement,
      close_set,
      constructed_sub_model_view_statement,
      guarded_statement,
      method,
      method_body,
      method_specif,
      method_statement,
      null_statement,
      referenced_sub_item_view_statement,
      send_representation_statement,
      send_representation_reference_statement,
      set_2d_relative_view_level,
      set_reference_lcs);

  USE FROM measure_schema
      (amount_of_substance_measure,
      amount_of_substance_unit,
      area_measure,
      area_unit,
      context_dependent_measure,
      context_dependent_unit,
      conversion_based_unit,
      count_measure,
      derived_unit,
      derived_unit_element,
```

```
      descriptive_measure,
      dimensional_exponents,
      electric_current_measure,
      electric_current_unit,
      global_unit_assigned_context,
      length_measure,
      length_measure_with_unit,
      length_unit,
      luminous_intensity_measure,
      luminous_intensity_unit,
      mass_measure,
      mass_unit,
      measure_value,
      measure_with_unit,
      named_unit,
      numeric_measure,
      parameter_value,
      plane_angle_measure,
      plane_angle_unit,
      positive_length_measure,
      positive_plane_angle_measure,
      positive_ratio_measure,
      ratio_measure,
      ratio_unit,
      si_unit,
      solid_angle_measure,
      solid_angle_unit,
      thermodynamic_temperature_measure,
      thermodynamic_temperature_unit,
      time_measure,
      time_unit,
      volume_measure,
      volume_unit);

  USE FROM person_organization_schema
      (address,
      organization,
      person,
      person_and_organization,
      personal_address,
      organizational_address );

  USE FROM date_time_schema
      (date,
      date_and_time,
      local_time,
      calendar_date,
      ordinal_date,
      week_of_year_and_day_date);
```



```
USE FROM geometry_schema
      (axis1_placement,
      axis2_placement_2D,
      axis2_placement_3D,
      geometric_representation_context,
      placement);

USE FROM representation_schema
      (representation,
      representation_item,
      representation_context);


USE FROM application_context_schema
      (application_context,
      application_context_element,
      application_protocol_definition);


(*
```

17.2 ISO13584_f_m_iim_schema global rule definitions

The following rules define the requirements for the **ISO13584_f_m_iim_schema** schema.

17.2.1 Exactly_one_dictionary_rule rule

The **exactly_one_dictionary_rule** rule states that a library exchange context is associated to exactly one **dictionary**.

EXPRESS specification:

```
*)
RULE exactly_one_dictionary_rule FOR(dictionary);
WHERE
      WR1: SIZEOF(dictionary) = 1;
END_RULE; -- exactly_one_library_rule
(*
```

Formal propositions:

WR1: there is exactly one **dictionary** defined in a library exchange context.

17.2.2 Class_associated_items_rule rule

The **class_associated_items_rule** rule states that each **class** has no more than two **associated_items** that are **class_table_relationship** and/or **class_document_relationship**.

EXPRESS specification:

```
*)
RULE class_associated_items_rule FOR(class);
WHERE
```



```
        WR1: QUERY(temp <* class | (SIZEOF(temp.associated_items) > 2)
             OR ((SIZEOF(temp.associated_items) = 1)
             AND NOT(('ISO13584_F_M_IIM_SCHEMA'
             +'.CLASS_TABLE_RELATIONSHIP' IN
             TYPEOF(temp.associated_items[1]))
             OR ('ISO13584_F_M_IIM_SCHEMA'
             +'.CLASS_DOCUMENT_RELATIONSHIP'
             IN TYPEOF(temp.associated_items[1]))))
             OR ((SIZEOF(temp.associated_items) = 2)
             AND NOT((('ISO13584_F_M_IIM_SCHEMA'
             + '.CLASS_TABLE_RELATIONSHIP'
             IN TYPEOF(temp.associated_items[1]))
             AND ('ISO13584_F_M_IIM_SCHEMA'
             +'.CLASS_DOCUMENT_RELATIONSHIP'
             IN TYPEOF(temp.associated_items[2])))
             OR (('ISO13584_F_M_IIM_SCHEMA'
             +'.CLASS_TABLE_RELATIONSHIP'
             IN TYPEOF(temp.associated_items[2]))
             AND ('ISO13584_F_M_IIM_SCHEMA'
             + '.CLASS_DOCUMENT_RELATIONSHIP'
             IN TYPEOF(temp.associated_items[1])))))))
             = [];
    END_RULE; -- class_associated_items_rule
    (*
```

Formal propositions:

WR1: each **class** has no more than two **associated_items** that are **class_table_relationship** and/or **class_document_relationship**.

17.2.3 Supplier_associated_items_rule rule

The **supplier_associated_items_rule** rule states that each supplier has no more than one **associated_items** that is a **supplier_program_library_relationship**.

EXPRESS specification:

```
    *)
    RULE supplier_associated_items_rule FOR(supplier_element);
    WHERE
         WR1: QUERY(temp <* supplier_element
              | (SIZEOF(temp.associated_items) > 1)
              OR ((SIZEOF(temp.associated_items) = 1)
              AND NOT('ISO13584_F_M_IIM_SCHEMA'
              + '.SUPPLIER_PROGRAM_LIBRARY_RELATIONSHIP'
              IN TYPEOF(temp.associated_items)))) = [];
    END_RULE; -- supplier_associated_items_rule
    (*
```


Formal propositions:

WR1: each supplier has no more than one **associated_items** that is a **supplier_program_library_relationship**.

NOTE A **supplier_program_library_relationship** may refer to zero program.

```
*)

END_SCHEMA; -- ISO13584_f_m_iim_schema

(*
```

17.3 Conformance class requirements

17.3.1 Conformance class 1

Conformance class 1 addresses those implementations that support dictionary elements of atomic functional models that may involve the use of the is-view-of relationship and that may be associated with documents and graphics. An implementation of conformance class 1 of the LIIM 24-2 shall support the standard data defined in annex I. It shall also support the following entities and related constructs.

```
FROM ISO13584_IEC61360_dictionary_schema
   (boolean_type,
   class_BSU,
   data_type_BSU,
   data_type_element,
   dates,
   dic_unit,
   dic_value,
   identified_document,
   int_currency_type,
   int_measure_type,
   int_type,
   item_names,
   label_with_language,
   mathematical_string,
   named_type,
   non_quantitative_code_type,
   non_quantitative_int_type,
   non_si_unit,
   number_type,
   property_BSU,
   real_currency_type,
   real_measure_type,
   real_type,
   string_type,
   supplier_BSU,
   supplier_element,
   syn_name_type,
   value_domain);

FROM ISO13584_IEC61360_language_resource_schema
```



```
    (global_language_assignment,
    present_translations,
    translated_label,
    translated_text);

  FROM ISO13584_extended_dictionary_schema
    (a_posteriori_view_of,
    class_document_relationship,
    dictionary_identification,
    dictionary_in_standard_format,
    library_iim_identification,
    document_BSU,
    document_element,
    document_element_with_http_access,
    document_element_with_translated_http_access,
    fm_class_view_of,
    functional_model_class,
    geometric_representation_context_type,
    RDB_table_content,
    referenced_document,
    referenced_graphics,
    representation_P_DET,
    representation_reference_type,
    view_control_variable_range,
    view_exchange_protocol_identification);

  FROM ISO13584_external_file_schema
    (A6_illustration,
    A9_illustration,
    document_content,
    external_file_unit,
    http_class_directory,
    http_file,
    http_protocol,
    illustration,
    language_specific_content,
    not_translatable_external_content,
    not_translated_external_content,
    standard_data_protocol,
    translated_external_content,
    property_value_external_item);

  FROM measure_schema
    (amount_of_substance_unit,
    area_unit,
    context_dependent_unit,
    conversion_based_unit,
    derived_unit,
    derived_unit_element,
    dimensional_exponents,
    electric_current_unit,
```



```
    global_unit_assigned_context,
    length_measure_with_unit,
    length_unit,
    luminous_intensity_unit,
    mass_unit,
    measure_value,
    measure_with_unit,
    named_unit,
    plane_angle_unit,
    ratio_unit,
    si_unit,
    solid_angle_unit,
    thermodynamic_temperature_unit,
    time_unit,
    volume_unit);

  FROM person_organization_schema
    (address,
    organization,
    person,
    person_and_organization,
    personal_address,
    organizational_address );

  FROM date_time_schema
    (date,
    date_and_time,
    local_time,
    calendar_date,
    ordinal_date,
    week_of_year_and_day_date);

  FROM application_context_schema
    (application_context,
    application_context_element,
    application_protocol_definition);
```

17.3.2 Conformance class 1E

Conformance class 1E addresses those implementations that support conformance class 1 and may support some supplier-defined external file protocols.

NOTE The target of all the conformance classes that include 'E' in their identifiers are the functional models suppliers who want to use a specific exchange format (for instance, the specific format of some CAD system) for the programs or representations referenced from a library delivery file.

Besides the entities and related constructs defined for conformance class 1, they shall also support the following entities and related constructs.

```
  FROM ISO13584_extended_dictionary_schema
    (dictionary);

  FROM ISO13584_external_file_schema
```


(non_standard_data_protocol);

17.3.3 Conformance class 2

Conformance class 2 addresses those implementations that support dictionary elements of both atomic functional models and of assembled functional models that reference subobjects. An implementation of conformance class 2 of the LIIM 24-2 shall support the standard data defined in annex I. It shall also support the entities and related constructs defined for conformance class 1, more the following entities and related constructs:

FROM ISO13584_IEC61360_dictionary_schema
(axis1_placement_type,
axis2_placement_2d_type,
axis2_placement_3d_type,
class_instance_type,
placement_type);

FROM ISO13584_extended_dictionary_schema
(representation_type);

17.3.4 Conformance class 2E

Conformance class 2E addresses those implementations that support conformance class 2 and may support some supplier-defined external file protocols.

Besides the entities and related constructs defined for conformance class 2, they shall also support the following entities and related constructs.

FROM ISO13584_extended_dictionary_schema(
dictionary);

FROM ISO13584_external_file_schema(
non_standard_data_protocol);

17.3.5 Conformance class 3

Conformance class 3 addresses those implementations that support dictionary elements and simple library specifications for atomic functional models. Conformance class 3 implementations are only required to support a subset of the numeric, Boolean and string operators defined by the **ISO13584_expressions_schema**, and to support tables. They are not required to support any relational-algebra operator, nor properties whose data type is a **level_type**, nor access to the various names of the classes and properties.

NOTE It is considered that conformance class 3 should cover the requirements of most of the existing nonstandardised part libraries systems. A system supporting conformance class 3 might be for instance implemented on a file management system with some basic object oriented engine.

Besides the requirements already stated for conformance class 1, an implementation of conformance class 3 shall also support the following entities and associated constructs.

FROM ISO13584_generic_expressions_schema
(environment);

FROM ISO13584_expressions_schema
(and_expression,
boolean_literal,


```
    boolean_variable,
    comparison_equal,
    comparison_greater,
    comparison_greater_equal,
    comparison_less,
    comparison_less_equal,
    comparison_not_equal,
    concat_expression,
    div_expression,
    equals_expression,
    format_function,
    int_literal,
    int_numeric_variable,
    maximum_function,
    minimum_function,
    minus_expression,
    minus_function,
    mult_expression,
    not_expression,
    or_expression,
    plus_expression,
    real_literal,
    real_numeric_variable,
    slash_expression,
    string_literal,
    string_variable,
    substring_expression,
    xor_expression);

  FROM ISO13584_instance_resource_schema
    (Null_value,
    Primitive_value,
    Null_or_primitive_value,
    Simple_value,
    Null_or_simple_value,
    Number_value,
    Null_or_number_value,
    Integer_value,
    Null_or_integer_value,
    Real_value,
    Null_or_real_value,
    Boolean_value,
    Null_or_boolean_value,
    Translatable_string_value,
    Translated_string_value,
    String_value,
  Null_or_translatable_string_value);

  FROM ISO13584_library_expressions_schema
    (entity_instance_literal,
    entity_instance_variable);
```

```
FROM ISO13584_table_resource_schema
   (boolean_column,
   column_traversal_variable_semantics,
   entity_instance_column,
   integer_column,
   real_column,
   string_column,
   table_literal);

FROM ISO13584_variable_semantics_schema
   (open_view_property_value_semantics,
   self_property_value_semantics);

FROM ISO13584_domain_resource_schema
   (constant_range_defined_domain,
   domain_restriction,
   functional_domain_restriction,
   library_expression_defined_value,
   guarded_functional_domain,
   guarded_simple_domain,
   others,
   predicate_defined_domain,
   simple_domain,
   simple_functional_domain,
   table_defined_domain,
   table_defined_value,
   type_defined_domain,
   variable_range_defined_domain);

FROM ISO13584_extended_dictionary_schema
   (class_table_relationship,
   program_library_BSU,
   program_library_element,
   program_reference_type,
   RDB_table_element,
   supplier_program_library_relationship,
   table_BSU,
   table_content,
   table_element);

FROM ISO13584_library_content_schema
   (explicit_functional_model_class_extension,
   functional_model_class_extension,
   library_in_standard_format,
   opt_or_mand_property_BSU);

FROM ISO13584_external_file_schema
   (linked_interface_program_protocol,
   message,
   program_library_content,
   program_reference,
```



```
    representation_reference,
    standard_simple_program_protocol,
    supplier_BSU_related_content);

  FROM ISO13584_method_schema
    (assignment_statement,
    begin_set,
    call_program_statement,
    close_set,
    guarded_statement,
    method,
    method_body,
    method_specif,
    method_statement,
    null_statement,
    send_representation_reference_statement,
    send_representation_statement,
    set_reference_lcs);
```

17.3.6 Conformance class 3E

Conformance class 3E addresses those implementations that support conformance class 3 and may support some supplier-defined external file protocols.

Besides the entities and related constructs defined for conformance class 3, they shall also support the following entities and related constructs.

```
  FROM ISO13584_extended_dictionary_schema
    (dictionary);

  FROM ISO13584_library_content_schema
    (library);

  FROM ISO13584_external_file_schema
    (non_standard_data_protocol,
    non_standard_simple_program_protocol);
```

17.3.7 Conformance class 4

Conformance class 4 addresses those implementations that support dictionary elements and library specifications of atomic functional models that involve all the operators defined in the **ISO13584_expressions_schema**, that support relational algebra for table definition, and access to the various names of the classes and properties.

NOTE The target systems for conformance class 4 are DBMS-based systems, where the databases are either relational (with an object oriented engine) or object oriented.

Besides the requirements already stated for conformance class 3, an implementation of conformance class 4 shall also support the following entities and associated constructs.

```
  FROM ISO13584_expressions_schema
    (abs_function,
    acos_function,
    asin_function,
    atan_function,
```



```
    cos_function,
    exp_function,
    index_expression,
    interval_expression,
    int_value_function,
    length_function,
    like_expression,
    log_function,
    log2_function,
    log10_function,
    mod_expression,
    odd_function,
    power_expression,
    sin_function,
    SQL_mappable_defined_function,
    square_root_function,
    tan_function,
    value_function);

  FROM ISO13584_table_resource_schema
    (difference_table_expression,
    intersect_table_expression,
    in_RDB_table_boolean_expression,
    multiple_arity_cartesian_product,
    natural_join_expression,
    projection_expression,
    select_expression,
    union_table_expression);

  FROM ISO13584_variable_semantics_schema
    (self_class_code_semantics,
    self_class_preferred_name_semantics,
    self_class_short_name_semantics,
    self_class_supplier_code_semantics,
    self_class_version_semantics,
    self_property_class_code_semantics,
    self_property_class_supplier_code_semantics,
    self_property_class_version_semantics,
    self_property_code_semantics,
    self_property_preferred_name_semantics,
    self_property_short_name_semantics,
    self_property_version_semantics);

  FROM ISO13584_method_schema
    (set_2d_relative_view_level);

  FROM geometry_schema
    (axis1_placement,
    axis2_placement_2D,
    axis2_placement_3D,
    direction,
```



```
    geometric_representation_context,
    placement,
    point);

  FROM representation_schema
    (representation,
    representation_context,
    representation_item);
```

17.3.8 Conformance class 4E

Conformance class 4E addresses those implementations that support conformance class 4 and may support some supplier-defined external file protocol.

Besides the entities and related constructs defined for conformance class 4, they shall also support the following entities and related constructs;

```
  FROM ISO13584_extended_dictionary_schema
    (dictionary);

  FROM ISO13584_library_content_schema
    (library);

  FROM ISO13584_external_file_schema
    (non_standard_data_protocol,
    non_standard_simple_program_protocol);
```

17.3.9 Conformance class 5

Conformance class 5 addresses those implementations that support dictionary elements and simple library specifications of functional models that reference subobjects to create view of assembled items. Conformance class 5 implementations are only required to support a subset of the numeric, Boolean and string operators defined by the **ISO13584_expressions_schema**, and to support tables. They are not required to support any relational-algebra operator, nor properties whose data type is a **level_type**, nor access to the various names of the classes and properties.

Besides the entities and related constructs defined for conformance class 3 and for conformance class 2, they shall also support the following entities and related constructs;

```
  FROM ISO13584_instance_resource_schema
    (Complex_value,
    Null_or_complex_value,
    Entity_instance_value,
    Null_or_entity_instance_value,
    Defined_entity_instance_value,
    Controlled_entity_instance_value,
    STEP_entity_instance_value,
    PLIB_entity_instance_value,
    Property_or_data_type_BSU,
    Level_spec_value,
    Null_or_level_spec_value,
    Int_level_spec_value,
    Null_or_int_level_spec_value,
    Real_level_spec_value,
    Null_or_real_level_spec_value,
```



```
    Property_value,
    Context_dependent_property_value,
    dic_class_instance,
    null_or_dic_class_instance,
    dic_f_model_instance,
    lib_f_model_instance);

FROM ISO13584_library_expressions_schema
    (binary_class_instance_constructor,
    class_instance_literal,
    class_instance_variable,
    exists_value,
    instance_comparison_equal,
    multiple_arity_class_instance_constructor,
    property_assignment,
    unary_class_instance_constructor);

FROM ISO13584_table_resource_schema
    (class_instance_column,
    RDB_table_variable);

FROM ISO13584_variable_semantics_schema
    (sub_property_path);

FROM ISO13584_domain_resource_schema
    (null_defined_value,
    subclass_defined_domain);

FROM ISO13584_method_schema
    (constructed_sub_model_view_statement);
```

17.3.10 Conformance class 5E

Conformance class 5E addresses those implementations that support conformance class 5 and may support some supplier-defined external file protocol.

Besides the entities and related constructs defined for conformance class 5, they shall also support the following entities and related constructs;

```
FROM ISO13584_extended_dictionary_schema
    (dictionary);

FROM ISO13584_library_content_schema
    (library);

FROM ISO13584_external_file_schema
    (compiler_version_type,
    non_standard_data_protocol,
    non_standard_simple_program_protocol);
```


17.3.11 Conformance class 6

Conformance class 6 addresses those implementations that support dictionary elements and library specifications of functional models that reference subobjects to create view of assembled items.

NOTE The target systems for conformance class 6 are 0bject-Oriented DBMS-based system with full object-oriented capabilities.

Besides the entities and related constructs defined for conformance class 5, they shall also support the following entities and related constructs;

```
FROM ISO13584_expressions_schema
   (abs_function,
   acos_function,
   asin_function,
   atan_function,
   cos_function,
   exp_function,
   index_expression,
   interval_expression,
   int_value_function,
   length_function,
   like_expression,
   log_function,
   log2_function,
   log10_function,
   mod_expression,
   odd_function,
   power_expression,
   sin_function,
   SQL_mappable_defined_function,
   square_root_function,
   tan_function,
   value_function);

FROM ISO13584_table_resource_schema
   (difference_table_expression,
   intersect_table_expression,
   in_RDB_table_boolean_expression,
   multiple_arity_cartesian_product,
   natural_join_expression,
   projection_expression,
   select_expression,
   union_table_expression,
   table_variable);

FROM ISO13584_variable_semantics_schema
   (self_class_code_semantics,
   self_class_preferred_name_semantics,
   self_class_short_name_semantics,
   self_class_supplier_code_semantics,
   self_class_version_semantics,
   self_property_class_code_semantics,
```



```
    self_property_class_version_semantics,
    self_property_code_semantics,
    self_property_preferred_name_semantics,
    self_property_short_name_semantics,
    self_property_version_semantics);

FROM ISO13584_method_schema
    (set_2d_relative_view_level,
    referenced_sub_item_view_statement);

FROM representation_schema
    (representation,
    representation_context,
    representation_item);

FROM geometry_schema
    (axis1_placement,
    axis2_placement_2D,
    axis2_placement_3D,
    direction,
    geometric_representation_context,
    placement,
    point);
```

17.3.12 Conformance class 6E

Conformance class 6E addresses those implementations that support conformance class 6 and may support some supplier-defined external file protocol.

Besides the entities and related constructs defined for conformance class 6, they shall also support the following entities and related constructs;

```
FROM ISO13584_extended_dictionary_schema
    (dictionary);

FROM ISO13584_library_content_schema
    (library);

FROM ISO13584_external_file_schema
    (compiler_version_type,
    non_standard_data_protocol,
    non_standard_simple_program_protocol);
```

18 Exchange of functional view classes: library integrated information model 24-3

Library integrated information model 24-3 enables the exchange of the dictionary elements that describe functional view classes. Note that the standard functional view classes are intended to be defined as standard data by the view exchange protocol series of parts of ISO 13584 and to be recognised by the implementations that support the view exchange protocol where the functional view class is defined. The exchange of functional view classes is therefore only needed for those implementations that want to support new supplier-defined functional view classes.

Conformance to the library integrated information model 24-3 includes satisfying the requirements stated in the **ISO13584_f_v_iim_schema** presented in annex K, the requirements of the implementation method(s) supported, the relevant requirements of the normative references and the support of the standard data defined in annex M.

An implementation shall support the ISO10303-21 implementation methods. Requirement with respect to these implementation methods are specified in annex N.

The **ISO13584_f_v_iim_schema** provides for a number of options that may be supported by an implementation. These options have been grouped into conformance classes. Four conformance classes are defined. Conformance to the library integrated information model 24-3 requires, at a minimum, conformance to class 1. Options are defined by each class and may be selected by an implementation. Conformance to a particular conformance class requires that all the **ISO13584_f_v_iim_schema** entities, types and associated constraints defined as part of the class, be supported, together with the standard data associated with the class.

The numbering schema of the conformance classes is as follows:

— the digit specifies the dictionary elements that shall be supported;

NOTE 1 An ISO 13584-conformant library shall not contain any **content_item** for functional view classes.

— the conformance class identifier may contain the letter E. The corresponding conformance classes are called extended conformance classes. These conformance classes supports descriptions that use supplier-defined external file protocols. Implementation conformant with an extended conformance class shall be able to process any entity belonging to the set of supported entities, even when they reference a external file protocol that is not part of the standard data but that is value-equal to a external file protocol they claim to support.

NOTE 2 The attribute values for the **external_file_protocol** entities that do not belong to the standard data defined in this part of ISO 13584 (annexes E, I and M) or to the standard data defined in one part of the view exchange protocol series of part of ISO 13584 are subject to prior agreement between the sender and the receiver. They are outside the scope of ISO 13584.

Conformance class are characterised as follows:

— class 1: dictionary elements for non instanciable functional views,

— class 1E: dictionary elements for non instanciable functional views with supplier-defined external file protocols,

— class 2: dictionary elements for functional views with view attributes,

— class 2E: dictionary elements for functional views with view attributes and supplier-defined external file protocols.

18.1 ISO13584_f_v_iim_schema short listing

This clause specifies the EXPRESS schema that uses elements either from the integrated resources of ISO 10303 (ISO 10303-4x) or from the logical resource or description methodology series of parts of ISO 13584 (ISO 13584-2x and ISO 13584-4x) to define the requirements of library integrated information model LIIM24-1.

Requirements stated in the referenced schemas apply exclusively to those items that are used from these schemas.

The expanded EXPRESS listing of this schema is presented in annex K.

NOTE 1 Some of the entities and related constructs USEd in this schema do not belong to any of the conformance classes defined in this part of ISO 13584. They are intended to be possibly allowed by some part of the view exchange protocol series of part of ISO 13584.

NOTE 2 This schema is the information model of supplier libraries. The information model of integrated libraries is outside the scope of ISO 13584.

EXPRESS specification:

```
*)
SCHEMA ISO13584_f_v_iim_schema;

USE FROM ISO13584_IEC61360_dictionary_schema
      (axis1_placement_type,
      axis2_placement_2d_type,
      axis2_placement_3d_type,
      boolean_type,
      class_BSU,
      class_instance_type,
      data_type_BSU,
      data_type_element,
      dates,
      dic_unit,
      dic_value,
      entity_instance_type,
      identified_document,
      integer_type,
      int_currency_type,
      int_measure_type,
      int_type,
      item_names,
      level_type,
      label_with_language,
      mathematical_string,
      named_type,
      non_quantitative_code_type,
      non_quantitative_int_type,
      non_si_unit,
      number_type,
      placement_type,
      property_BSU,
      real_currency_type,
      real_measure_type,
      real_type,
      string_type,
      supplier_BSU,
      supplier_element,
      value_code_type,
      value_domain);

USE FROM ISO13584_IEC61360_language_resource_schema
      (global_language_assignment,
      present_translations,
      translated_label,
      translated_text);
```



```
USE FROM ISO13584_extended_dictionary_schema
      (table_BSU,
      class_document_relationship,
      dictionary_identification,
      dictionary,
      dictionary_in_standard_format,
      document_BSU,
      document_element,
      functional_view_class,
      geometric_representation_context_type,
      library_iim_identification,
      non_instantiable_functional_view_class,
      representation_P_DET,
      representation_type,
      view_exchange_protocol_identification);

USE FROM ISO13584_external_file_schema
      (document_content,
      external_file_unit,
      http_protocol,
      http_file,
      http_class_directory,
      language_specific_content,
      non_standard_protocol,
      not_translatable_external_content,
      not_translated_external_content,
      standard_data_protocol,
      translated_external_content);

USE FROM measure_schema
      (amount_of_substance_measure,
      amount_of_substance_unit,
      area_measure,
      area_unit,
      context_dependent_measure,
      context_dependent_unit,
      conversion_based_unit,
      count_measure,
      derived_unit,
      derived_unit_element,
      descriptive_measure,
      dimensional_exponents,
      electric_current_measure,
      electric_current_unit,
      length_measure,
      length_measure_with_unit,
      length_unit,
      luminous_intensity_measure,
      luminous_intensity_unit,
      mass_measure,
      mass_unit,
```

```
      measure_value,
      measure_with_unit,
      named_unit,
      numeric_measure,
      parameter_value,
      plane_angle_measure,
      plane_angle_unit,
      positive_length_measure,
      positive_plane_angle_measure,
      positive_ratio_measure,
      ratio_measure,
      ratio_unit,
      si_unit,
      solid_angle_measure,
      solid_angle_unit,
      thermodynamic_temperature_measure,
      thermodynamic_temperature_unit,
      time_measure,
      time_unit,
      volume_measure,
      volume_unit);

  USE FROM person_organization_schema
      (address,
      organization,
      person,
      person_and_organization,
      personal_address,
      organizational_address );

  USE FROM date_time_schema
      (date,
      date_and_time,
      local_time,
      calendar_date,
      ordinal_date,
      week_of_year_and_day_date);


  USE FROM application_context_schema
      (application_context,
      application_context_element,
      application_protocol_definition);

  USE FROM representation_schema
      (representation,
      representation_context,
      representation_item);

  (*
```


18.2 ISO13584_f_v_iim_schema global rule definitions

The following rules define the requirements for the **ISO13584_f_v_iim_schema** schema.

18.2.1 Exactly_one_dictionary_rule rule

The **exactly_one_dictionary_rule** rule states that a library exchange context is associated to exactly one **dictionary**.

EXPRESS specification:

```
*)
RULE exactly_one_dictionary_rule FOR(dictionary);
WHERE
     WR1: SIZEOF(dictionary) = 1;
END_RULE; -- exactly_one_library_rule
(*
```

Formal propositions:

WR1: there is exactly one **dictionary** defined in a library exchange context.

18.2.2 Class_associated_items_rule rule

The **class_associated_items_rule** rule states that each **class** has no more than two **associated_items** that are **class_table_relationship** and/or **class_document_relationship**.

EXPRESS specification:

```
*)
RULE class_associated_items_rule FOR(class);
WHERE
     WR1: QUERY(temp <* class | (SIZEOF(temp.associated_items) > 2)
          OR ((SIZEOF(temp.associated_items) = 1)
          AND NOT(('ISO13584_F_V_IIM_SCHEMA'
          +'.CLASS_TABLE_RELATIONSHIP' IN
          TYPEOF(temp.associated_items[1]))
          OR ('ISO13584_F_V_IIM_SCHEMA'
          +'.CLASS_DOCUMENT_RELATIONSHIP'
          IN TYPEOF(temp.associated_items[1]))))
          OR ((SIZEOF(temp.associated_items) = 2)
          AND NOT((('ISO13584_F_V_IIM_SCHEMA'
          + '.CLASS_TABLE_RELATIONSHIP'
          IN TYPEOF(temp.associated_items[1]))
          AND ('ISO13584_F_V_IIM_SCHEMA'
          +'.CLASS_DOCUMENT_RELATIONSHIP'
          IN TYPEOF(temp.associated_items[2])))
          OR (('ISO13584_F_V_IIM_SCHEMA'
          +'.CLASS_TABLE_RELATIONSHIP'
          IN TYPEOF(temp.associated_items[2]))
          AND ('ISO13584_F_V_IIM_SCHEMA'
          + '.CLASS_DOCUMENT_RELATIONSHIP'
          IN TYPEOF(temp.associated_items[1])))))))
```

```
          = [];
END_RULE; -- class_associated_items_rule
(*
```

Formal propositions:

WR1: each **class** has no more than two **associated_items** that are **class_table_relationship** and/or **class_document_relationship**.

```
*)

END_SCHEMA; -- ISO13584_f_v_iim_schema
(*
```

18.3 Conformance class requirements

18.3.1 Conformance class 1

Conformance class 1 addresses those implementations that support dictionary elements of supplier-defined **non_instantiable_functional_view_class**. An implementation of conformance class 1 of the library integrated information model 24-3 shall support the standard data defined in annex M. It shall also support the following entities and related constructs.

```
FROM ISO13584_IEC61360_dictionary_schema
   (class_BSU,
   dates,
   dic_value,
   identified_document,
   item_names,
   label_with_language,
   mathematical_string,
   non_quantitative_int_type,
   property_BSU,
   supplier_BSU,
   supplier_element,
   syn_name_type,
   value_domain);

FROM ISO13584_IEC61360_language_resource_schema
   (global_language_assignment,
   present_translations,
   translated_label,
   translated_text);

FROM ISO13584_extended_dictionary_schema
   (class_document_relationship,
   dictionary_identification,
   dictionary_in_standard_format,
   document_BSU,
   document_element,
   document_element_with_http_access,
   document_element_with_translated_http_access,
```



```
    library_iim_identification,
    non_instantiable_functional_view_class,
    referenced_document,
    referenced_graphics,
    representation_P_DET);

  FROM ISO13584_external_file_schema
    (document_content,
    external_file_unit,
    http_class_directory,
    http_file,
    http_protocol,
    language_specific_content,
    not_translatable_external_content,
    not_translated_external_content,
    standard_data_protocol,
    translated_external_content,
    property_value_external_item);

  FROM measure_schema
    (amount_of_substance_unit,
    area_unit,
    context_dependent_unit,
    conversion_based_unit,
    derived_unit,
    derived_unit_element,
    dimensional_exponents,
    electric_current_unit,
    length_measure_with_unit,
    length_unit,
    luminous_intensity_unit,
    mass_unit,
    measure_value,
    named_unit,
    plane_angle_unit,
    ratio_unit,
    si_unit,
    solid_angle_unit,
    thermodynamic_temperature_unit,
    time_unit,
    volume_unit);

  FROM person_organization_schema
    (address,
    organization,
    person,
    person_and_organization,
    personal_address,
    organizational_address );

  FROM date_time_schema
    (date,
```

```
    date_and_time,
    local_time,
    calendar_date,
    ordinal_date,
    week_of_year_and_day_date);

  FROM application_context_schema
    (application_context,
    application_context_element,
    application_protocol_definition);
```

18.3.2 Conformance class 1E

Conformance class 1E addresses those implementations that support conformance class 1 and may support some supplier-defined external file protocols.

Besides the entities and related constructs defined for conformance class 1, they shall also support the following entities and related constructs.

```
  FROM ISO13584_extended_dictionary_schema
    (dictionary);

  FROM ISO13584_external_file_schema
    (non_standard_data_protocol);
```

18.3.3 Conformance class 2

Conformance class 2 addresses those implementations that support dictionary elements of supplier-defined functional view classes that may contain view attributes. An implementation of conformance class 2 of the library integrated information model 24-3 shall support the standard data defined in annex M. It shall also support the entities and related constructs defined for conformance class 1 plus the following entities and related constructs.

```
  FROM ISO13584_IEC61360_dictionary_schema
    (axis1_placement_type,
    axis2_placement_2d_type,
    axis2_placement_3d_type,
    boolean_type,
    currency_code,
    data_type_BSU,
    data_type_element,
    dic_unit,
    int_currency_type,
    int_measure_type,
    int_type,
    named_type,
    non_quantitative_code_type,
    non_si_unit,
    number_type,
    placement_type,
    real_currency_type,
    real_measure_type,
    real_type,
```



```
    string_type);

  FROM ISO13584_extended_dictionary_schema
    (functional_view_class,
    representation_type);
```

18.3.4 Conformance class 2E

Conformance class 2E addresses those implementations that support conformance class 2 and may support some supplier-defined external file protocols.

Besides the entities and related constructs defined for conformance class 2, they shall also support the following entities and related constructs.

```
  FROM ISO13584_extended_dictionary_schema
    (dictionary);

  FROM ISO13584_external_file_schema
    (non_standard_data_protocol);
```

Annex A
(normative)

Short names of entities defined in this part

Table A.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 A.1 - Short names of entities

Long Name	Short Name
A_POSTERIORI_SEMANTIC_RELATIONSHIP	APS0
A_POSTERIORI_VIEW_OF	APVO
A_PRIORI_SEMANTIC_RELATIONSHIP	APS1
A6_ILLUSTRATION	A6ILL
A9_ILLUSTRATION	A9ILL
ABSTRACT_FUNCTIONAL_MODEL_CLASS	AFMC
ASSIGNMENT_STATEMENT	ASSSTT
BEGIN_SET	BGNST
BINARY_CLASS_INSTANCE_CONSTRUCTOR	BCIC
BINARY_CLASS_INSTANCE_EXPRESSION	BCIE
BINARY_TABLE_EXPRESSION	BNTBEX
BOOLEAN_COLUMN	BLNCLM
CALL_PROGRAM_STATEMENT	CLPRST
CLASS_BSU_RELATED_CONTENT	CBRC
CLASS_DOCUMENT_RELATIONSHIP	CLDCRL
CLASS_EXTENSION	CLSEXT
CLASS_EXTENSION_EXTERNAL_ITEM	CEEI
CLASS_INSTANCE_COLUMN	CLINCL
CLASS_INSTANCE_CONSTRUCTOR	CLINCN
CLASS_INSTANCE_EXPRESSION	CLINEX
CLASS_INSTANCE_LITERAL	CLINLT
CLASS_INSTANCE_VARIABLE	CLINVR
CLASS_RELATED_DICTIONARY_ELEMENT	CRDE
CLASS_TABLE_RELATIONSHIP	CLTBRL
CLOSE_SET	CLSST
COLUMN	COLUMN
COLUMN_TRAVERSAL_VARIABLE_SEMANTICS	CTVS
COMPLEX_COLUMN	CMPCLM
COMPONENT_CLASS_CASE_OF	CCCO
CONSTANT_RANGE_DEFINED_DOMAIN	CRDD
CONSTRUCTED_SUB_MODEL_VIEW_STATEMENT	CSMVS

Table A.1 (continued)

Long Name	Short Name
CONTEXT_DEPENDENT_PROPERTY_VALUE	CDPV
DATA_EXCHANGE_SPECIFICATION_IDENTIFICATION	DES0
DATA_PROTOCOL	DTPRT
DIALOGUE_RESOURCE	DLGRSR
DIC_CLASS_INSTANCE	DCCLIN
DIC_COMPONENT_INSTANCE	DCCMIN
DIC_F_MODEL_INSTANCE	DFMI
DIC_F_VIEW_INSTANCE	DFVI
DIC_FEATURE_INSTANCE	DCFTIN
DIC_ITEM_INSTANCE	DCITIN
DIC_MATERIAL_INSTANCE	DCMTIN
DICTIONARY	DCTNRY
DICTIONARY_EXTERNAL_ITEM	DCEXIT
DICTIONARY_IDENTIFICATION	DCTIDN
DICTIONARY_IN_STANDARD_FORMAT	DISF
DIFFERENCE_TABLE_EXPRESSION	DFTBEX
DOCUMENT_BSU	DCMBS
DOCUMENT_CONTENT	DCMCNT
DOCUMENT_ELEMENT	DCMELM
DOCUMENT_ELEMENT_WITH_HTTP_ACCESS	DEWHA
DOCUMENT_ELEMENT_WITH_TRANSLATED_HTTP_ACCESS	DEW0
DOMAIN_RESTRICTION	DMNRST
ENTITY_INSTANCE_COLUMN	ENINCL
ENTITY_INSTANCE_EXPRESSION	ENINEX
ENTITY_INSTANCE_LITERAL	ENINLT
ENTITY_INSTANCE_VARIABLE	ENINVR
EXISTS_VALUE	EXSVL
EXPLICIT_FUNCTIONAL_MODEL_CLASS_EXTENSION	EFMCE
EXPLICIT_ITEM_CLASS_EXTENSION	EICE
EXPLICIT_MODEL_CLASS_EXTENSION	EMCE
EXTERNAL_CONTENT	EXTCNT
EXTERNAL_FILE_PROTOCOL	EXFLPR
EXTERNAL_FILE_UNIT	EXFLUN
EXTERNAL_ITEM	EXTITM
FEATURE_CLASS	FTRCLS
FEATURE_CLASS_CASE_OF	FCCO
FM_CLASS_VIEW_OF	FCVO
FORMATTED_COLUMN	FRMCLM

Table A.1 (continued)

Long Name	Short Name
FUNCTIONAL_DOMAIN_RESTRICTION	FNDMRS
FUNCTIONAL_MODEL_CLASS	FNMDCL
FUNCTIONAL_MODEL_CLASS_EXTENSION	FMCE
FUNCTIONAL_VIEW_CLASS	FNVWCL
GEOMETRIC_REPRESENTATION_CONTEXT_TYPE	GRCT
GUARDED_FUNCTIONAL_DOMAIN	GRFNDM
GUARDED_SIMPLE_DOMAIN	GRSMDM
GUARDED_STATEMENT	GRDSTT
HTTP_CLASS_DIRECTORY	HTCLDR
HTTP_FILE	HTTFL
HTTP_PROTOCOL	HTTPRT
ILLUSTRATION	ILLSTR
IMPLICIT_MODEL_CLASS_EXTENSION	IMCE
IN_RDB_TABLE_BOOLEAN_EXPRESSION	IRTBE
INSTANCE_COMPARISON_EQUAL	INCMEQ
INT_LEVEL_SPEC_COLUMN	ILSC
INT_LEVEL_SPEC_LITERAL	ILSL
INT_LEVEL_SPEC_VALUE	ILS0
INT_LEVEL_SPEC_VARIABLE	ILSV
INTEGER_COLUMN	INTCLM
INTERSECT_TABLE_EXPRESSION	INTBEX
ITEM_CLASS_CASE_OF	ICCO
ITEM_CLASS_EXTENSION	ITCLEX
LANGUAGE_SPECIFIC_CONTENT	LNSPCN
LEVEL_SPEC_COLUMN	LVSPCL
LEVEL_SPEC_EXPRESSION	LVSPEX
LEVEL_SPEC_LITERAL	LVSPLT
LEVEL_SPEC_VALUE	LVSPVL
LEVEL_SPEC_VARIABLE	LVSPVR
LIB_COMPONENT_INSTANCE	LBCMIN
LIB_F_MODEL_INSTANCE	LFMI
LIB_FEATURE_INSTANCE	LBFTIN
LIB_ITEM_INSTANCE	LBITIN
LIB_MATERIAL_INSTANCE	LBMTIN
LIBRARY	LBRRY
LIBRARY_EXPRESSION_DEFINED_VALUE	LEDV
LIBRARY_IIM_IDENTIFICATION	LBI0

Table A.1 (continued)

Long Name	Short Name
LIBRARY_IN_STANDARD_FORMAT	LISF
LINKED_INTERFACE_PROGRAM_PROTOCOL	LIPP
MATERIAL_CLASS_CASE_OF	MCCO
MESSAGE	MSSG
METHOD	METHOD
METHOD_BODY	MTHBDY
METHOD_SPECIF	MTHSPC
METHOD_STATEMENT	MTHSTT
MODEL_CLASS_EXTENSION	MDCLEX
MODELLING_STATEMENT	MDLSTT
MULTIPLE_ARITY_CARTESIAN_PRODUCT	MACP
MULTIPLE_ARITY_CLASS_INSTANCE_CONSTRUCTOR	MACIC
MULTIPLE_ARITY_CLASS_INSTANCE_EXPRESSION	MACIE
MULTIPLE_ARITY_TABLE_EXPRESSION	MATE
NATURAL_JOIN_EXPRESSION	NTJNEX
NON_INSTANTIABLE_FUNCTIONAL_VIEW_CLASS	NIFVC
NON_STANDARD_DATA_PROTOCOL	NSDP
NON_STANDARD_PROTOCOL	NNSTPR
NON_STANDARD_SIMPLE_PROGRAM_PROTOCOL	NSSPP
NOT_TRANSLATABLE_EXTERNAL_CONTENT	NTEC
NOT_TRANSLATED_EXTERNAL_CONTENT	NTE0
NULL_DEFINED_VALUE	NLDFVL
NULL_STATEMENT	NLLSTT
NULL_VALUE	NLLVL
NUMBER_COLUMN	NMBCLM
OPEN_VIEW_PROPERTY_SEMANTICS	OVPS
OPEN_VIEW_PROPERTY_VALUE_SEMANTICS	OVPVS
OPEN_VIEW_VARIABLE_SEMANTICS	OVVS
OPT_OR_MAND_PROPERTY_BSU	OOMPB
OTHERS	OTHERS
PREDEFINED_REPRESENTATION_CALL_STATEMENT	PRCS
PREDICATE_DEFINED_DOMAIN	PRDFDM
PROGRAM_LIBRARY_BSU	PRLBBS
PROGRAM_LIBRARY_CONTENT	PRLBCN
PROGRAM_LIBRARY_ELEMENT	PRLBEL
PROGRAM_PROTOCOL	PRGPRT
PROGRAM_REFERENCE	PRGRFR

Table A.1 (continued)

Long Name	Short Name
PROGRAM_REFERENCE_TYPE	PRRFTY
PROJECTION_EXPRESSION	PRJEXP
PROPERTY_ASSIGNMENT	PRPASS
PROPERTY_CLASSIFICATION	PRPCLS
PROPERTY_SEMANTICS	PRPSMN
PROPERTY_VALUE	PRPVL
PROPERTY_VALUE_EXTERNAL_ITEM	PVEI
PROPERTY_VALUE_RECOMMENDED_PRESENTATION	PVRP
RDB_TABLE_CONTENT	RDTBCN
RDB_TABLE_ELEMENT	RDTBEL
RDB_TABLE_EXTENSION	RDTBEX
RDB_TABLE_SPECIFICATION	RDTBSP
RDB_TABLE_VARIABLE	RDTBVR
REAL_COLUMN	RLCLM
REAL_LEVEL_SPEC_COLUMN	RLSC
REAL_LEVEL_SPEC_LITERAL	RLSL
REAL_LEVEL_SPEC_VALUE	RLS0
REAL_LEVEL_SPEC_VARIABLE	RLSV
REFERENCED_DOCUMENT	RFRDCM
REFERENCED_GRAPHICS	RFRGRP
REFERENCED_SUB_ITEM_VIEW_STATEMENT	RSIVS
REPRESENTATION_P_DET	RPPDT
REPRESENTATION_REFERENCE	RPRRFR
REPRESENTATION_REFERENCE_TYPE	RPRFTY
REPRESENTATION_TYPE	RPRTYP
SELECT_EXPRESSION	SLCEXP
SELF_CLASS_CODE_SEMANTICS	SCCS
SELF_CLASS_NAME_SEMANTICS	SCNS
SELF_CLASS_PREFERRED_NAME_SEMANTICS	SCPNS
SELF_CLASS_SHORT_NAME_SEMANTICS	SCSNS
SELF_CLASS_SUPPLIER_CODE_SEMANTICS	SCSCS
SELF_CLASS_VARIABLE_SEMANTICS	SCV0
SELF_CLASS_VERSION_SEMANTICS	SCVS
SELF_PROPERTY_CLASS_CODE_SEMANTICS	SPCCS
SELF_PROPERTY_CLASS_SUPPLIER_CODE_SEMANTICS	SPCSCS
SELF_PROPERTY_CLASS_VERSION_SEMANTICS	SPCVS
SELF_PROPERTY_CODE_SEMANTICS	SPCS
SELF_PROPERTY_NAME_SEMANTICS	SPNS

Table A.1 (continued)

Long Name	Short Name
SELF_PROPERTY_PREFERRED_NAME_SEMANTICS	SPPNS
SELF_PROPERTY_SEMANTICS	SLPRSM
SELF_PROPERTY_SHORT_NAME_SEMANTICS	SPSNS
SELF_PROPERTY_VALUE_SEMANTICS	SPV0
SELF_PROPERTY_VERSION_SEMANTICS	SPVS
SELF_VARIABLE_SEMANTICS	SLVRSM
SEND_REPRESENTATION_REFERENCE_STATEMENT	SRRS
SEND_REPRESENTATION_STATEMENT	SNRPST
SET_2D_RELATIVE_VIEW_LEVEL	S2RVL
SET_REFERENCE_LCS	STRFLC
SET_TABLE_EXPRESSION	STTBEX
SIMPLE_CLASS_INSTANCE_EXPRESSION	SCIE
SIMPLE_COLUMN	SMPCLM
SIMPLE_DOMAIN	SMPDMN
SIMPLE_ENTITY_INSTANCE_EXPRESSION	SEIE
SIMPLE_FUNCTIONAL_DOMAIN	SMFNDM
SIMPLE_LEVEL_SPEC_EXPRESSION	SLSE
SIMPLE_PROGRAM_PROTOCOL	SMPRPR
SIMPLE_STATEMENT	SMPSTT
SIMPLE_TABLE_EXPRESSION	SMTBEX
STANDARD_DATA_PROTOCOL	STDTPR
STANDARD_PROTOCOL	STNPRT
STANDARD_SIMPLE_PROGRAM_PROTOCOL	SSPP
STRING_COLUMN	STRCLM
SUB_OBJECT_VIEW_STATEMENT	SOVS
SUB_PROPERTY_PATH	SBPRPT
SUBCLASS_DEFINED_DOMAIN	SBDFDM
SUPPLIER_BSU_RELATED_CONTENT	SBRC
SUPPLIER_PROGRAM_LIBRARY_RELATIONSHIP	SPLR
SUPPLIER_RELATED_DICTIONARY_ELEMENT	SRDE
TABLE_BSU	TBLBS
TABLE_CONTENT	TBLCNT
TABLE_DEFINED_DOMAIN	TBDFDM
TABLE_DEFINED_VALUE	TBDFVL
TABLE_ELEMENT	TBLELM
TABLE_EXPRESSION	TBLEXP

Table A.1 (continued)

Long Name	Short Name
TABLE_EXTENSION	TBLEXT
TABLE_IDENTIFICATION	TBLIDN
TABLE_LITERAL	TBLLTR
TABLE_SPECIFICATION	TBLSPC
TABLE_VARIABLE	TBLVRB
TRANSLATED_EXTERNAL_CONTENT	TREXCN
TRANSLATED_STRING_VALUE	TRSTVL
TYPE_DEFINED_DOMAIN	TYDFDM
UNARY_CLASS_INSTANCE_CONSTRUCTOR	UCIC
UNARY_CLASS_INSTANCE_EXPRESSION	UCIE
UNARY_TABLE_EXPRESSION	UNTBEX
UNCONTROLLED_ENTITY_INSTANCE_VALUE	UEIV
UNION_TABLE_EXPRESSION	UNT0
VARIABLE_RANGE_DEFINED_DOMAIN	VRDD
VIEW_CONTROL_VARIABLE_RANGE	VCVR
VIEW_EXCHANGE_PROTOCOL_IDENTIFICATION	VEP0

Annex B
(normative)

Information object registration

B.1 Document identification

In order to provide for unambiguous identification of an information object in an open system, the object identifier:

{ iso standard 13584 part (24) version(1) }

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 13584-1.

B.2 Schema identification

B.2.1 ISO13584_instance_resource_schema

The **ISO13584_instance_resource_schema** (see clause 7) is assigned the object identifier:

{ iso standard 13584 part (24) version(1) object(1) ISO13584-instance-resource-schema(1) }

B.2.2 ISO13584_library_expressions_schema

The **ISO13584_library_expressions_schema** (see clause 8) is assigned the object identifier:

{ iso standard 13584 part (24) version(1) object(1) ISO13584-library-expressions-schema(2) }

B.2.3 ISO13584_table_resource_schema

The **ISO13584_table_resource_schema** (see clause 9) is assigned the object identifier:

{ iso standard 13584 part (24) version(1) object(1) ISO13584-table-resource-schema(3) }

B.2.4 ISO13584_variable_semantics_schema

The **ISO13584_variable_semantics_schema** (see clause 10) is assigned the object identifier:

{ iso standard 13584 part (24) version(1) object(1) ISO13584-variable-semantics-schema(4) }

B.2.5 ISO13584_domain_resource_schema

The **ISO13584_domain_resource_schema** (see clause 11) is assigned the object identifier:

{ iso standard 13584 part (24) version(1) object(1) ISO13584-domain-resource-schema(5) }

B.2.6 ISO13584_extended_dictionary_schema

The **ISO13584_extended_dictionary_schema** (see clause 12) is assigned the object identifier:

{ iso standard 13584 part (24) version(1) object(1) ISO13584-extended-dictionary-schema(6) }

B.2.7 ISO13584_library_content_schema

The **ISO13584_library_content_schema** (see clause 13) is assigned the object identifier:

{ iso standard 13584 part (24) version(1) object(1) ISO13584-library-content-schema(7) }

B.2.8 ISO13584_external_file_schema

The **ISO13584_external_file_schema** (see clause 14) is assigned the object identifier:

{ iso standard 13584 part (24) version(1) object(1) ISO13584-external-file-schema(8) }

B.2.9 ISO13584_method_schema

The **ISO13584_method_schema** (see clause 15) is assigned the object identifier:

{ iso standard 13584 part (24) version(1) object(1) ISO13584-method-schema(9) }

B.2.10 ISO13584_g_m_iim_schema

The **ISO13584_g_m_iim_schema** (see clause 18) is assigned the object identifier:

{ iso standard 13584 part (24) version(1) object(1) ISO13584_g_m_iim_schema (10) }

B.2.11 ISO13584_f_m_iim_schema

The **ISO13584_f_m_iim_schema** (see clause 19) is assigned the object identifier:

{ iso standard 13584 part (24) version(1) object(1) ISO13584_f_m_iim_schema (11) }

B.2.12 ISO13584_f_v_iim_schema

The **ISO13584_f_v_iim_schema** (see clause 20) is assigned the object identifier:

{ iso standard 13584 part (24) version(1) object(1) ISO13584_f_v_iim_schema (12) }

Annex C
(normative)

ISO13584_g_m_iim_library_implicit_schema expanded listing

This annex references a listing of the complete EXPRESS schema specified in clause 16 of this part of ISO 13584 without comments or other explanatory text but with the constraints defined in the **ISO13584_g_m_iim_conformance_schema**. The name of this schema is **ISO13584_g_m_iim_library_implicit_schema**. In this listing, all the elements used either from the integrated resources of ISO 10303 (ISO 10303-4x) or from the logical resource or description methodology series of parts of ISO 13584 (ISO 13584-2x and ISO 13584-4x) by the **ISO13584_g_m_iim_schema** and the constraints defined in the **ISO13584_g_m_iim_conformance_schema** are gathered in an unique schema without any external reference.

This schema may be used:

— to exchange libraries that reference the **ISO13584_g_m_iim_schema** and its associated **ISO13584_g_m_iim_conformance_schema**, but that does not reference any view exchange protocol, and

— to exchange libraries that reference the **ISO13584_g_m_iim_schema** and its associated **ISO13584_g_m_iim_conformance_schema**, and that do reference to some view exchange protocols; in this case, the constraints defined in these view exchange protocols are not checked.

This schema may also be completed to check the constraints defined in all the referenced view exchange protocols using the following process for each referenced view exchange protocol.

Assume that V1 is a referenced view exchange protocols and that it specifies two constraint schemas, the schema names of which are S1_V1 and S2_V1.

a) Build the long form of the S1_V1 schema and give to the resulting schema the same name: "S1_V1";

b) Build the long form of the S2_V1 schema and give to the resulting schema the same name: "S2_V1";

c) Replace everywhere in the long form of the S1_V1 schema, the string "S1_V1" by "ISO13584_g_m_iim_library_implicit_schema" with the same case;

d) Replace everywhere in the long form of the S2_V1 schema, the string "S2_V1" by "ISO13584_g_m_iim_library_implicit_schema" with the same case;

e) Check that all the entities referenced in the S1_V1 schema and in the S2_V1 schema are already existing in the **ISO13584_g_m_iim_library_implicit_schema**, else reference to the library integrated information model 24-1 and to the view exchange protocol S1 by a same library delivery file is not allowed.

NOTE 1 The information model of a library delivery file and the entities it may contain are specified by a library integrated information model. A view exchange protocol may only add constraints.

EXAMPLE The view exchange protocol defined in ISO 13584-102 references the **abstract_functional_model_class** entity. Therefore, this view exchange protocol cannot be used with the **ISO13584_g_m_iim_library_implicit_schema** that specifies the requirement of LIIM 24-1. This library integrated information model does not reference any EXPRESS resource constructs for modelling functional models.

f) Add the content of the long form of the S1_V1 schema to the content of the **ISO13584_g_m_iim_library_implicit_schema**, removing possible duplicates;

g) Add the content of the long form of the S2_V1 schema to the content of the **ISO13584_g_m_iim_library_implicit_schema**, removing possible duplicates.

When the above process is performed for view exchange protocols V1, V2,...Vn, the resulting **ISO13584_g_m_iim_library_implicit_schema** may be used for exchanging any library that references the **ISO13584_g_m_iim_schema** and its associated **ISO13584_g_m_iim_conformance_schema** as its library integrated information model, and that references whole or part of the V1, V2,...Vn view exchange protocol set. This schema also includes the constraints of all the referenced view exchange protocols.

The listing of the **ISO13584_g_m_iim_library_implicit_schema** schema is available in computer-interpretable form and can be found at the following URL:

http://www.tc184-sc4.org/EXPRESS/

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

NOTE 2 The information provided in computer-interpretable form at the above URLs is normative.

NOTE 3 In some errors are identified in the EXPRESS code during implementation and before publication of Technical Corrigendum, the description of these errors, together with the corrections recommended for PLIB implementations by the part editors can be found at the following URL:

http://www.lisi.ensma.fr/ftp/pub/PLIB_release_notes/Part24/Part24-IS/

Annex D
(normative)

ISO13584_g_m_iim_schema short names of entities

This annex references a listing of the EXPRESS entity names and corresponding short names for the EXPRESS schema specified in annex C of this part of ISO 13584. This listing is available in computer-interpretable form and can be found at the following URL:

http://www.tc184-sc4.org/Short_Names/

NOTE The information provided in computer-interpretable form at the above URLs is informative. The information that is contained in the body of this part of ISO 10303 is normative.

Annex E
(normative)

Standard data requirements for the library integrated information model 24-1

Standard data are the entity instances that shall be recognised by any implementation conformant with ISO 13584 that claims conformance to some conformance class of some library integrated information model or view exchange protocol.

Standard data shall be specified by each library integrated information model and by each view exchange protocol, and for each of them, for each conformance class.

Standard data may include:

— instances of **basic_semantic_unit**s, associated with the corresponding **dictionary_element** and possibly **content_item**,

— instances of **external_file_protocol**s, and

— instance of other entities required to define the previous entities instances.

Recognition of a **basic_semantic_unit** means that a value-equal **basic_semantic_unit** is already stored in the user system, together with a corresponding **dictionary_element** and possibly a **content_item** as specified in the view exchange protocol or library integrated information model standard data. This implies that a reference to a value-equal **basic_semantic_unit** in a supplier library is interpreted as a reference to the pre-existing **basic_semantic_unit**.

NOTE Value equality of two entity instances means that all their corresponding attributes have the same values. Value-equality between two basic semantic units implies that they both identify the same concept.

EXAMPLE 1 Examples of **basic_semantic_units** that may be defined as standard data in a view exchange protocol include the **class_BSU** that identifies the functional view class possibly defined by the view exchange protocol and the **property_BSU** that identifies the view control variable of this functional view class.

Recognition of an external file protocol means that external files that reference a value equal **external_file_protocol** shall be processed by an implementation that recognises this **external_file_protocol**.

EXAMPLE 2 Example of an external file protocol that may be defined as standard data by a view exchange protocol or a library integrated information model is the ISO standard ISO 8859-1 that specifies a 8-bit, single-byte-coded graphics character set for Latin alphabet N°1.

Standard data are specified by means of a set of constraints that shall be fulfilled by any library that claims conformance to some conformance class of LIIM 24-1. The following standard data are specified by library integrated information model 24-1.

E.1 Constraints on a library delivery file for referencing library integrated information model 24-1

This subclause defines **library_iim_identification** instance values that are allowed for use in a library delivery file to reference library integrated information model 24-1 defined in this part of ISO 13584.

The set of allowed values is defined by means of Table E.1 that specifies for each conformance class the allowed values of **library_iim_identification.name** and **library_iim_identification.application**, and by means of one EXPRESS schema that contains a global rule. This rule shall be fulfilled by any library delivery file that references library integrated information model 24-1, defined in this part of ISO 13584 in any of its conformance class. The goal of this rule is to specify the allowed values for the

other attributes of **library_iim_identification** that shall be used to reference library integrated information model 24-1, by means of relationships with **library_iim_identification.name** and **library_iim_identification.application**.

This rule is included in the **ISO13584_g_m_iim_library_implicit_schema** specified in annex C.

E.2 Conformance class specification table

Table E.1 specifies the values of **library_iim_id.name** and **library_iim_id.application** that are allowed for use in a **library_iim_id** to reference library integrated information model 24-1 in either of its conformance classes.

Table E.1 — ISO 13584 LIIM 24-1 conformance class specification

Conformance Class	library_iim_identification.name mandatory value	library_iim_identification.application mandatory value
0	'ISO_13584_24_1'	'0'
1	'ISO_13584_24_1'	'1'
2	'ISO_13584_24_1'	'2'
3	'ISO_13584_24_1'	'3'
4	'ISO_13584_24_1'	'4'
5	'ISO_13584_24_1'	'5'
6	'ISO_13584_24_1'	'6'
1E	'ISO_13584_24_1'	'1E'
2E	'ISO_13584_24_1'	'2E'
3E	'ISO_13584_24_1'	'3E'
4E	'ISO_13584_24_1'	'4E'
5E	'ISO_13584_24_1'	'5E'
6E	'ISO_13584_24_1'	'6E'

E.3 Standard data for conformance class 0

None.

E.4 Standard data for conformance class 1 to 6E (all the conformance classes but conformance class 0)

E.4.1 Constraints on a library delivery file conform to the library integrated model LIIM 24-1

The **library_iim_identification** instance values allowed for use in a library delivery file conform to the library integrated model LIIM 24-1 defined in this part of ISO 13584 shall obey the constraints defined in the following EXPRESS schema.

EXPRESS specification:

```
*)
SCHEMA ISO13584_g_m_iim_conformance_schema;

USE FROM ISO13584_IEC61360_dictionary_schema(
      item_names);

USE FROM ISO13584_IEC61360_language_resource_schema(
      translated_label);

USE FROM person_organization_schema(
      organization);

USE FROM support_resource_schema(
      label);

USE FROM ISO13584_extended_dictionary_schema(
      data_exchange_specification_identification,
      library_iim_identification);

USE FROM ISO13584_external_file_schema(
      http_protocol,
      standard_data_protocol,
      external_file_protocol);
(*
```

NOTE The schemas used above can be found in the following document:

ISO13584_IEC61360_dictionary_schema	IEC 61360-2

(which is duplicated for convenience in informative annex D of ISO 13584-42),

person_organization_schema	ISO 10303-41,
ISO13584_extended_dictionary_schema	This part of ISO 13584,
ISO13584_external_file_schema	This part of ISO 13584.

E.4.2 Allowed_reference_to_LIIM_24_1_rule rule

The **allowed_reference_to_LIIM_24_1_rule** rule defines a formal constraint and an informal constraint on **library_iim_identification**s to be allowed for use to reference conformance class 1 to 6E of library integrated model LIIM 24-1 defined in this part of ISO 13584. A **library_iim _identification** is allowed for use to reference conformance class 1 to 6E of library integrated model LIIM 24-1 if the following conditions hold:

— the **name** attribute of the **library_iim_identification** that reference library integrated model LIIM 24-1 shall be equal to 'ISO_13584_24_1', and

— the **status** attribute of the **library_iim_identification** shall be equal to 'WD', 'CD' or 'DIS', 'FDIS', 'IS', 'TS', 'PAS' or 'ITA' and

— the **application** attribute of the **library_iim_identification** shall have ‘1', '2', '3', '4', '5' or '6' possibly followed by 'E' as its value, and

— if the conformance class of library integrated model LIIM 24-1referenced is not an extended conformance class, the **external_file_protocol**s referenced by the **external_file_protocols** attribute of the **library_iim_identification** shall fulfil the constraints required by the **conformant_external_file_protocol_24_1** function.

Moreover, a **library_iim_identification** is allowed for use to reference conformance class 1 to 6 of library integrated model LIIM 24-1 if one of the two following conditions hold concerning the **http_file**s that may be referenced directly or indirectly from the **library_iim_identification**,

— either each referenced **http_file** it is associated with a **mime** attribute and an **exchange_format** attribute corresponding to MIME type and subtype that correspond to a specification that is publicly available, or

— it is associated with a **mime** attribute and an **exchange_format** attribute corresponding to MIME type and subtype that correspond to a specification that is associated with public domain Internet-available readers.

Reference to **http_file**s corresponding to other MIME types and subtypes may only be done in extended conformance classes. This is documented as an informal proposition **IP1** in **allowed_reference_to_LIIM_24_1_rule**.

EXPRESS specification:

```
*)
RULE allowed_reference_to_LIIM_24_1_rule FOR(
      library_iim_identification);
WHERE
      WR1: QUERY(liim_id <* library_iim_identification |
           ((liim_id\data_exchange_specification_identification
                     .status = 'WD')
                OR
           (liim_id\data_exchange_specification_identification
                     .status = 'CD')
                OR
           (liim_id\data_exchange_specification_identification
                     .status = 'DIS')
                OR
           (liim_id\data_exchange_specification_identification
                     .status = 'FDIS')
                OR
           (liim_id\data_exchange_specification_identification
                     .status = 'IS')
                OR
           (liim_id\data_exchange_specification_identification
                     .status = 'TS')
                OR
           (liim_id\data_exchange_specification_identification
                     .status = 'PAS')
                OR
```



```
            (liim_id\data_exchange_specification_identification
                     .status = 'ITA')
            )
            AND
            (liim_id\data_exchange_specification_identification.name
            = 'ISO_13584_24_1')
            AND
            is_correct_liim_24_1_application_value(liim_id)
            AND
            (QUERY(efp <*
            liim_id\data_exchange_specification_identification
            .external_file_protocols
            | NOT(is_extended_liim_24_1_application_value(liim_id))
            AND
            NOT(conformant_external_file_protocol_24_1([efp]))
            ) = []))
            = QUERY(liim_id <* library_iim_identification |
            (liim_id\data_exchange_specification_identification
            .name = 'ISO_13584_24_1'));
    END_RULE; -- allowed_reference_to_LIIM_24_1_rule
    (*
```

Formal propositions:

WR1: when referencing library integrated model LIIM 24-1 defined in this part of ISO 13584, the **library_iim_identification.name** shall have ‘ISO_13584_24_1’ as its value, **library_iim_identification.status** shall be equal to 'WD', 'CD', 'DIS', 'FDIS', 'IS', 'TS', 'PAS' or 'ITA', the **library_iim_identification.application** shall have ‘1', '2', '3', '4', '5' or '6' as its value, possibly followed by 'E', and, if the conformance class of library integrated model LIIM 24-1referenced is not an extended conformance class, the **library_iim_identification.external_file_protocols** shall fulfil the constraints specifications required by the **conformant_external_file_protocol_24_1** function defined below.

Informal propositions:

IP1: when it references library integrated model LIIM 24-1 defined in this part of ISO 13584 in one of the conformance class 1, 2, 3, 4, 5 or 6, a **library_iim_identification** may only reference, directly or indirectly, **http_file**s characterised by MIME types and subtypes that either correspond to specifications that are publicly available, or to specifications that are associated with public domain Internet-available readers.

E.4.3 conformant_http_protocol_24_1 function

The **conformant_http_protocol_24_1** function checks whether an **external_file_protocol** may be referenced as the HTTP protocol by a **library_iim_identification** that references library integrated model LIIM 24-1 in any of its conformance classes, or not. It returns TRUE if the given **external_file_protocol** is allowed for reference, otherwise, it returns FALSE. An **external_file_protocol** may be referenced as the HTTP protocol by a **library_iim_identification** that reference library integrated model LIIM 24-1 in any of its conformance classes if the following conditions hold:

— the **external_file_protocol** shall be a **http_protocol**, and

— the **organisation** attribute of the **external_file_protocol** shall reference an organization of which the **id** attribute equals to 'IAB' and the **name** attribute equals to 'Internet Architecture Board', and

— the **protocol_name** attribute of the **external_file_protocol** shall equal to 'HTTP' or to ‘HTTPS’, and

— the **designation** attribute of the **external_file_protocol** shall reference an **item_names** for which the **preferred_name** attribute equals to 'Hypertext Transfer Protocol' and the **short_name** attribute equals to 'RFC' followed by four digits and possibly some other characters.

EXPRESS specification:

```
*)
FUNCTION conformant_http_protocol_24_1(
     ef: external_file_protocol): BOOLEAN;

LOCAL
     ok: BOOLEAN := TRUE;
END_LOCAL;

IF (('ISO13584_EXTERNAL_FILE_SCHEMA'
     + '.HTTP_PROTOCOL' IN TYPEOF(ef)) AND
     (ef.organisation.id = 'IAB') AND
     (ef.organisation.name = 'Internet Architecture Board') AND
     ((ef.protocol_name = 'HTTP')
          OR (ef.protocol_name = 'HTTPS'))AND
     (ef.designation.preferred_name = 'Hypertext Transfer Protocol'))
THEN
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.TRANSLATED_LABEL'
          IN TYPEOF(ef.designation.short_name))
     THEN
          REPEAT i := 1 TO SIZEOF(ef.designation.short_name.labels);
               IF (ef.designation.short_name.labels[i]
                    LIKE 'RFC####&')
               THEN
                    ok := ok AND TRUE;
               ELSE
                    ok := ok AND FALSE;
               END_IF;
          END_REPEAT;
          RETURN(OK);
     ELSE
          IF ef.designation.short_name
               LIKE 'RFC####&'
          THEN
               RETURN(TRUE);
          ELSE
               RETURN(FALSE);
          END_IF;
     END_IF;
ELSE
     RETURN(FALSE);
END_IF;
```



```
END_FUNCTION; -- conformant_http_protocol_24_1
(*
```

E.4.4 conformant_8859_1_protocol_24_1 function

The **conformant_8859_1_protocol_24_1** function checks whether an **external_file_protocol** may be referenced as the ISO 8859-1 protocol by a **library_iim_identification** that references library integrated model LIIM 24-1 in any of its conformance classes, or not. It returns TRUE if the given **external_file_protocol** is allowed for reference, otherwise, it returns FALSE. An **external_file_protocol** may be referenced as the ISO 8859-1 protocol by a **library_iim_identification** that represents reference library integrated model LIIM 24-1 in any of its conformance classes, if the following conditions hold:

— the **external_file_protocol** shall be a **standard_data_protocol**, and

— the **organisation** attribute of the **external_file_protocol** shall reference an organization of which the **id** attribute equals to 'ISO' and the **name** attribute equals to 'International Organisation for Standardisation', and

— the **protocol_name** attribute of the **external_file_protocol** shall equal to 'ISO_8859_1', and

— the **designation** attribute of the **external_file_protocol** shall reference an **item_names** for which the **preferred_name** attribute equals to 'Latin alphabet No 1' and the **short_name** attribute equals to 'ISO 8859-1'.

EXPRESS specification:

```
*)
FUNCTION conformant_8859_1_protocol_24_1(
     ef: external_file_protocol): BOOLEAN;

IF (('ISO13584_EXTERNAL_FILE_SCHEMA'
     + '.STANDARD_DATA_PROTOCOL' IN TYPEOF(ef)) AND
     (ef.organisation.id = 'ISO') AND
     (ef.organisation.name
          = 'International Organisation for Standardisation') AND
     (ef.protocol_name = 'ISO_8859_1') AND
     (ef.designation.preferred_name = 'Latin alphabet No 1') AND
     (ef.designation.short_name = 'ISO 8859-1'))
THEN
     RETURN(TRUE);
ELSE
     RETURN(FALSE);
END_IF;

END_FUNCTION; -- conformant_8859_1_protocol_24_1
(*
```

E.4.5 conformant_external_file_protocol_24_1 function

The **conformant_external_file_protocol_24_1** function checks whether all the **external_file_protocol**s of a set of **external_file_protocol**s may be referenced as an library

integrated model LIIM 24-1 by a **library_iim_identification** that references library integrated model LIIM 24-1 in one of its conformance class 1 to 6, or not. It returns TRUE if all the **external_file_protocol**s of a set of **external_file_protocol**s are allowed for reference, otherwise, it returns FALSE.

An **external_file_protocol** may be referenced by a **library_iim_identification** that represents conformance class 1 to 6 of library integrated model LIIM 24-1 if it may be referenced:

— either as the HTTP protocol, or

— as the ISO 8859-1 protocol.

NOTE In extended conformance classes of library integrated model LIIM 24-1, any other **external_file_protocol** may be referenced, subject to private agreement between the sender and the receiver.

EXPRESS specification:

```
*)
FUNCTION conformant_external_file_protocol_24_1(
     s: SET [0:?] OF external_file_protocol): BOOLEAN;

REPEAT i := 1 TO SIZEOF(s);
     IF NOT(conformant_8859_1_protocol_24_1(s[i])
          OR conformant_http_protocol_24_1(s[i]))
     THEN
          RETURN(FALSE);
     END_IF;
END_REPEAT;

RETURN(TRUE);

END_FUNCTION; -- conformant_external_file_protocol_24_1
(*
```

E.4.6 is_correct_liim_24_1_application_value function

The **is_correct_liim_24_1_application_value** function checks that the **liim_id library_iim_identification** is compatible with the conformance classes associated to the LIMM 24-1.

EXPRESS specification:

```
*)
FUNCTION is_correct_liim_24_1_application_value(
     liim_id: library_iim_identification): BOOLEAN;

IF EXISTS(liim_id\data_exchange_specification_identification.
     application)
     AND
     ((liim_id\data_exchange_specification_identification.
          application[1] = '1')
     OR
     (liim_id\data_exchange_specification_identification.
          application[1] = '2')
     OR
```



```
      (liim_id\data_exchange_specification_identification.
           application[1] = '3')
      OR
      (liim_id\data_exchange_specification_identification.
           application[1] = '4')
      OR
      (liim_id\data_exchange_specification_identification.
           application[1] = '5')
      OR
      (liim_id\data_exchange_specification_identification.
           application[1] = '6'))
      AND
      ((liim_id\data_exchange_specification_identification.
           application LIKE '#')
      OR
      (liim_id\data_exchange_specification_identification.
           Application LIKE '#E'))
THEN
      RETURN(TRUE);
ELSE
      RETURN(FALSE);
END_IF;

END_FUNCTION; -- is_correct_liim_24_1_application_value
(*
```

E.4.7 is_extended_liim_24_1_application_value function

The **is_extended_liim_24_1_application_value** function checks whether the **liim_id library_iim_identification** is associated to an extended conformance class.

EXPRESS specification:

```
*)
FUNCTION is_extended_liim_24_1_application_value(
      liim_id: library_iim_identification): BOOLEAN;

IF EXISTS(liim_id\data_exchange_specification_identification.
           application) AND
      (liim_id\data_exchange_specification_identification.
           application LIKE '#E')
THEN
      RETURN(TRUE);
ELSE
      RETURN(FALSE);
END_IF;

END_FUNCTION; -- is_extended_liim_24_1_application_value
(*
*)
END_SCHEMA; -- ISO13584_g_m_iim_conformance_schema
(*
```


Annex F
(normative)

Implementation method specific requirements for the library integrated information model 24-1

Conformance to the library integrated information model 24-1 shall be realised in one or more implementation methods. The implementation methods defines what types of exchange behaviour is required with respect to exchange protocols.

One implementation method is defined for the library delivery file: ISO 10303-21. The implementation methods for the possible external files referenced from the library delivery file and whose **external_file_protocol** belong to the standard data of the library integrated information model 24-1 are defined by the standard referenced in this **external_file_protocol**, possibly further specified as part of the description of the library integrated information model standard data (see annex E).

The implementation methods for the possible external files referenced from the library delivery files and whose external file protocols are supplier-defined (xxE conformance classes) shall be based on the provisions, if any, contained in the referenced standard in the case of a **standard_protocol**. They shall be based on previous agreement between the library data suppliers and the library users in the case of a **non_standard_protocol**.

For the exchange structure, the file format of the library delivery file shall be encoded according to the syntax and EXPRESS language mapping defined in ISO 10303-21 for the schema defined in annex C of this part of ISO 13584. The header of the exchange structure shall identify use of this part of ISO 13584 by the schema name 'ISO13584_g_m_iim_library_implicit_schema'.

NOTE Identification of the library delivery file is done by separate agreement between the sender and the receiver and is outside the scope of this part of ISO 13584.

Annex G
(normative)

ISO13584_f_m_iim_library_implicit_schema expanded listing

This annex references a listing of the complete EXPRESS schema specified in clause 16 of this part of ISO 13584 without comments or other explanatory text but with the constraints defined in the **ISO13584_f_m_iim_conformance_schema**. The name of this schema is **ISO13584_f_m_iim_library_implicit_schema**. In this listing, all the elements used either from the integrated resources of ISO 10303 (ISO 10303-4x) or from the logical resource or description methodology series of parts of ISO 13584 (ISO 13584-2x and ISO 13584-4x) by the **ISO13584_f_m_iim_schema** and the constraints defined in the **ISO13584_f_m_iim_conformance_schema** are gathered in an unique schema without any external reference.

This schema may be used:

— to exchange libraries that reference the **ISO13584_f_m_iim_schema** and its associated **ISO13584_f_m_iim_conformance_schema**, but that does not reference any view exchange protocol, and

— to exchange libraries that reference the **ISO13584_f_m_iim_schema** and its associated **ISO13584_f_m_iim_conformance_schema**, and that do reference some view exchange protocols; in this case, the constraints defined in these view exchange protocols are not checked.

This schema may also be completed to check the constraints defined in all the referenced view exchange protocols using the following process for each referenced view exchange protocol.

Assume that V1 is a referenced view exchange protocols and that it specifies two constraint schemas, the schema names of which are S1_V1 and S2_V1.

a) Build the long form of the S1_V1 schema and give to the resulting schema the same name: "S1_V1";

b) Build the long form of the S2_V1 schema and give to the resulting schema the same name: "S1_V1";

c) Replace everywhere in the long form of the S1_V1 schema, the string "S1_V1" by "ISO13584_f_m_iim_library_implicit_schema" with the same case;

d) Replace everywhere in the long form of the S2_V1 schema, the string "S2_V1" by "ISO13584_f_m_iim_library_implicit_schema" with the same case;

e) Check that all the entities referenced in the S1_V1 schema and in the S2_V1 schema are already existing in the **ISO13584_f_m_iim_library_implicit_schema**, else reference to the library integrated information model 24-2 and to the view exchange protocol S1 by a same library delivery file is not allowed.

NOTE 1 The information model of a library delivery file and the entities it may contain are specified by a library integrated information model. A view exchange protocol may only add constraints.

f) Add the content of the long form of the S1_V1 schema to the content of the **ISO13584_f_m_iim_library_implicit_schema**, removing possible duplicates;

g) Add the content of the long form of the S2_V1 schema to the content of the **ISO13584_f_m_iim_library_implicit_schema**, removing possible duplicates.

When the above process is performed for view exchange protocols V1, V2,...Vn, the resulting **ISO13584_f_m_iim_library_implicit_schema** may be used for exchanging any library that references the **ISO13584_f_m_iim_schema** and its associated

ISO13584_f_m_iim_conformance_schema as its library integrated information model, and that references whole or part of the V1, V2,...Vn view exchange protocol set. This schema also includes the constraints of all the referenced view exchange protocols.

The listing of the **ISO13584_f_m_iim_library_implicit_schema** schema is available in computer-interpretable form and can be found at the following URL:

http://www.tc184-sc4.org/EXPRESS/

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

NOTE 2 The information provided in computer-interpretable form at the above URLs is normative.

NOTE 3 If some errors are identified in the EXPRESS code during implementation and before publication of Technical Corrigendum, the description of these errors, together with the corrections recommended for PLIB implementations by the part editors can be found at the following URL:

http://www.lisi.ensma.fr/ftp/pub/PLIB_release_notes/Part24/Part24-IS/

Annex H
(informative)

ISO13584_f_m_iim_schema short names of entities

This annex references a listing of the EXPRESS entity names and corresponding short names for the EXPRESS schema specified in annex G of this part of ISO 13584. This listing is available in computer-interpretable form and can be found at the following URL:

http://www.tc184-sc4.org/Short_Names/

NOTE The information provided in computer-interpretable form at the above URLs is informative. The information that is contained in the body of this part of ISO 10303 is normative.

Annex I
(normative)

Standard data requirements for the library integrated information model 24-2

Standard data are the entity instances that shall be recognised by any implementation conformant with ISO 13584 that claims conformance to some conformance class of some library integrated information model or view exchange protocol.

Standard data shall be specified by each library integrated information model and by each view exchange protocol, and for each of them, for each conformance class.

Standard data may include:

— instances of **basic_semantic_unit**s, associated with the corresponding **dictionary_element** and possibly **content_item**,

— instances of **external_file_protocol**s, and

— instance of other entities required to define the previous entities instances.

Recognition of a **basic_semantic_unit** means that a value-equal **basic_semantic_unit** is already stored in the user system, together with a corresponding **dictionary_element** and possibly a **content_item** as specified in the view exchange protocol or library integrated information model standard data. This implies that a reference to a value-equal **basic_semantic_unit** in a supplier library is interpreted as a reference to the pre-existing **basic_semantic_unit**.

NOTE Value equality of two entity instances means that all their corresponding attributes have the same values. Value-equality between two basic semantic units implies that they both identify the same concept.

EXAMPLE 1 Examples of **basic_semantic_units** that may be defined as standard data in a view exchange protocol include the **class_BSU** that identifies the functional view class possibly defined by the view exchange protocol and the **property_BSU** that identifies the view control variable of this functional view class.

Recognition of an external file protocol means that external files that reference a value equal **external_file_protocol** shall be processed by an implementation that recognises this **external_file_protocol**.

EXAMPLE 2 Example of an external file protocol that may be defined as standard data by a view exchange protocol or a library integrated information model is the ISO standard ISO 8859-1 that specifies a 8-bit, single-byte-coded graphics character set for Latin alphabet N°1.

Standard data are specified by means of a set of constraints that shall be fulfilled by any library that claims conformance to some conformance class of LIIM 24-2. The following standard data are specified by library integrated information model 24-2.

I.1 Constraints on a library delivery file for referencing library integrated information model 24-2

This subclause defines **library_iim_identification** instance values that are allowed for use in a library delivery file to reference library integrated information model 24-2 defined in this part of ISO 13584.

The set of allowed values is defined by means of Table I.1 that specifies for each conformance class the allowed values of **library_iim_identification.name** and **library_iim_identification.application**, and by means of one EXPRESS schema that contains a global rule. This rule shall be fulfilled by any library delivery file that references library integrated information model 24-2, defined in this part of ISO 13584 in any of its conformance class. The goal of this rule is to specify the allowed values for the

other attributes of **library_iim_identification** that shall be used to reference library integrated information model 24-2, by means of relationships with **library_iim_identification.name** and **library_iim_identification.name.application**.

This rule is included in the **ISO13584_f_m_iim_library_implicit_schema** specified in annex G.

I.2 Conformance class specification table

Table I.1 specifies the values of **library_iim_identification.name** and **library_iim_identification.application** that are allowed for use in a **library_iim_identification** to reference library integrated information model 24-2 in either of its conformance classes.

Table I.1 — ISO 13584 LIIM 24-2 conformance class specification

Conformance Class	library_iim_identification.name mandatory value	library_iim_identification.application mandatory value
1	'ISO_13584_24_2'	'1'
2	'ISO_13584_24_2'	'2'
3	'ISO_13584_24_2'	'3'
4	'ISO_13584_24_2'	'4'
5	'ISO_13584_24_2'	'5'
6	'ISO_13584_24_2'	'6'
1E	'ISO_13584_24_2'	'1E'
2E	'ISO_13584_24_2'	'2E'
3E	'ISO_13584_24_2'	'3E'
4E	'ISO_13584_24_2'	'4E'
5E	'ISO_13584_24_2'	'5E'
6E	'ISO_13584_24_2'	'6E'

I.3 Standard data for conformance class 1 to 6E (all the conformance classes)

I.3.1 Constraints on a library delivery file conform to the library integrated model LIIM 24-2

The **library_iim_identification** instance values allowed for use in a library delivery file conform to the library integrated model LIIM 24-2 defined in this part of ISO 13584 shall obey the constraints defined in the following EXPRESS schema.

EXPRESS specification:

```
*)
SCHEMA ISO13584_f_m_iim_conformance_schema;

USE FROM ISO13584_IEC61360_dictionary_schema(
      item_names);

USE FROM ISO13584_IEC61360_language_resource_schema(
      translated_label);

USE FROM support_resource_schema(
      label);

USE FROM person_organization_schema(
      organization);

USE FROM ISO13584_extended_dictionary_schema(
      data_exchange_specification_identification,
      library_iim_identification);

USE FROM ISO13584_external_file_schema(
      http_protocol,
      standard_data_protocol,
      external_file_protocol);
(*
```

NOTE The schemas used above can be found in the following document:
ISO13584_IEC61360_dictionary_schema IEC 61360-2
(which is duplicated for convenience in informative annex D of ISO 13584-42),
person_organization_schema ISO 10303-41,
ISO13584_extended_dictionary_schema This part of ISO 13584,
ISO13584_external_file_schema This part of ISO 13584.

I.3.2 Allowed_reference_to_LIIM_24_2_rule rule

The **allowed_reference_to_LIIM_24_2_rule** rule defines a formal constraint and an informal constraint on **library_iim_identification**s to be allowed for use to reference conformance class 1 to 6E of library integrated model LIIM 24-2 defined in this part of ISO 13584. A **library_iim_identification** is allowed for use to reference conformance class 1 to 6E of library integrated model LIIM 24-2 if the following conditions hold:

— the **name** attribute of the **library_iim_identification** that reference library integrated model LIIM 24-2 shall be equal to 'ISO_13584_24_2', and

— the **status** attribute of the **library_iim_identification** shall be equal to 'WD', 'CD', 'DIS', 'FDIS', 'IS', 'TS', 'PAS' or 'ITA' and

— the **application** attribute of the **library_iim_identification** shall have ‘1', '2', '3', '4', '5' or '6' possibly followed by 'E' as its value, and

— if the conformance class of library integrated model LIIM 24-2 referenced is not an extended conformance class, the **external_file_protocol**s referenced by the **external_file_protocols** attribute of the **library_iim_identification** shall fulfil the constraints required by the **conformant_external_file_protocol_24_2** function.

Moreover, a **library_iim_identification** is allowed for use to reference conformance class 1 to 6 of library integrated model LIIM 24-2 if one of the two following conditions hold concerning the **http_file**s that may be referenced directly or indirectly from the **library_iim_identification**,

— either each referenced **http_file** it is associated with a **mime** attribute and an **exchange_format** attribute corresponding to MIME type and subtype that correspond to a specification that is publicly available, or

— it is associated with a **mime** attribute and an **exchange_format** attribute corresponding to MIME type and subtype that correspond to a specification that is associated with public domain Internet-available readers.

Reference to **http_file**s corresponding to other MIME types and subtypes may only be done in extended conformance classes. This is documented as an informal proposition **IP1** in **allowed_reference_to_LIIM_24_2_rule**.

EXPRESS specification:

```
*)
RULE allowed_reference_to_LIIM_24_2_rule FOR(
     library_iim_identification);
WHERE
     WR1: QUERY(liim_id <* library_iim_identification |
          ((liim_id\data_exchange_specification_identification
               .status = 'WD')
               OR
          (liim_id\data_exchange_specification_identification
               .status = 'CD')
               OR
          (liim_id\data_exchange_specification_identification
               .status = 'DIS')
               OR
          (liim_id\data_exchange_specification_identification
               .status = 'FDIS')
               OR
          (liim_id\data_exchange_specification_identification
               .status = 'IS')
               OR
          (liim_id\data_exchange_specification_identification
               .status = 'TS')
               OR
          (liim_id\data_exchange_specification_identification
               .status = 'PAS')
               OR
          (liim_id\data_exchange_specification_identification
               .status = 'ITA'))
          AND
          (liim_id\data_exchange_specification_identification.
               Name = 'ISO_13584_24_2')
          AND
          is_correct_liim_24_2_application_value(liim_id)
          AND
          (QUERY(efp <*
```



```
            liim_id\data_exchange_specification_identification
            .external_file_protocols
            | NOT(is_extended_liim_24_2_application_value(liim_id))
            AND
            NOT(conformant_external_file_protocol_24_2([efp]))
            ) = []))
       = QUERY(liim_id <* library_iim_identification |
            (liim_id\data_exchange_specification_identification
            .name = 'ISO_13584_24_2'));
END_RULE; -- allowed_reference_to_LIIM_24_2_rule
(*
```

Formal propositions:

WR1: when referencing library integrated model LIIM 24-2 defined in this part of ISO 13584, the **library_iim_identification.name** shall have ‘ISO_13584_24_2’ as its value, **library_iim_identification.status** shall be equal to 'WD', 'CD', 'DIS', 'FDIS', 'IS', 'TS', 'PAS' or 'ITA', the **library_iim_identification.application** shall have ‘1', '2', '3', '4', '5' or '6' as its value, possibly followed by 'E', and, if the conformance class of library integrated model LIIM 24-2referenced is not an extended conformance class, the **library_iim_identification.external_file_protocols** shall fulfil the constraints specifications required by the **conformant_external_file_protocol_24_2** function defined below.

Informal propositions:

IP1: when it references library integrated model LIIM 24-2 defined in this part of ISO 13584 in one of the conformance class 1, 2, 3, 4, 5 or 6, a **library_iim_identification** may only reference, directly or indirectly, **http_file**s characterised by MIME types and subtypes that either correspond to specifications that are publicly available, or to specifications that are associated with public domain Internet-available readers.

I.3.3 conformant_http_protocol_24_2 function

The **conformant_http_protocol_24_2** function checks whether an **external_file_protocol** may be referenced as the HTTP protocol by a **library_iim_identification** that references library integrated model LIIM 24-2 in any of its conformance classes, or not. It returns TRUE if the given **external_file_protocol** is allowed for reference, otherwise, it returns FALSE. An **external_file_protocol** may be referenced as the HTTP protocol by a **library_iim_identification** that reference library integrated model LIIM 24-2 in any of its conformance classes if the following conditions hold:

— the **external_file_protocol** shall be a **http_protocol**, and

— the **organisation** attribute of the **external_file_protocol** shall reference an organization of which the **id** attribute equals to 'IAB' and the **name** attribute equals to 'Internet Architecture Board', and

— the **protocol_name** attribute of the **external_file_protocol** shall equal to 'HTTP' or to ‘HTTPS’, and

— the **designation** attribute of the **external_file_protocol** shall reference an **item_names** for which the **preferred_name** attribute equals to 'Hypertext Transfer Protocol' and the **short_name** attribute equals to 'RFC' followed by four digits and possibly some other characters.

EXPRESS specification:

```
*)
FUNCTION conformant_http_protocol_24_2(
      ef: external_file_protocol): BOOLEAN;

LOCAL
      ok: BOOLEAN := TRUE;
END_LOCAL;

IF (('ISO13584_EXTERNAL_FILE_SCHEMA.HTTP_PROTOCOL' IN TYPEOF(ef)) AND
      (ef.organisation.id = 'IAB') AND
      (ef.organisation.name = 'Internet Architecture Board') AND
      ((ef.protocol_name = 'HTTP')
           OR (ef.protocol_name = 'HTTPS'))AND
      (ef.designation.preferred_name = 'Hypertext Transfer Protocol'))
THEN
      IF 'ISO13584_IEC61360_DICTIONARY_SCHEMA.TRANSLATED_LABEL'
           IN TYPEOF(ef.designation.short_name)
      THEN
           REPEAT i := 1 TO SIZEOF(ef.designation.short_name.labels);
                IF ef.designation.short_name.labels[i] LIKE 'RFC####&'
                THEN
                     ok := ok AND TRUE;
                ELSE
                     ok := ok AND FALSE;
                END_IF;
           END_REPEAT;
           RETURN(OK);
      ELSE
           IF ef.designation.short_name
                LIKE 'RFC####&'
           THEN
                RETURN(TRUE);
           ELSE
                RETURN(FALSE);
           END_IF;
      END_IF;
ELSE
      RETURN(FALSE);
END_IF;

END_FUNCTION; -- conformant_http_protocol_24_2
(*
```

I.3.4 conformant_8859_1_protocol_24_2 function

The **conformant_8859_1_protocol_24_2** function checks whether an **external_file_protocol** may be referenced as the ISO 8859-1 protocol by a **library_iim_identification** that references library integrated model LIIM 24-2 in any of its conformance classes, or not. It returns TRUE if the given **external_file_protocol** is allowed for reference, otherwise, it returns FALSE. An **external_file_protocol** may be referenced as the ISO 8859-1 protocol by a

library_iim_identification that represents reference library integrated model LIIM 24-2 in any of its conformance classes, if the following conditions hold:

— the **external_file_protocol** shall be a **standard_data_protocol**, and

— the **organisation** attribute of the **external_file_protocol** shall reference an organization of which the **id** attribute equals to 'ISO' and the **name** attribute equals to 'International Organisation for Standardisation', and

— the **protocol_name** attribute of the **external_file_protocol** shall equal to 'ISO_8859_1', and

— the **designation** attribute of the **external_file_protocol** shall reference an **item_names** for which the **preferred_name** attribute equals to 'Latin alphabet No 1' and the **short_name** attribute equals to 'ISO 8859-1'.

EXPRESS specification:

```
*)
FUNCTION conformant_8859_1_protocol_24_2(
     ef: external_file_protocol): BOOLEAN;

IF (('ISO13584_EXTERNAL_FILE_SCHEMA'
     + '.STANDARD_DATA_PROTOCOL' IN TYPEOF(ef)) AND
     (ef.organisation.id = 'ISO') AND
     (ef.organisation.name
     = 'International Organisation for Standardisation') AND
     (ef.protocol_name = 'ISO_8859_1') AND
     (ef.designation.preferred_name
     = 'Latin alphabet No 1') AND
     (ef.designation.short_name = 'ISO 8859-1')
     )
THEN
     RETURN(TRUE);
ELSE
     RETURN(FALSE);
END_IF;
END_FUNCTION; -- conformant_8859_1_protocol_24_2
(*
```

I.3.5 conformant_external_file_protocol_24_2 function

The **conformant_external_file_protocol_24_2** function checks whether all the **external_file_protocol**s of a set of **external_file_protocol**s may be referenced as an library integrated model LIIM 24-2 by a **library_iim_identification** that references library integrated model LIIM 24-2 in one of its conformance class 1 to 6, or not. It returns TRUE if all the **external_file_protocol**s of a set of **external_file_protocol**s are allowed for reference, otherwise, it returns FALSE.

An **external_file_protocol** may be referenced by a **library_iim_identification** that represents conformance class 1 to 6 of library integrated model LIIM 24-2 if it may be referenced:

— either as the HTTP protocol, or

— as the ISO 8859-1 protocol.

NOTE In extended conformance classes of library integrated model LIIM 24-2, any other e**xternal_file_protocol** may be referenced, subject to private agreement between the sender and the receiver.

EXPRESS specification:

```
*)
FUNCTION conformant_external_file_protocol_24_2(
      s: SET [0:?] OF external_file_protocol): BOOLEAN;

REPEAT i := 1 TO SIZEOF(s);
      IF NOT(conformant_8859_1_protocol_24_2(s[i])
           OR conformant_http_protocol_24_2(s[i]))
      THEN
           RETURN(FALSE);
      END_IF;
END_REPEAT;

RETURN(TRUE);

END_FUNCTION; -- conformant_external_file_protocol_24_2
(*
```

I.3.6 is_correct_liim_24_2_application_value function

The **is_correct_liim_24_2_application_value** function checks that the **liim_id library_iim_identification** is compatible with the conformance classes associated to the LIMM 24-2.

EXPRESS specification:

```
*)
FUNCTION is_correct_liim_24_2_application_value(
      liim_id: library_iim_identification): BOOLEAN;

IF EXISTS(liim_id\data_exchange_specification_identification
      .application)
      AND
      ((liim_id\data_exchange_specification_identification
           .application[1] = '1')
           OR
      (liim_id\data_exchange_specification_identification
           .application[1] = '2')
           OR
      (liim_id\data_exchange_specification_identification
           .application[1] = '3')
           OR
      (liim_id\data_exchange_specification_identification
           .application[1] = '4')
           OR
      (liim_id\data_exchange_specification_identification
           .application[1] = '5')
           OR
      (liim_id\data_exchange_specification_identification
```



```
          .application[1] = '6'))
     AND
     ((liim_id\data_exchange_specification_identification
          .application LIKE '#')
          OR
     (liim_id\data_exchange_specification_identification
          .application LIKE '#E'))
THEN
     RETURN(TRUE);
ELSE
     RETURN(FALSE);
END_IF;
END_FUNCTION; -- is_correct_liim_24_2_application_value
(*
```

I.3.7 is_extended_liim_24_2_application_value function

The **is_extended_liim_24_2_application_value** function checks whether, the **liim_id library_iim_identification** is associated to an extended conformance class.

EXPRESS specification:

```
*)
FUNCTION is_extended_liim_24_2_application_value(
     liim_id: library_iim_identification): BOOLEAN;
IF EXISTS(liim_id\data_exchange_specification_identification
     .application)
     AND
     (liim_id\data_exchange_specification_identification
          .application LIKE '#E')
THEN
     RETURN(TRUE);
ELSE
     RETURN(FALSE);
END_IF;
END_FUNCTION; -- is_extended_liim_24_2_application_value
(*

*)

END_SCHEMA; -- ISO13584_f_m_iim_conformance_schema

(*
```

Annex J
(normative)

Implementation method specific requirements for the library integrated information model 24-2

Conformance to the library integrated information model 24-2 shall be effected by one or more implementation methods. The implementation methods defines what types of exchange behaviour is required with respect to exchange protocols.

One implementation method is defined for the library delivery file: ISO 10303-21. The implementation methods for the possible external files referenced from the library delivery file and whose **external_file_protocol**s belongs to the standard data of the library integrated information model 24-2 are defined by the standard referenced in this protocol, possibly further specified as part of the description of the integrated information model standard data (see annex I).

The implementation methods for the possible external files referenced from the library delivery files and whose external file protocols are supplier-defined (xxE conformance classes) shall be based on the provisions, if any, contained in the referenced standard in the case of a **standard_protocol**. They shall be based on previous agreement between the library data suppliers and the library users in the case of a **non_standard_protocol**.

For the exchange structure, the file format of the library delivery file shall be encoded according to the syntax and EXPRESS language mapping defined in ISO 10303-21 for the schema defined in annex G of this part of ISO 13584. The header of the exchange structure shall identify use of this part of ISO 13584 by the schema name 'ISO13584_f_m_iim_library_implicit_schema'.

NOTE Identification of the library delivery file is done by separate agreement between the sender and the receiver and is outside the scope of this part of ISO 13584.

Annex K
(normative)

ISO13584_f_v_iim_library_implicit_schema expanded listing

This annex references a listing of the complete EXPRESS schema specified in clause 18 of this part of ISO 13584 without comments or other explanatory text but with the constraints defined in the **ISO13584_f_v_iim_conformance_schema**. The name of this schema is **ISO13584_f_v_iim_library_implicit_schema**. In this listing, all the elements used either from the integrated resources of ISO 10303 (ISO 10303-4x) or from the logical resource or description methodology series of parts of ISO 13584 (ISO 13584-2x and ISO 13584-4x) by the **ISO13584_f_v_iim_schema** and the constraints defined in the **ISO13584_f_v_iim_conformance_schema** are gathered in an unique schema without any external reference.

This schema may be used:

— to exchange libraries that reference the **ISO13584_f_v_iim_schema** and its associated **ISO13584_f_v_iim_conformance_schema**, but that does not reference any view exchange protocol, and

— to exchange libraries that reference the **ISO13584_f_v_iim_schema** and its associated **ISO13584_f_v_iim_conformance_schema**, and that do reference some view exchange protocols; in this case, the constraints defined in these view exchange protocols are not checked.

This schema may also be completed to check the constraints defined in all the referenced view exchange protocols using the following process for each referenced view exchange protocol.

Assume that V1 is a referenced view exchange protocols and that it specifies two constraint schemas, the schema names of which are S1_V1, S2_V1.

a) Build the long form of the S1_V1 schema and give to the resulting schema the same name: "S1_V1";

b) Build the long form of the S2_V1 schema and give to the resulting schema the same name: "S2_V1";

c) Replace everywhere in the long form of the S1_V1 schema, the string "S1_V1" by "ISO13584_f_v_iim_library_implicit_schema" with the same case;

d) Replace everywhere in the long form of the S2_V1 schema, the string "S2_V1" by "ISO13584_f_v_iim_library_implicit_schema" with the same case;

e) Check that all the entities referenced in the S1_V1 schema and in the S2_V1 schema are already existing in the **ISO13584_f_v_iim_library_implicit_schema**, else reference to the library integrated information model 24-3 and to the view exchange protocol S1 by a same library delivery file is not allowed.

NOTE 1 The information model of a library delivery file and the entities it may contain are specified by a library integrated information model. A view exchange protocol may only add constraints.

f) Add the content of the long form of the S1_V1 schema to the content of the **ISO13584_f_v_iim_library_implicit_schema**, removing possible duplicates;

g) Add the content of the long form of the S2_V1 schema to the content of the **ISO13584_f_v_iim_library_implicit_schema**, removing possible duplicates.

When the above process is performed for view exchange protocols V1, V2,...Vn, the resulting **ISO13584_f_v_iim_library_implicit_schema** may be used for exchanging any library that references

the **ISO13584_f_v_iim_schema** and its associated **ISO13584_f_v_iim_conformance_schema** as its library integrated information model, and that references whole or part of the V1, V2,...Vn view exchange protocol set. This schema also includes the constraints of all the referenced view exchange protocols.

The listing of the **ISO13584_f_v_iim_library_implicit_schema** schema is available in computer-interpretable form and can be found at the following URL:

http://www.tc184-sc4.org/EXPRESS/

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

NOTE 2 The information provided in computer-interpretable form at the above URLs is normative.

NOTE 3 In some errors are identified in the EXPRESS code during implementation and before publication of Technical Corrigendum, the description of these errors, together with the corrections recommended for PLIB implementations by the part editors can be found at the following URL:

http://www.lisi.ensma.fr/ftp/pub/PLIB_release_notes/Part24/Part24-IS/

Annex L
(informative)

ISO13584_f_v_iim_schema short names of entities

This annex references a listing of the EXPRESS entity names and corresponding short names of the EXPRESS schema specified in annex K of this part of ISO 13584. This listing is available in computer-interpretable form and can be found at the following URL:

http://www.tc184-sc4.org/Short_Names/

NOTE The information provided in computer-interpretable form at the above URLs is informative. The information that is contained in the body of this part of ISO 10303 is normative.

Annex M
(normative)

Standard data requirements for the library integrated information model 24-3

Standard data are the entity instances that shall be recognised by any implementation conformant with ISO 13584 that claims conformance to some conformance class of some library integrated information model or view exchange protocol.

Standard data shall be specified by each library integrated information model and by each view exchange protocol, and for each of them, for each conformance class.

Standard data may include:

— instances of **basic_semantic_unit**s, associated with the corresponding **dictionary_element** and possibly **content_item**,

— instances of **external_file_protocol**s, and

— instance of other entities required to define the previous entities instances.

Recognition of a **basic_semantic_unit** means that a value-equal **basic_semantic_unit** is already stored in the user system, together with a corresponding **dictionary_element** and possibly a **content_item** as specified in the view exchange protocol or library integrated information model standard data. This implies that a reference to a value-equal **basic_semantic_unit** in a supplier library is interpreted as a reference to the pre-existing **basic_semantic_unit**.

NOTE Value equality of two entity instances means that all their corresponding attributes have the same values. Value-equality between two basic semantic units implies that they both identify the same concept.

EXAMPLE 1 Examples of **basic_semantic_units** that may be defined as standard data in a view exchange protocol include the **class_BSU** that identifies the functional view class possibly defined by the view exchange protocol and the **property_BSU** that identifies the view control variable of this functional view class.

Recognition of an external file protocol means that external files that reference a value equal **external_file_protocol** shall be processed by an implementation that recognises this **external_file_protocol**.

EXAMPLE 2 Example of an external file protocol that may be defined as standard data by a view exchange protocol or a library integrated information model is the ISO standard ISO 8859-1 that specifies a 8-bit, single-byte-coded graphics character set for Latin alphabet N°1.

Standard data are specified by means of a set of constraints that shall be fulfilled by any library that claims conformance to some conformance class of LIIM 24-3. The following standard data are specified by library integrated information model 24-3.

M.1 Constraints on a library delivery file for referencing library integrated information model 24-3

This subclause defines **library_iim_identification** instance values that are allowed for use in a library delivery file to reference library integrated information model 24-3 defined in this part of ISO 13584.

The set of allowed values is defined by means of Table M.1 that specifies for each conformance class the allowed values of **library_iim_identification.name** and **library_iim_identification.application**, and by means of one EXPRESS schema that contains a global rule. This rules shall be fulfilled by any library delivery file that references library integrated information model 24-3, defined in this part of ISO 13584 in any of its conformance class. The goal of this rule is to specify the allowed values for the other attributes of **library_iim_identification** that shall be used to reference library integrated

information model 24-3, by means of relationships with **library_iim_identification.name** and **library_iim_identification.application**.

This rule is included in the **ISO13584_f_v_iim_library_implicit_schema** specified in annex K.

M.2 Conformance class specification table

Table M.1 specifies the values of **library_iim_identification.name** and **library_iim_identification.application** that are allowed for use in a **library_iim_identification** to reference library integrated information model 24-3 in either of its conformance classes.

Table M.1 — ISO 13584 LIIM 24-3 conformance class specification

Conformance Class	library_iim_identification.name mandatory value	library_iim_identification.application mandatory value
1	'ISO_13584_24_3'	'1'
2	'ISO_13584_24_3'	'2'
1E	'ISO_13584_24_3'	'1E'
2E	'ISO_13584_24_3'	'2E'

M.3 Standard data for conformance class 1 to 2E (all the conformance classes)

M.3.1 Constraints on a library delivery file conform to the library integrated model LIIM 24-3

The **library_iim_identification** instance values allowed for use in a library delivery file conform to the library integrated model LIIM 24-3 defined in this part of ISO 13584 shall obey the constraints defined in the following EXPRESS schema.

EXPRESS specification:

```
*)
SCHEMA ISO13584_f_v_iim_conformance_schema;

USE FROM ISO13584_IEC61360_dictionary_schema(
      item_names);

USE FROM ISO13584_IEC61360_language_resource_schema(
      translated_label);

USE FROM person_organization_schema(
      organization);

USE FROM support_resource_schema(
      label);
```

```
    USE FROM ISO13584_extended_dictionary_schema(
          data_exchange_specification_identification,
          library_iim_identification);

    USE FROM ISO13584_external_file_schema(
          http_protocol,
          standard_data_protocol,
          external_file_protocol);
    (*
```

NOTE The schemas used above can be found in the following document:
ISO13584_IEC61360_dictionary_schema IEC 61360-2
(which is duplicated for convenience in informative annex D of ISO 13584-42),
person_organization_schema ISO 10303-41,
ISO13584_extended_dictionary_schema This part of ISO 13584,
ISO13584_external_file_schema This part of ISO 13584.

M.3.2 Allowed_reference_to_LIIM_24_3_rule rule

The **allowed_reference_to_LIIM_24_3_rule** rule defines a formal constraint and an informal constraint on **library_iim_identification**s to be allowed for use to reference conformance class 1 to 2E of library integrated model LIIM 24-3 defined in this part of ISO 13584. A **library_iim_identification** is allowed for use to reference conformance class 1 to 2E of library integrated model LIIM 24-3 if the following conditions hold:

— the **name** attribute of the **library_iim_identification** that reference library integrated model LIIM 24-3 shall be equal to 'ISO_13584_24_3', and

— the **status** attribute of the **library_iim_identification** shall be equal to 'WD', 'CD', 'DIS', 'FDIS', 'IS', 'TS', 'PAS' or 'ITA' and

— the **application** attribute of the **library_iim_identification** shall have '1' or '2' possibly followed by 'E' as its value, and

— if the conformance class of library integrated model LIIM 24-3 referenced is not an extended conformance class, the **external_file_protocol**s referenced by the **external_file_protocols** attribute of the **library_iim_identification** shall fulfil the constraints required by the **conformant_external_file_protocol_24_3** function.

Moreover, a **library_iim_identification** is allowed for use to reference conformance class 1 to 2 of library integrated model LIIM 24-3 if one of the two following conditions hold concerning the **http_file**s that may be referenced directly or indirectly from the **library_iim_identification**,

— either each referenced **http_file** it is associated with a **mime** attribute and an **exchange_format** attribute corresponding to MIME type and subtype that correspond to a specification that is publicly available, or

— it is associated with a **mime** attribute and an **exchange_format** attribute corresponding to MIME type and subtype that correspond to a specification that is associated with public domain Internet-available readers.

Reference to **http_file**s corresponding to other MIME types and subtypes may only be done in extended conformance classes. This is documented as an informal proposition **IP1** in **allowed_reference_to_LIIM_24_3_rule**.

EXPRESS specification:

```
*)
RULE allowed_reference_to_LIIM_24_3_rule FOR(
      library_iim_identification);
WHERE
      WR1: QUERY(liim_id <* library_iim_identification |
           ((liim_id\data_exchange_specification_identification
                .status = 'WD')
                OR
           (liim_id\data_exchange_specification_identification
                .status = 'CD')
                OR
           (liim_id\data_exchange_specification_identification
                .status = 'DIS')
                OR
           (liim_id\data_exchange_specification_identification
                .status = 'FDIS')
                OR
           (liim_id\data_exchange_specification_identification
                .status = 'IS')
                OR
           (liim_id\data_exchange_specification_identification
                .status = 'TS')
                OR
           (liim_id\data_exchange_specification_identification
                .status = 'PAS')
                OR
           (liim_id\data_exchange_specification_identification
                .status = 'ITA'))
           AND
           (liim_id\data_exchange_specification_identification.name
           = 'ISO_13584_24_3')
           AND
           is_correct_liim_24_3_application_value(liim_id)
           AND
           (QUERY(efp <*
           liim_id\data_exchange_specification_identification
           .external_file_protocols
           | NOT(is_extended_liim_24_3_application_value(liim_id))
           AND
           NOT(conformant_external_file_protocol_24_3([efp]))
           ) = []))
      = QUERY(liim_id <* library_iim_identification |
           (liim_id\data_exchange_specification_identification
           .name = 'ISO_13584_24_3'));
END_RULE; -- allowed_reference_to_LIIM_24_3_rule
(*
```

Formal propositions:

WR1: when referencing library integrated model LIIM 24-3 defined in this part of ISO 13584, the **library_iim_identification.name** shall have 'ISO_13584_24_3' as its value, **library_iim_identification.status** shall be equal to 'WD', 'CD', 'DIS', 'FDIS', 'IS', 'TS', 'PAS' or 'ITA', the **library_iim_identification.application** shall have '1' or '2' as its value, possibly followed by 'E', and, if the conformance class of library integrated model LIIM 24-3 referenced is not an extended conformance class, the **library_iim_identification.external_file_protocols** shall fulfil the constraints specifications required by the **conformant_external_file_protocol_24_3** function defined below.

Informal propositions:

IP1: when it references library integrated model LIIM 24-3 defined in this part of ISO 13584 in one of the conformance class 1 or 2 a **library_iim_identification** may only reference, directly or indirectly, **http_file**s characterised by MIME types and subtypes that either correspond to specifications that are publicly available, or to specifications that are associated with public domain Internet-available readers.

M.3.3 conformant_http_protocol_24_3 function

The **conformant_http_protocol_24_3** function checks whether an **external_file_protocol** may be referenced as the HTTP protocol by a **library_iim_identification** that references library integrated model LIIM 24-3 in any of its conformance classes. It returns TRUE if the given **external_file_protocol** is allowed for reference, otherwise, it returns FALSE. An **external_file_protocol** may be referenced as the HTTP protocol by a **library_iim_identification** that reference library integrated model LIIM 24-3 in any of its conformance classes if the following conditions hold:

— the **external_file_protocol** shall be a **http_protocol**, and

— the **organisation** attribute of the **external_file_protocol** shall reference an organization of which the **id** attribute equals to 'IAB' and the **name** attribute equals to 'Internet Architecture Board', and

— the **protocol_name** attribute of the **external_file_protocol** shall equal to 'HTTP' or to 'HTTPS', and

— the **designation** attribute of the **external_file_protocol** shall reference an **item_names** for which the **preferred_name** attribute equals to 'Hypertext Transfer Protocol' and the **short_name** attribute equals to 'RFC' followed by four digits and possibly some other characters.

EXPRESS specification:

```
*)
FUNCTION conformant_http_protocol_24_3(
      ef: external_file_protocol): BOOLEAN;

LOCAL
      OK: BOOLEAN := TRUE;
END_LOCAL;

IF (('ISO13584_EXTERNAL_FILE_SCHEMA'
      + '.HTTP_PROTOCOL' IN TYPEOF(ef)) AND
      (ef.organisation.id = 'IAB') AND
      (ef.organisation.name = 'Internet Architecture Board') AND
      ((ef.protocol_name = 'HTTP')
```



```
          OR (ef.protocol_name = 'HTTPS')) AND
     (ef.designation.preferred_name = 'Hypertext Transfer Protocol'))
THEN
     IF 'ISO13584_IEC61360_DICTIONARY_SCHEMA.TRANSLATED_LABEL'
          IN TYPEOF(ef.designation.short_name)
     THEN
          REPEAT i := 1 TO SIZEOF(ef.designation.short_name.labels);

               IF ef.designation.short_name.labels[i]
                    LIKE 'RFC####&'
               THEN
                    ok := ok AND TRUE;
               ELSE
                    ok := ok AND FALSE;
               END_IF;
          END_REPEAT;
          RETURN(OK);
     ELSE
          IF ef.designation.short_name
               LIKE 'RFC####&'
          THEN
               RETURN(TRUE);
          ELSE
               RETURN(FALSE);
          END_IF;
     END_IF;
ELSE
     RETURN(FALSE);
END_IF;

END_FUNCTION; -- conformant_http_protocol_24_3
(*
```

M.3.4 conformant_8859_1_protocol_24_3 function

The **conformant_8859_1_protocol_24_3** function checks whether an **external_file_protocol** may be referenced as the ISO 8859-1 protocol by a **library_iim_identification** that references library integrated model LIIM 24-3 in any of its conformance classes. It returns TRUE if the given **external_file_protocol** is allowed for reference, otherwise, it returns FALSE. An **external_file_protocol** may be referenced as the ISO 8859-1 protocol by a **library_iim_identification** that represents reference library integrated model LIIM 24-3 in any of its conformance classes, if the following conditions hold:

— the **external_file_protocol** shall be a **standard_data_protocol**, and

— the **organisation** attribute of the **external_file_protocol** shall reference an organization of which the **id** attribute equals to 'ISO' and the **name** attribute equals to 'International Organisation for Standardisation', and

— the **protocol_name** attribute of the **external_file_protocol** shall equal to 'ISO_8859_1', and

— the **designation** attribute of the **external_file_protocol** shall reference an **item_names** for which the **preferred_name** attribute equals to 'Latin alphabet No 1' and the **short_name** attribute equals to 'ISO 8859-1'.

EXPRESS specification:

```
*)
FUNCTION conformant_8859_1_protocol_24_3(
     ef: external_file_protocol): BOOLEAN;

IF (('ISO13584_EXTERNAL_FILE_SCHEMA'
     + '.STANDARD_DATA_PROTOCOL' IN TYPEOF(ef)) AND
     (ef.organisation.id = 'ISO') AND
     (ef.organisation.name
          = 'International Organisation for Standardisation') AND
     (ef.protocol_name = 'ISO_8859_1') AND
     (ef.designation.preferred_name
          = 'Latin alphabet No 1') AND
     (ef.designation.short_name = 'ISO 8859-1'))
THEN
     RETURN(TRUE);
ELSE
     RETURN(FALSE);
END_IF;
END_FUNCTION; -- conformant_8859_1_protocol_24_3
(*
```

M.3.5 conformant_external_file_protocol_24_3 function

The **conformant_external_file_protocol_24_3** function checks whether all the **external_file_protocol**s of a set of **external_file_protocol**s may be referenced as an library integrated model LIIM 24-3 by a **library_iim_identification** that references library integrated model LIIM 24-3 in one of its conformance class 1 to 6, or not. It returns TRUE if all the **external_file_protocol**s of a set of **external_file_protocol**s are allowed for reference, otherwise, it returns FALSE.

An **external_file_protocol** may be referenced by a **library_iim_identification** that represents conformance class 1 to 6 of library integrated model LIIM 24-3 if it may be referenced:

— either as the HTTP protocol, or

— as the ISO 8859-1 protocol.

NOTE In extended conformance classes of library integrated model LIIM 24-3, any other e**xternal_file_protocol** may be referenced, subject to private agreement between the sender and the receiver.

EXPRESS specification:

```
*)
FUNCTION conformant_external_file_protocol_24_3(
     s: SET [0:?] OF external_file_protocol): BOOLEAN;

REPEAT i := 1 TO SIZEOF(s);
     IF NOT(conformant_8859_1_protocol_24_3(s[i])
```



```
          OR conformant_http_protocol_24_3(s[i]))
     THEN
          RETURN(FALSE);
     END_IF;
END_REPEAT;

RETURN(TRUE);

END_FUNCTION; -- conformant_external_file_protocol_24_3
(*
```

M.3.6 is_correct_liim_24_3_application_value function

The **is_correct_liim_24_3_application_value** function checks that the **liim_id library_iim_identification** is compatible with the conformance classes associated to the LIMM 24-3.

EXPRESS specification:

```
*)
FUNCTION is_correct_liim_24_3_application_value(
     liim_id: library_iim_identification): BOOLEAN;

IF EXISTS(liim_id\data_exchange_specification_identification.
     application)
     AND
     ((liim_id\data_exchange_specification_identification.
          application[1] = '1')
               OR
     (liim_id\data_exchange_specification_identification.
          application[1] = '2'))
     AND
     ((liim_id\data_exchange_specification_identification.application
          LIKE '#')
               OR
     (liim_id\data_exchange_specification_identification.application
          LIKE '#E'))
THEN
     RETURN(TRUE);
ELSE
     RETURN(FALSE);
END_IF;

END_FUNCTION; -- is_correct_liim_24_3_application_value
(*
```

M.3.7 is_extended_liim_24_3_application_value function

The **is_extended_liim_24_3_application_value** function checks whether the **liim_id library_iim_identification** is associated to an extended conformance class.

EXPRESS specification:

```
*)
FUNCTION is_extended_liim_24_3_application_value(
      liim_id: library_iim_identification): BOOLEAN;

IF EXISTS(liim_id\data_exchange_specification_identification.
      application)
      AND
      (liim_id\data_exchange_specification_identification.
           application LIKE '#E')
THEN
      RETURN(TRUE);
ELSE
      RETURN(FALSE);
END_IF;
END_FUNCTION; -- is_extended_liim_24_3_application_value
(*

*)

END_SCHEMA; -- ISO13584_f_v_iim_conformance_schema

(*
```


Annex N
(normative)

Implementation method specific requirements for the library integrated information model 24-3

Conformance to the library integrated information model 24-3 shall be effected by one or more implementation methods. The implementation methods defines what types of exchange behaviour is required with respect to exchange protocols.

One implementation method is defined for the library delivery file: ISO 10303-21. The implementation methods for the possible external files referenced from the library delivery file and whose **external_file_protocol** belong to the standard data of the library integrated information model 24-3 are defined by the standard referenced in this external file protocol, possibly further specified as part of the description of the library integrated information model 24-3 standard data (see annex K).

The implementation methods for the possible external files referenced from the library delivery files and whose external file protocols are supplier-defined (xxE conformance classes) shall be based on the provisions, if any, contained in the referenced standard in the case of a **standard_protocol**. They shall be based on previous agreement between the library data suppliers and the library users in the case of a **non_standard_protocol**.

For the exchange structure, the file format of the library delivery file shall be encoded according to the syntax and EXPRESS language mapping defined in ISO 10303-21 for the schema defined in annex K of this part of ISO 13584. The header of the exchange structure shall identify use of this part of ISO 13584 by the schema name 'ISO13584_f_v_iim_schema'.

NOTE Identification of the library delivery file is done by separate agreement between the sender and the receiver and is outside the scope of this part of ISO 13584.

Annex O
(informative)

Logical description of the compiling process of ISO 13584-conformant dictionaries and libraries

A library description is designed by one library data supplier who is responsible for the **dictionary_element**s and **content_item**s provided in the ISO 13584-conformant library exchange context. Such a description is intended to be compiled by an ISO 13584-conformant LMS to make it usable by a library end-user.

A logical description of such a compiling process is therefore needed to define the consequences for the library end-user of the description choices done by the library data supplier.

A library description from one library data supplier may also contain BSUs or **dictionary_element**s that refer to other library data suppliers. These information elements are provided for convenience and their compiling process is outside the scope of the process described in this annex.

EXAMPLE Assume that a library data supplier wants to reference (for instance through an **a_posteriori_case_of** relationship) the dictionary defined in IEC 61360-4. Assume that some end-user libraries contain this dictionary, and some others do not. In order to decide to reference such a dictionary, or not, and to duplicate the definitions from IEC 61360-4, or not, the supplier shall know what would be the consequences for both kinds of end-user libraries.

The goal of this annex is to specify what shall be the result of a library compiling process however this process be performed.

NOTE 1 The logical process defined in this annex is only intended to specify the required result of a compiling process. The process itself is not standardised by this part of ISO 13584 and may be different from the logical description provided below.

An ISO 13584 exchange context, whether it contains only **dictionary_element** or it contains also a library specification, is modelled through one **dictionary** or **library** entity.

It is assumed in the following description of the compiling process that the user dictionary and/or library is also modelled through the same entities. It is also assumed that the dictionary and/or library to be compiled refers to a library integrated information model and to view exchange protocol(s) in some conformance classes that are supported by the receiving system.

NOTE 2 References to library integrated information models and to view exchange protocols are made by means of **library_iim_identification** and **view_exchange_protocol_identification** entities.

The design of the information models presented in this part of ISO 13584 are based on the following assumptions about the logical behaviour of the compiling process of a supplier library.

The compiling process is a two step process.

a) In the first step all the inverse pointers defined in the information model are solved. That is, the values of the attributes of the EXPRESS entities defined as INVERSE in the information model, and that are not represented in the exchange context, are computed. Moreover, inverse pointers from the **class_BSU**s to the **document_BSU**s and **table_BSU**s that reference these **class_BSU**s through their **name_scope** attributes are computed and created.

b) In the second step, if the system supports conformance classes defined as extended conformance classes (see clauses 16, 17 and 18 of this part of ISO 13584) the **external_file_protocol**s referenced by the **library** or **dictionary** entity are compared to the supported **external_file_protocol**s, and if the **library** or **dictionary** entity references not supported **external_file_protocol**s, the compiling process fails and the **library** or **dictionary** is not compiled; otherwise, the **library** or **dictionary** is compiled. The compiling process of the **library** or **dictionary** is a sequential process that consists in compiling successively each attribute of the **library** or **dictionary** and, for the LIST ordered attributes, each item of this

attribute following the LIST order, and for the SET-valued attributes, each item of this attribute in any order.

NOTE Only those items, provided by the responsible supplier of the library, need to be compiled.

Compiling any item consists of processing this item (inserting it or updating it in the user **library** or **dictionary**) and then in compiling all the items referenced directly or indirectly by this item, by direct or inverse pointers.

The processing of any item may fail. Compiling failure is not an error that stops the compiling process. It is an exception that shall be acknowledged as follows:

— the compiling process returns to the state it had before compiling the current **library** or **dictionary**-referenced item, and

— this **library** or **dictionary**-referenced item is skipped and the compiling process goes on until the end of the **library** or **dictionary**.

Processing of a **supplier_BSU** fails if:

— its **dictionary_element** is not provided in the supplier library and/or dictionary and is not present in the user library and/or dictionary.

Processing of a **program_library_BSU** fails if:

— its **dictionary_element** is not provided in the supplier library and/or dictionary and is not present in the user library and/or dictionary, or

— its **content_item** is not provided in the supplier library and/or dictionary and is not present in the user library and/or dictionary, or

— its version is greater than the corresponding BSU available in the user library and/or dictionary and its **dictionary_element** is not provided in the supplier library and/or dictionary, or

— its version is greater than the corresponding BSU available in the user library and/or dictionary and its **content_item** is not provided in the supplier library and/or dictionary.

Processing of a **linked_interface_program_protocol** fails if:

— the processing of any **program_library_BSU** it references in its **link_libraries** attribute failed.

Processing of a **class_BSU** fails if:

— its **dictionary_element** is not provided in the supplier library and/or dictionary and is not present in the user library and/or dictionary, or

— its version is greater than the version of the corresponding BSU available in the user library and/or dictionary and its **dictionary_element** is not provided, or

— its **dictionary_element** is provided in the supplier library and/or dictionary and the user library and/or dictionary contains a **content_item** corresponding to this BSU and a **content_item** corresponding BSU in the supplier library is not provided, or

— the processing of any BSU referred to, directly or indirectly either by the **class_BSU** (i.e., its **supplier_BSU** through its **name_scope**) or by its **class dictionary_element** failed, or

— the processing of any **linked_interface_program_protocol** referenced by the **used_protocols** attribute of the **model_class_extension content_item** failed.

Processing of a BSU referred to from a **class dictionary_element** fails if:

— its **dictionary_element** is not provided in the supplier library and/or dictionary and is not present in the user library and/or dictionary, or

— its version is greater than the version of the corresponding BSU available in the user library and/or dictionary and its **dictionary_element** is not provided, or

— its version is greater than the version of the corresponding BSU available in the user library and/or dictionary and the user library and/or dictionary contains a **content_item** corresponding to this BSU and a **content_item** corresponding BSU in the supplier library is not provided, or

— it is a **table_BSU** that has no corresponding BSU in the user library and/or dictionary and its **table_content** is not provided, or

— it is a **document_BSU** that has no corresponding BSU in the user library and/or dictionary and its **document dictionary_element** is not provided.

NOTE 3 Processing of **document_BSU** is not considered has having failed when

— either no **document_content** is provided, or

— the system is unable to compile the **document_content** because the **external_file_protocol**(s) it references is (are) not supported.

Processing of an **a_posteriori_semantic_relationship** fails if:

— the processing of either **class_BSU** referred to in the relationship failed.

When the processing of an item fails, the items referenced directly or indirectly by this item are not compiled and this item is skipped in the compiling process.

This assumption about the compiling process:

— ensures the feasibility of the compiling process of a **library** and/or **dictionary** whose integrated information model is one of the ones defined in clauses 16, 17 or 18 of this International Standard,

— ensures that when a reference hierarchy (such as the **dictionary** defined in IEC 61360-4) has already been compiled, a supplier library and/or dictionary can make reference to properties, data types and tables associated, through their **name_scope** attribute, to the reference hierarchy (i.e., they are *visible*) even if they are not referenced, through their **described_by** attribute or through a **class_BSU_relationship** by any class in this hierarchy (not required to be *applicable*).

Annex P
(informative)

Commented example of Parts Library physical files

This annex presents, on an example, an overview of the different resources and steps involved in the description of a parts library according to the ISO 13584 series.

The content of this annex is as follows:

— Clause 1: describes the example parts family and discusses the two different levels of description of such a family: dictionary definition and explicit/implicit library specification.

— Clause 2: explicit description of a parts library

  — Clause 2.1: outlines the main resource constructs involved in the physical file that defines this family.

  — Clause 2.2: outlines the main resource constructs involved in the physical file that defines a functional model class able to create a geometry view of this family.

  — Clause 2.3: presents the two resulting physical files conformant with ISO 10303-21.

— Clause 3: implicit description of a parts library.

  — Clause 3.1: outlines the main resource constructs involved in the physical file that defines this family.

  — Clause 3.2: outlines the main resource constructs involved in the physical file that defines a functional model class able to create a geometry view of this family.

  — Clause 3.3: presents the resulting physical files conformant with ISO 10303-21.

NOTE All the examples of physical files presented in clauses P.2.1, P2.2 and P.3.1, P3.2 are extracted from those presented respectively in clauses P.2.3 and P3.3. Therefore, the content of the entities referenced from these examples may be found (with the same entity names respectively in clauses P.2.3 and P3.3.

P.1 Capturing a parts family in ISO 13584

Figure P.1 illustrates the parts family intended to be described. It is a family of washers, denoted PAW, that is sold by a bearing supplier and that is used as a bearing in some mechanical contexts.

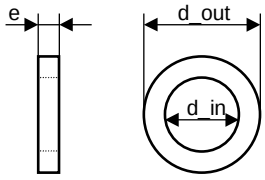


Figure P.1 — PAW family description

Such a parts family may be described at two levels of abstraction.

The dictionary level (**ISO13584_IEC61360_dictionary_schema** and **ISO13584_extended_dictionary_schema**) permits the description of the concepts of a part family, i.e., the supplier(s), the class(es), the property(ies), the table(s), etc. It defines what are the meaning and the value type of each property, in which classes the properties are visible, by which supplier the classes are specified, etc. Such a data **dictionary_element** may be done both for the general model

description (what is this family of parts) and for the functional model description (what kind of representations may be defined for this family of parts).

The library level (**ISO13584_library_content_schema**) permits the restriction of the values domains of the properties to implicitly define the allowed instances of the parts family. This level is for instance useful when describing a paper catalogue, because all the allowed values of the different parts are clearly specified through tables and algebraic functions.

These two levels of abstraction have to be represented in two different ways.

The following figure (Figure P.2) presents an example of instances of a family that is only defined at the dictionary level. The permitted instances are those where the values of the properties describing the part belong to the type defined in the data dictionary.

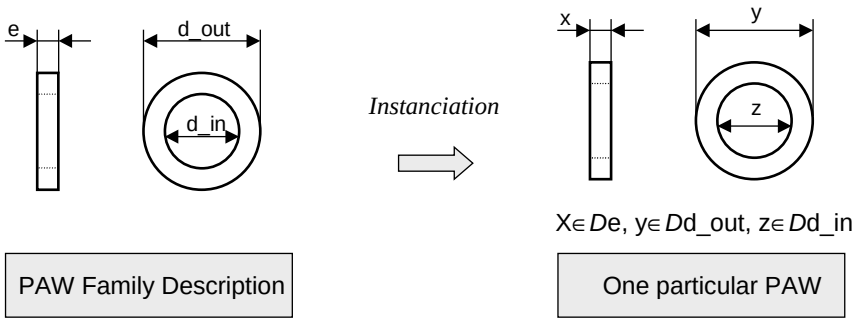


Figure P.2 — Instance of a dictionary description

P.1.1 Explicit modeling approach

Figure P.3 describes and presents an example of a family that is associated with a two-fold description: **dictionary_element** and library specification. The allowed instances are those where the values of the properties describing the part belong to the explicit list of authorized tuples of values defined in the library specification of a part. This approach is called the explicit representation of a library content.

d_in	e	d_out
10	1	15
11	1	16.5
13	2	19.5
17	3	25.5
19	4	28.5

Figure P.3 — Explicit representation of a dictionary description

P.1.2 Implicit modeling approach

The library content may also be described by specifying the allowed instances where the values of the properties describing the part belong to the implicit list of authorized tuples of values defined in the library specification of a part. This approach is called the implicit representation of a library content. It is illustrated in Figure P.4.

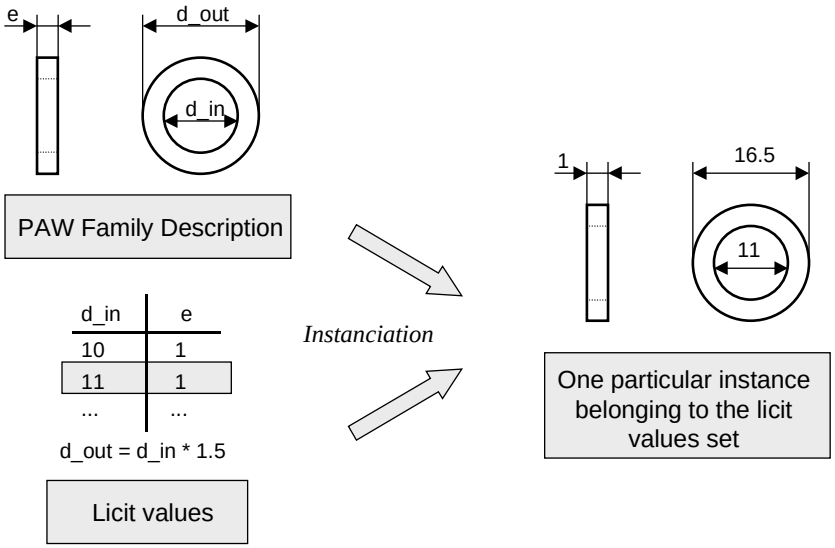


Figure P.4 — Implicit representation of a dictionary description

P.2 Capturing a parts family in ISO 13584 using the explicit representation of a library content

P.2.1 Description of the PAW parts family

This description is two-fold. First the concept of PAW and of its properties are defined. Second, the allowed instances are precisely (but implicitly) defined.

P.2.1.1 Dictionary description: the BSU mechanism

The description of a data dictionary according to the **ISO13584_IEC61360_dictionary_schema** schema requires the specification of what are the identifiers of the different concepts (called basic semantic unit: BSU) involved in the parts family definition. These identifiers define unambiguously and universally each concept within an ISO 13584-compliant data dictionary.

The following example (Figure P.5) outlines the resources used for the specification of these identifiers:

```
/*BSU for supplier */
/* The code of the supplier must be defined according to ISO13584-26: Supplier identification
Here, the code doesn't follow the ISO 13584-26 requirements, because the supplier code is not
known at the moment */
#20 = SUPPLIER_BSU ('INA', *);

/* BSU for component_class */
/* The class BSUs defines the identification of the various classes, and who is the supplier that is
responsible of the class definitions */
#50 = CLASS_BSU ('BEARING', '001', #20);
#60 = CLASS_BSU ('PAW', '001', #20);

/* BSU for properties */
/* The property BSU defines the identifications of the properties and the class where these
properties are visible */
```



```
#90 = PROPERTY_BSU ('d_in', '001', #50);
#100 = PROPERTY_BSU ('d_out', '001', #50);
#110 = PROPERTY_BSU ('e', '001', #60);
```

Figure P.5 — Identifiers of the concepts involved in the PAW family

P.2.1.2 Dictionary description: the dictionary element definition

A BSU only identifies a concept. A **dictionary_element** provides a computer-sensible and human-readable definition of the concept. This relationship between these two levels is presented in Figure P.6.

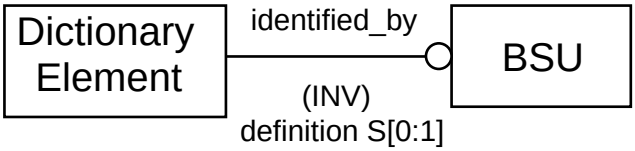


Figure P.6 — The BSU / Dictionary element relationship

The following figure (Figure P.7) outlines the main structure of the **dictionary_element**s corresponding to the previous basic semantic units identifiers.

```
  /* Supplier definition */
  #21=SUPPLIER_ELEMENT(
            #20, /* reference to its BSU */
            $, '001',
            #22, #23); /* organisation and address */
  /* Property definition */
  /* d_in */
  #91=NON_DEPENDENT_P_DET(
            #90, /* reference to its BSU */
            $, '001',
            #92, /* item_names (human-readable name of the concept,
                      with possible translations) */
            TEXT('inner diameter'), $, $, $, $, (), $,
            'TO3', /* the data element type classification,
                      according to ISO 31*/
            #93, /* the specific data type of the property
                 (not represented: measure in mm) */
            $);
/* d_out */
#101 = NON_DEPENDENT_P_DET(
            #100, $, '001', #102, TEXT('outer diameter'), $,
            $, $, $, (), $, 'TO3', #93, $);
/* e */
#111 = NON_DEPENDENT_P_DET(
            #110, $, '001', #112, TEXT('thickness'), $, $, $,
            $, (), $, 'TO3', #93, $);
  /* Class definition*/
  /* Part class */
  #71=COMPONENT_CLASS(
            #50, /* reference to its BSU */
            $, '001',
            #72, /* item_names */
```



```
            TEXT('Class associated...'), /* definition */
            $, $, $, $,
            (#90, #100),  /* the list of the properties that may be
            used to describe an instance of this class
            (applicability of the properties) */
            (), $, (),(), $);
/* PAW class */
  #81 = COMPONENT_CLASS(#60, $, '001', #82, TEXT('Class associated to the
            PAW part family'), $, $, $, #50, (#110), (), $, (),(), $);
```

Figure P.7 — Dictionary_element of the concepts involved in the PAW family

P.2.1.3 Explicit library specification: description of the class extension

The PAW family being a catalogue defined family, only some instances are really allowed. This set of instances is explicitly defined through:

a) the choice (by the library data supplier) of the identification characteristics of the family (in this example: the **d_in** property),

b) the explicit description of each of the components delivered by the parts supplier.

P.2.1.3.1 Overall architecture

The relationship between a dictionary element and an associated library specification **content_item** is outlined in Figure P.8:

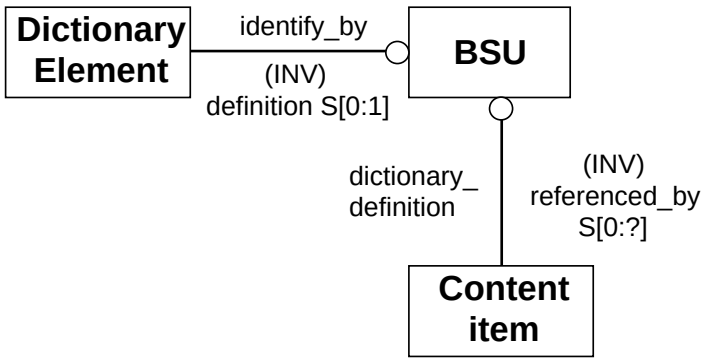


Figure P.8 — The Dictionary Element / Library Content relationship

Each content item shall be consistent with the corresponding dictionary element definition (respect of the type / value relationship).

P.2.1.3.2 Instances description

Each class instance is described by the list of its property values. Figure P.9 gives the description, by extension of one instance corresponding to the part family for which **d_in** equals 10, **e** equals 1 and **d_out** equals 15.

```
/* Extension of a library component */
  #8100=LIB_COMPONENT_INSTANCE(
            #60, /* the BSU associated to the class which
            the current instance is an instance of */
```



```
            (#8101, #8102, #8103), /* the property values */
            (), $, $, $, $, $);
/* Property values of the extension of a class */
  #8101=PROPERTY_VALUE(REAL_VALUE(10.0), #90);
  #8102=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
  #8103=PROPERTY_VALUE(REAL_VALUE(15.0), #110);
```

Figure P.9 — Description of one particular instance of the PAW parts family

P.2.1.3.2 Explicit class extension

The extension of a class is given by the set of instances of a class. Each instance of a class contains a set of property values that correspond to the values of the properties belonging to the part family described by this instance. Figure P.10 describes the extension of a class by an explicit modeling.

```
/* Dictionary extension */
/* Extension of a class */
  #8000= EXPLICIT_ITEM_CLASS_EXTENSION(
            #60,   /*Reference to the BSU */
            (), (), (), '001', '001', (),
            (#90), /* an identification property: d_in */
            (#8100, #8200, #8300, #8400, #8500),  /* the extension of
            class given by a list of class instances */
            .T., $, $, (), $, (), ());
```

Figure P.10 — Description of the PAW explicit class extension

P.2.2 Description of geometric representations for the PAW parts family

P.2.2.1 Description of the functional model dictionary properties

The description of the PAW family is identical to the one described and commented in the previous clause, except that supplier elements have been introduced in order to describe the view and the geometry suppliers (Figure P.11).

```
/* BSU for suppliers */
#20=SUPPLIER_BSU('INA', *);  /* Note: INA code unknown */
          /* Supplier of the part family */
#30=SUPPLIER_BSU('9/19860073600021', *);
          /* LISI/ENSMA code in the coding scheme ICD=0009: SIRET
          number */
          /* Supplier of the functional model class */

#40=SUPPLIER_BSU('0112/1///13584_101_1', *);
          /*Identification of ISO 13584-101 according to ISO 13584-26*/
          /* Supplier of the functional view */
```

Figure P.11 — Description of the supplier identifiers

The two **class_BSU**s **PAW_Gemetry** and **basic_geometry** identify respectively the classes corresponding to the functional model and of the functional view (Figure P.12).


```
/* BSU for classes */
#50=CLASS_BSU('Bearing', '001', #20);
#60=CLASS_BSU('PAW', '001', #20);
#130=CLASS_BSU('PAW_Geometry', '001', #30);
#140=CLASS_BSU('basic_geometry', '001', #40);
```

Figure P.12 — Description of the class identifiers

The following **property_BSU** describes the properties of the parts family PAW as defined in the previous clause (Figure P.13).

```
/* BSU for properties */
#90=PROPERTY_BSU('d_in', '001', #50);
#100=PROPERTY_BSU('d_out', '001', #50);
#110=PROPERTY_BSU('e', '001', #50);
```

Figure P.13 — Description of the general model property identifiers

The definition of the geometry for a given part, and particularly for the part family PAW requires the definition of representation properties. These properties are defined as for part family properties through the BSU mechanism (Figure P.14).

```
/* BSU for properties */
#150=PROPERTY_BSU('geometry_level', '001', #140);
#160=PROPERTY_BSU('detail_level', '001', #140);
#170=PROPERTY_BSU('side', '001', #140);
#180=PROPERTY_BSU('prg', '001', #130);
#200=PROPERTY_BSU('variant', '001', #140);
#210=PROPERTY_BSU('unreg_variant', '001', #140);
```

Figure P.14 — Description of the functional model / view property identifiers

The following **supplier_element** describes LISI/ENSMA as supplier. It will be used as supplier of a functional model class (Figure P.15).

```
/* supplier description */
#31=SUPPLIER_ELEMENT(#30, $, '001', #32, #33);
#32=ORGANIZATION('LISI/ENSMA', 'LISI/ENSMA', '');
#33=ADDRESS($, $, $, $, $, $, $, 'FRANCE', $, $, $, $);
```

Figure P.15 — Functional model supplier description

Data type elements associated to representation properties are **representation_P_DET**s. For example, the following data element (Figure P.16) describes the variable allowing to refer programs (**prg**).

```
/* prg */
#91=REPRESENTATION_P_DET(#180,
```

```
      $, '001', #92,
      TEXT('variable used to reference geometry programs'),
      $, $, $, $, (), $, 'A58', #93, $);
#92=ITEM_NAMES(LABEL('related program'), (), LABEL(''), $, $);
#93=PROGRAM_REFERENCE_TYPE((
      'ISO13584_F_M_IIM_LIBRARY_IMPLICIT_SCHEMA.PROGRAM_REFERENCE'));
```

Figure P.16 — Property description for referencing programs

P.2.2.2 Description of the geometric functional model

We assume now that some library data supplier wants to provide a geometric representation for all the instances of the PAW family that are described in an explicit manner (by extension). This requires the description of a functional model class.

A functional model class is intended to represent different perspectives of the different parts described in the general model class. A functional model class has to be described like a general model class, i.e., through a class definition and through a dictionary extension (library specification).

A **functional_model_class** describes a particular view (is-view-of relationship) of a given parts family (described as a general model class), according to the point of view specified by a **functional_view_class**.

In the example, it will be defined a functional model class representing some kind of geometry (specified by) a **functional_view_class** for the PAW parts family.

A **fm_class_view_of** is a functional model class that refers to a well defined general model class (here the class that models the PAW family) and that provides a particular kind of representation (specified by a **functional_view_class**) for this general model class.

A functional model class is not required to provide representation for all the values of the functional view class view control variables. The range of supported values is specified by **view_control_variable_range** as shown in Figure P.17.

```
/* v_c_v range */
#155=VIEW_CONTROL_VARIABLE_RANGE(#150, 1, 1);
#165=VIEW_CONTROL_VARIABLE_RANGE(#160, 2, 2);
#175=VIEW_CONTROL_VARIABLE_RANGE(#170, 1, 6);
#205=VIEW_CONTROL_VARIABLE_RANGE (#200, 1, 1);
#215=VIEW_CONTROL_VARIABLE_RANGE (#210, 0, 0);
```

Figure P.17 — View control variables range definition

In our example, the functional model class only provides 2D views (range 1..1, for #150 that is ‘geometry_level’), with standard representation (range 2..2, for #160 that is ‘detail-level’) for all the sides from ‘front’ to ‘bottom’ (range 1..6, for # 170 that is ‘side’).

Moreover, to be able to create the geometry, the **fm_class_view_of** needs to import some properties of the PAW family. The **dictionary_element** of the **fm_class_view_of** is presented in the Figure P.18.

The following instance describes the **is_view_of** relationship through a **fm_class_view_of** class supplied by LISI/ENSMA (Figure P.18).

```
/* Functional model class view_of definition*/
```



```
#71=FM_CLASS_VIEW_OF(
      #130,     /* reference to BSU */
      $, '001', #72,  /* item names */
      TEXT('Explicit ...'), $, $, $, $,
      (#180), /* BSU of ‘prg’ and ‘required_side’ properties */
      (), *, *, *, *, *,
      #140, /* the created view (reference to the BSU of the
      functional_view_class */
      (#155, #165, #175, #205, #215), /* the vcv ranges */
      (#150, #160, #170, #200, #210), /* vcv’s imported from the
      functional view class */
      (), (), (), (), (), (), (), (),
      #60, /* is_view_of relationship. Reference to the functional
                model */
      (#90, #100, #110),/* imported properties from the general model
*/
       (),(),());
#72=ITEM_NAMES(LABEL('Functional model class of PAW'), (),
      LABEL('fm class of PAW'), $, $);
```

Figure P.18 — Specification of the view created by a functional model class

P.2.2.3 Library specification of the functional model class

An **explicit_functional_model_class_extension** is the **content_item** that constitutes the library specification a **functional_model_class** and **fm_class_view_of** subtype). It gives the set of the properties that need to be valued in the context of an instance of such a class.

In the case of an explicit representation, the instances of a functional model are explity enumerated. In the following **explict_functional_model_extension** instance, the instances of a functional model are described by **lib_F_model_instance**s in the range 3000 to 3450.

```
/* Functional model class extension*/
#1300=EXPLICIT_FUNCTIONAL_MODEL_CLASS_EXTENSION(
      #130,
      (#2501, #2502, #2503,#2504, #2505, #2506), /* the set of program
      references. The programs which allow to display geometry.*/
      (#7), (#12), '001', '001', (), (),
      (#90, #170),  /* properties needed to display the geometry */
      (#3000, #3010, #3020, #3030, #3040, #3050,
      #3100, #3110, #3120, #3130, #3140, #3150,
      #3200, #3210, #3220, #3230, #3240, #3250,
      #3300, #3310, #3320, #3330, #3340, #3350,
      #3400, #3410, #3420, #3430, #3440, #3450), /* The extension of
all
      the instances of a functional model. They are given by the
      lib_f_model_instances */
      .T., $, (#90, #100, #110), #180, $, $, (), $);
```

Figure P.19 — Description by extension of the instances of a functional a functional model

According to the previous instance, references to programs that describe geometry representation are performed. These **program_reference**s refer themselves to external files *PAW_p1.for* .. *PAW_p6.for* containing geometry programs written in the FORTRAN language according to view exchange protocol ISO 13584-101 (Figure P.20).

```
/* Reference to programs which display geometry */
#2501=PROGRAM_REFERENCE(#7, #2601, 'Add1_PAW', 'PAW_p1',
     (#90, #100, #110), $, $);
#2502=PROGRAM_REFERENCE(#7, #2602, 'Add2_PAW', 'PAW_p2', (#90, #100,
#110), $, $);
...
#2601=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2701));
#2602=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2702));
...
#2701=LANGUAGE_SPECIFIC_CONTENT((#2801), #2801, $);
#2702=LANGUAGE_SPECIFIC_CONTENT((#2802), #2802, $);
...
/* Description of the source files for geometry */
#2801=EXTERNAL_FILE_UNIT('PAW_p1.for', '7bit');
#2802=EXTERNAL_FILE_UNIT('PAW_p2.for', '7bit');
...
```

Figure P.20 — References to FORTRAN programs that display geometry.

Instances of functional models described by extension, describe themselves their property values by extension. Figure P.21 shows one instance of functional model which describes by extension the values of its properties.

```
/* Description of library functional model instance by extension */
#3000=LIB_F_MODEL_INSTANCE(#130, (#3001, #3002, #3003, #3004, #3005,
     #3006, #3007), ());
#3000=LIB_F_MODEL_INSTANCE(#130, (#3001, #3008, #3009, #3002, #3003,
     #3004, #3005, #3006, #3007), ());
#3001=PROPERTY_VALUE(REAL_VALUE(10.0), #90);
#3008=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#3009=PROPERTY_VALUE(REAL_VALUE(15.0), #110);
#3002=PROPERTY_VALUE(1, #170);
#3003=PROPERTY_VALUE(1, #150);
#3004=PROPERTY_VALUE(2, #160);
#3005=PROPERTY_VALUE(1, #200);
#3006=PROPERTY_VALUE(0, #210);
#3007=PROPERTY_VALUE(#2501, #180);
```

Figure P.21 — The BSU / Dictionary element relationship

P.2.3 Resulting Physical files

In this clause, the complete examples of physical files are provided. The first physical file, conformant with LIBRARY INTEGRATED INFORMATION MODEL 24-1 defines the PAW family. The second physical file, conformant with LIBRARY INTEGRATED INFORMATION MODEL 24-2 and VIEW EXCHANGE PROTOCOL DIS 101 (for the definition of the geometry view) defines the 2D geometry of the PAW family by means of programs conformant with ISO 13584-31.

P.2.3.1 Example of a general model

This subclause contains the complete physical file presented in clause P.2.1.

```
  */
ISO-10303-21;
HEADER;

FILE_DESCRIPTION (('THIS IS AN EXAMPLE OF AN EXPLICIT GENERAL MODEL'),
'2');
FILE_NAME('P24_gm_explicit.p21',
          '2001-07-30T17:38:14',
          (''),
          ('LISI/ENSMA'),
          'ECCO RUNTIME SYSTEM BUILT-IN PREPROCESSOR V2.3.3',
          'ECCO RUNTIME SYSTEM V2.3.3',
          '');
FILE_SCHEMA (('ISO13584_G_M_IIM_LIBRARY_IMPLICIT_SCHEMA'));
ENDSEC;
DATA;

  /* Global library description */
#2 = LIBRARY_IN_STANDARD_FORMAT ($, $, $, $, (), #20, #11, (), (),
(#20), (#50, #60),
(), #3, $, $, ());
#3 = ITEM_NAMES (LABEL('Explicit general model example'), (), LABEL(''),
$, $);
#10 = GLOBAL_LANGUAGE_ASSIGNMENT ('en');
#11=LIBRARY_IIM_IDENTIFICATION($, 'IS', 'ISO_13584_24_1', 2001, '3', $,
());

  /* DICTIONARY DESCRIPTION */
  /*BSU for supplier */
  #20 = SUPPLIER_BSU ('INA', *);

  /* BSU for component_class */
  #50 = CLASS_BSU ('BEARING', '001', #20);
  #60 = CLASS_BSU ('PAW', '001', #20);

  /* BSU for properties */
  #90 = PROPERTY_BSU ('d_in', '001', #50);
  #100 = PROPERTY_BSU ('d_out', '001', #50);
  #110 = PROPERTY_BSU ('e', '001', #60);

  /* Dictionary properties description */
  /* supplier description */
  #21 = SUPPLIER_ELEMENT (#20, $, '001', #22, #23);
  #22 = ORGANIZATION ($, 'INA', '');
  #23 = ADDRESS ($, $, $, $, $, $, $, 'GERMANY', $, $, $, $);

  /* d_in */
```

```
 #91 = NON_DEPENDENT_P_DET(#90, $, '001', #92, TEXT('inner diameter'),
 $,
 $, $, $, (), $, 'TO3', #93, $);
 #92 = ITEM_NAMES(LABEL('inner diameter'), (), LABEL(''), $, $);
 #93 = REAL_MEASURE_TYPE('NR2..3.3', #94);
 #94 = DIC_UNIT(#95, $);
 #95 = SI_UNIT(*, .MILLI., .METRE.);
 /* d_out */
 #101 = NON_DEPENDENT_P_DET (#100, $, '001', #102, TEXT('outer
 diameter'),
  $, $, $, $, (), $, 'TO3', #93, $);
 #102 = ITEM_NAMES (LABEL('outer diameter'), (), LABEL(''), $, $);
 #103 = REAL_MEASURE_TYPE ('NR2..3.3', #104);
 #104 = DIC_UNIT (#105, $);
 #105 = SI_UNIT (*, .MILLI., .METRE.);

 /* e */
 #111 = NON_DEPENDENT_P_DET (#110, $, '001', #112, TEXT('thickness'), $,
  $, $, $, (), $, 'TO3', #93, $);
 #112 = ITEM_NAMES (LABEL('thickness'), (), LABEL(''), $, $);
 #113 = REAL_MEASURE_TYPE ('NR2..3.3', #114);
 #114 = DIC_UNIT (#115, $);
 #115 = SI_UNIT (*, .MILLI., .METRE.);

 /* Dictionary class description */
 /* Part class */
 #71 = COMPONENT_CLASS (#50, $, '001', #72, TEXT('Class associated to
 the
 generic bearing family'), $, $, $, $, (#90, #100), (), $, (),(), $);
 #72 = ITEM_NAMES (LABEL('Generic bearing family'), (), LABEL('Bearing
 family'), $,
 $);

 /* PAW class */
#81 = COMPONENT_CLASS (#60, $, '001', #82, TEXT('Class associated to the
PAW part family'), $, $, $, #50, (#110), (), $, (),(), $);
#82 = ITEM_NAMES (LABEL('PAW family'), (), LABEL('PAW'), $, $);

 /* Dictionary extension */
#8000=
EXPLICIT_ITEM_CLASS_EXTENSION(#60,(),(),(),'001','001',(),(),(#90),(
#8100,#8200,#8300,#8400,#8500),.T.,$,$,(),$,(),());
```



```
#8100=LIB_COMPONENT_INSTANCE(#60, (#8101, #8102, #8103), (), $, $, $, $,
.T., $);
#8101=PROPERTY_VALUE(REAL_VALUE(10.0), #90);
#8102=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#8103=PROPERTY_VALUE(REAL_VALUE(15.0), #110);

#8200=LIB_COMPONENT_INSTANCE(#60, (#8201, #8202, #8203), (), $, $, $, $,
.T., $);
#8201=PROPERTY_VALUE(REAL_VALUE(11.0), #90);
#8202=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#8203=PROPERTY_VALUE(REAL_VALUE(16.5), #110);

#8300=LIB_COMPONENT_INSTANCE(#60, (#8301, #8302, #8303), (), $, $, $, $,
.T., $);
#8301=PROPERTY_VALUE(REAL_VALUE(13.0), #90);
#8302=PROPERTY_VALUE(REAL_VALUE(2.0), #100);
#8303=PROPERTY_VALUE(REAL_VALUE(19.5), #110);

#8400=LIB_COMPONENT_INSTANCE(#60, (#8401, #8402, #8403), (), $, $, $, $,
.T., $);
#8401=PROPERTY_VALUE(REAL_VALUE(17.0), #90);
#8402=PROPERTY_VALUE(REAL_VALUE(3.0), #100);
#8403=PROPERTY_VALUE(REAL_VALUE(25.5), #110);

#8500=LIB_COMPONENT_INSTANCE(#60, (#8501, #8502, #8503), (), $, $, $, $,
.T., $);
#8501=PROPERTY_VALUE(REAL_VALUE(19.0), #90);
#8502=PROPERTY_VALUE(REAL_VALUE(4.0), #100);
#8503=PROPERTY_VALUE(REAL_VALUE(28.5), #110);

  ENDSEC;
  END-ISO-10303-21;
  /*
```

P.2.3.2 Example of a functional model

This subclause contains a physical file of an explicitly described functional model that defines the 2D geometry of the PAW family by means of programs conformant with ISO 13584-31.

```
  */
  ISO-10303-21;
  HEADER;
  FILE_DESCRIPTION(('PLIB EXPLICIT FUNCTIONAL MODEL EXAMPLE 1'), '1');
  FILE_NAME('P25_fm_explicit_p101.p21',
            '2001-07-30T18:38:14',
            (''),
            ('LISI/ENSMA'),
            'ECCO RUNTIME SYSTEM BUILT-IN PREPROCESSOR V2.3.3',
            'ECCO RUNTIME SYSTEM V2.3.3',
            '');
  FILE_SCHEMA(('ISO13584_F_M_IIM_LIBRARY_IMPLICIT_SCHEMA'));
```

```
ENDSEC;

DATA;

/* Global library description */
#2=LIBRARY_IN_STANDARD_FORMAT($, $, $, $, (), #30, #11, (#7), (#12),
(#20, #30,
#40), (#50, #60, #140, #130), (), #3, $, $, ());
#3=ITEM_NAMES(LABEL('Explicit functional model: Geometry'), (),
LABEL('Geometry'), $,
$);
#6=ORGANIZATION('LISI/ENSMA', 'LISI/ENSMA', '');
#7=STANDARD_SIMPLE_PROGRAM_PROTOCOL(#6, $, 'ISO_IS_13584_31',
'001', $, #8, $, 'FORTRAN', .SOURCE., $, $, $);
#8=ITEM_NAMES(LABEL('Geometric prog. interface'), (),
LABEL('ISO_IS_13584_31'), $, $);
#11=LIBRARY_IIM_IDENTIFICATION($, 'IS', 'ISO_13584_24_2', 2001, '3', $,
());
#12=VIEW_EXCHANGE_PROTOCOL_IDENTIFICATION($, 'IS', 'ISO13584_101',
2001, '2D', $,
(#7), $);
#10=GLOBAL_LANGUAGE_ASSIGNMENT('en');

/* DICTIONARY DESCRIPTION */
/*BSU for suppliers */
#20=SUPPLIER_BSU('INA', *);  /* INA code: parts family supplier*/
#30=SUPPLIER_BSU('9/19860073600021', *);
/* LISI/ENSMA code in the coding scheme ICD=0009: SIRET number */
#40=SUPPLIER_BSU('0112/1///13584_101_1', *);
/* Identification of ISO 13584-101 according to ISO 13584-26 */

/* BSU for component_class */
#50=CLASS_BSU('Bearing', '001', #20);
#60=CLASS_BSU('PAW', '001', #20);
#130=CLASS_BSU('PAW_Geometry', '001', #30);
#140=CLASS_BSU('basic_geometry', '001', #40);

/* BSU for properties */
#90=PROPERTY_BSU('d_in', '001', #50);
#100=PROPERTY_BSU('d_out', '001', #50);
#110=PROPERTY_BSU('e', '001', #50);
#150=PROPERTY_BSU('geometry_level', '001', #140);
#160=PROPERTY_BSU('detail_level', '001', #140);
#170=PROPERTY_BSU('side', '001', #140);
#180=PROPERTY_BSU('prg', '001', #130);
#200=PROPERTY_BSU('variant', '001', #140);
#210=PROPERTY_BSU('unreg_variant', '001', #140);

/* v_c_v range */
#155=VIEW_CONTROL_VARIABLE_RANGE(#150, 1, 1);
#165=VIEW_CONTROL_VARIABLE_RANGE(#160, 2, 2);
#175=VIEW_CONTROL_VARIABLE_RANGE(#170, 1, 6);
```



```
#205=VIEW_CONTROL_VARIABLE_RANGE (#200, 1, 1);
#215=VIEW_CONTROL_VARIABLE_RANGE (#210, 0, 0);

/* supplier description */
#31=SUPPLIER_ELEMENT(#30, $, '001', #32, #33);
#32=ORGANIZATION('LISI/ENSMA', 'LISI/ENSMA', '');
#33=ADDRESS($, $, $, $, $, $, $, 'FRANCE', $, $, $, $);

/* Dictionary properties description */
/* prg */
#91=REPRESENTATION_P_DET (#180, $, '001', #92, TEXT('variable used to
reference geometry programs'), $, $, $, $, (), $, 'A58', #93, $);
#92=ITEM_NAMES (LABEL('related program'), (), LABEL(''), $, $);
#93=PROGRAM_REFERENCE_TYPE
(('ISO13584_F_M_IIM_LIBRARY_IMPLICIT_SCHEMA.PROGRAM_REFERENCE'));

/* Dictionary class description */
/* Functional model class view_of definition*/
#71=FM_CLASS_VIEW_OF(#130, $, '001', #72, TEXT('Explicit functional
model
class describing the 2d standard geometry of PAW'), $, $, $, $, (#180),
(), *, *, *, *, *, #140, (#155, #165, #175, #205, #215), (#150, #160,
#170, #200, #210),
(), (), (), (), (), (), (), (), #60, (#90, #100, #110), (),(),());
#72=ITEM_NAMES(LABEL('Functional model class of PAW'), (), LABEL('fm
class of PAW'), $, $);

/* LIBRARY DESCRIPTION */

#1300=EXPLICIT_FUNCTIONAL_MODEL_CLASS_EXTENSION(#130, (#2501, #2502,
#2503,
#2504, #2505, #2506), (#7), (#12), '001', '001', (), (),
(#90, #100, #110, #170),
(#3000, #3010, #3020, #3030, #3040, #3050,
#3100, #3110, #3120, #3130, #3140, #3150,
#3200, #3210, #3220, #3230, #3240, #3250,
#3300, #3310, #3320, #3330, #3340, #3350,
#3400, #3410, #3420, #3430, #3440, #3450),
.T., $, (#90, #100, #110), #180, $, $, (), $);

#2501=PROGRAM_REFERENCE(#7, #2601, 'Add1_PAW', 'PAW_p1', (#90, #100,
#110), (), ());
#2502=PROGRAM_REFERENCE(#7, #2602, 'Add2_PAW', 'PAW_p2', (#90, #100,
#110), (), ());
#2503=PROGRAM_REFERENCE(#7, #2603, 'Add3_PAW', 'PAW_p3', (#90, #100,
#110), (), ());
#2504=PROGRAM_REFERENCE(#7, #2604, 'Add4_PAW', 'PAW_p4', (#90, #100,
#110), (), ());
#2505=PROGRAM_REFERENCE(#7, #2605, 'Add5_PAW', 'PAW_p5', (#90, #100,
#110), (), ());
```

```
#2506=PROGRAM_REFERENCE(#7, #2606, 'Add6_PAW', 'PAW_p6', (#90, #100,
#110), (), ());

#2601=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2701));
#2602=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2702));
#2603=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2703));
#2604=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2704));
#2605=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2705));
#2606=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2706));
#2701=LANGUAGE_SPECIFIC_CONTENT((#2801), #2801, $);
#2702=LANGUAGE_SPECIFIC_CONTENT((#2802), #2802, $);
#2703=LANGUAGE_SPECIFIC_CONTENT((#2803), #2803, $);
#2704=LANGUAGE_SPECIFIC_CONTENT((#2804), #2804, $);
#2705=LANGUAGE_SPECIFIC_CONTENT((#2805), #2805, $);
#2706=LANGUAGE_SPECIFIC_CONTENT((#2806), #2806, $);
#2801=EXTERNAL_FILE_UNIT('PAW_p1.for', '7bit');
#2802=EXTERNAL_FILE_UNIT('PAW_p2.for', '7bit');
#2803=EXTERNAL_FILE_UNIT('PAW_p3.for', '7bit');
#2804=EXTERNAL_FILE_UNIT('PAW_p4.for', '7bit');
#2805=EXTERNAL_FILE_UNIT('PAW_p5.for', '7bit');
#2806=EXTERNAL_FILE_UNIT('PAW_p6.for', '7bit');

/*
     GM      |                          FV                            | FM
-------------------------------------------------------------------------
d_in d_out e | side | geom_level | det_level | var | unreg_var | prg

*/

#3000=LIB_F_MODEL_INSTANCE(#130, (#3001, #3008, #3009, #3002, #3003,
#3004, #3005, #3006, #3007), ());
#3001=PROPERTY_VALUE(REAL_VALUE(10.0), #90);
#3008=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#3009=PROPERTY_VALUE(REAL_VALUE(15.0), #110);
#3002=PROPERTY_VALUE(INTEGER(1), #170);
#3003=PROPERTY_VALUE(INTEGER(1), #150);
#3004=PROPERTY_VALUE(INTEGER(2), #160);
#3005=PROPERTY_VALUE(INTEGER(1), #200);
#3006=PROPERTY_VALUE(INTEGER(0), #210);
#3007=PROPERTY_VALUE(#2501, #180);
#3010=LIB_F_MODEL_INSTANCE(#130, (#3011, #3018, #3019, #3012, #3013,
#3014, #3015, #3016, #3017), ());
#3011=PROPERTY_VALUE(REAL_VALUE(10.0), #90);
#3018=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#3019=PROPERTY_VALUE(REAL_VALUE(15.0), #110);
#3012=PROPERTY_VALUE(INTEGER(2), #170);
#3013=PROPERTY_VALUE(INTEGER(1), #150);
#3014=PROPERTY_VALUE(INTEGER(2), #160);
#3015=PROPERTY_VALUE(INTEGER(1), #200);
#3016=PROPERTY_VALUE(INTEGER(0), #210);
#3017=PROPERTY_VALUE(#2502, #180);
```



```
#3020=LIB_F_MODEL_INSTANCE(#130, (#3021, #3028, #3029, #3022, #3023,
#3024, #3025, #3026, #3027), ());
#3021=PROPERTY_VALUE(REAL_VALUE(10.0), #90);
#3028=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#3029=PROPERTY_VALUE(REAL_VALUE(15.0), #110);
#3022=PROPERTY_VALUE(INTEGER(3), #170);
#3023=PROPERTY_VALUE(INTEGER(1), #150);
#3024=PROPERTY_VALUE(INTEGER(2), #160);
#3025=PROPERTY_VALUE(INTEGER(1), #200);
#3026=PROPERTY_VALUE(INTEGER(0), #210);
#3027=PROPERTY_VALUE(#2503, #180);
#3030=LIB_F_MODEL_INSTANCE(#130, (#3031, #3038, #3039, #3032, #3033,
#3034, #3035, #3036, #3037), ());
#3031=PROPERTY_VALUE(REAL_VALUE(10.0), #90);
#3038=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#3039=PROPERTY_VALUE(REAL_VALUE(15.0), #110);
#3032=PROPERTY_VALUE(INTEGER(4), #170);
#3033=PROPERTY_VALUE(INTEGER(1), #150);
#3034=PROPERTY_VALUE(INTEGER(2), #160);
#3035=PROPERTY_VALUE(INTEGER(1), #200);
#3036=PROPERTY_VALUE(INTEGER(0), #210);
#3037=PROPERTY_VALUE(#2504, #180);
#3040=LIB_F_MODEL_INSTANCE(#130, (#3041, #3048, #3049, #3042, #3043,
#3044, #3045, #3046, #3047), ());
#3041=PROPERTY_VALUE(REAL_VALUE(10.0), #90);
#3048=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#3049=PROPERTY_VALUE(REAL_VALUE(15.0), #110);
#3042=PROPERTY_VALUE(INTEGER(5), #170);
#3043=PROPERTY_VALUE(INTEGER(1), #150);
#3044=PROPERTY_VALUE(INTEGER(2), #160);
#3045=PROPERTY_VALUE(INTEGER(1), #200);
#3046=PROPERTY_VALUE(INTEGER(0), #210);
#3047=PROPERTY_VALUE(#2505, #180);
#3050=LIB_F_MODEL_INSTANCE(#130, (#3051, #3058, #3059, #3052, #3053,
#3054, #3055, #3056, #3057), ());
#3051=PROPERTY_VALUE(REAL_VALUE(10.0), #90);
#3058=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#3059=PROPERTY_VALUE(REAL_VALUE(15.0), #110);
#3052=PROPERTY_VALUE(INTEGER(6), #170);
#3053=PROPERTY_VALUE(INTEGER(1), #150);
#3054=PROPERTY_VALUE(INTEGER(2), #160);
#3055=PROPERTY_VALUE(INTEGER(1), #200);
#3056=PROPERTY_VALUE(INTEGER(0), #210);
#3057=PROPERTY_VALUE(#2506, #180);

#3100=LIB_F_MODEL_INSTANCE(#130, (#3101, #3108, #3109, #3102, #3103,
#3104, #3105, #3106, #3107), ());
#3101=PROPERTY_VALUE(REAL_VALUE(11.0), #90);
#3108=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#3109=PROPERTY_VALUE(REAL_VALUE(16.5), #110);
#3102=PROPERTY_VALUE(INTEGER(1), #170);
```

```
#3103=PROPERTY_VALUE(INTEGER(1), #150);
#3104=PROPERTY_VALUE(INTEGER(2), #160);
#3105=PROPERTY_VALUE(INTEGER(1), #200);
#3106=PROPERTY_VALUE(INTEGER(0), #210);
#3107=PROPERTY_VALUE(#2501, #180);
#3110=LIB_F_MODEL_INSTANCE(#130, (#3111, #3118, #3119, #3112, #3113,
#3114, #3115, #3116, #3117), ());
#3111=PROPERTY_VALUE(REAL_VALUE(11.0), #90);
#3118=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#3119=PROPERTY_VALUE(REAL_VALUE(16.5), #110);
#3112=PROPERTY_VALUE(INTEGER(2), #170);
#3113=PROPERTY_VALUE(INTEGER(1), #150);
#3114=PROPERTY_VALUE(INTEGER(2), #160);
#3115=PROPERTY_VALUE(INTEGER(1), #200);
#3116=PROPERTY_VALUE(INTEGER(0), #210);
#3117=PROPERTY_VALUE(#2502, #180);
#3120=LIB_F_MODEL_INSTANCE(#130, (#3121, #3128, #3129, #3122, #3123,
#3124, #3125, #3126, #3127), ());
#3121=PROPERTY_VALUE(REAL_VALUE(11.0), #90);
#3128=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#3129=PROPERTY_VALUE(REAL_VALUE(16.5), #110);
#3122=PROPERTY_VALUE(INTEGER(3), #170);
#3123=PROPERTY_VALUE(INTEGER(1), #150);
#3124=PROPERTY_VALUE(INTEGER(2), #160);
#3125=PROPERTY_VALUE(INTEGER(1), #200);
#3126=PROPERTY_VALUE(INTEGER(0), #210);
#3127=PROPERTY_VALUE(#2503, #180);
#3130=LIB_F_MODEL_INSTANCE(#130, (#3131, #3138, #3139, #3132, #3133,
#3134, #3135, #3136, #3137), ());
#3131=PROPERTY_VALUE(REAL_VALUE(11.0), #90);
#3138=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#3139=PROPERTY_VALUE(REAL_VALUE(16.5), #110);
#3132=PROPERTY_VALUE(INTEGER(4), #170);
#3133=PROPERTY_VALUE(INTEGER(1), #150);
#3134=PROPERTY_VALUE(INTEGER(2), #160);
#3135=PROPERTY_VALUE(INTEGER(1), #200);
#3136=PROPERTY_VALUE(INTEGER(0), #210);
#3137=PROPERTY_VALUE(#2504, #180);
#3140=LIB_F_MODEL_INSTANCE(#130, (#3141, #3148, #3149, #3142, #3143,
#3144, #3145, #3146, #3147), ());
#3141=PROPERTY_VALUE(REAL_VALUE(11.0), #90);
#3148=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#3149=PROPERTY_VALUE(REAL_VALUE(16.5), #110);
#3142=PROPERTY_VALUE(INTEGER(5), #170);
#3143=PROPERTY_VALUE(INTEGER(1), #150);
#3144=PROPERTY_VALUE(INTEGER(2), #160);
#3145=PROPERTY_VALUE(INTEGER(1), #200);
#3146=PROPERTY_VALUE(INTEGER(0), #210);
#3147=PROPERTY_VALUE(#2505, #180);
#3150=LIB_F_MODEL_INSTANCE(#130, (#3151, #3158, #3159, #3152, #3153,
#3154, #3155, #3156, #3157), ());
#3151=PROPERTY_VALUE(REAL_VALUE(11.0), #90);
```



```
#3158=PROPERTY_VALUE(REAL_VALUE(1.0), #100);
#3159=PROPERTY_VALUE(REAL_VALUE(16.5), #110);
#3152=PROPERTY_VALUE(INTEGER(6), #170);
#3153=PROPERTY_VALUE(INTEGER(1), #150);
#3154=PROPERTY_VALUE(INTEGER(2), #160);
#3155=PROPERTY_VALUE(INTEGER(1), #200);
#3156=PROPERTY_VALUE(INTEGER(0), #210);
#3157=PROPERTY_VALUE(#2506, #180);

#3200=LIB_F_MODEL_INSTANCE(#130, (#3201, #3208, #3209, #3202, #3203,
#3204, #3205, #3206, #3207), ());
#3201=PROPERTY_VALUE(REAL_VALUE(13.0), #90);
#3208=PROPERTY_VALUE(REAL_VALUE(2.0), #100);
#3209=PROPERTY_VALUE(REAL_VALUE(19.5), #110);
#3202=PROPERTY_VALUE(INTEGER(1), #170);
#3203=PROPERTY_VALUE(INTEGER(1), #150);
#3204=PROPERTY_VALUE(INTEGER(2), #160);
#3205=PROPERTY_VALUE(INTEGER(1), #200);
#3206=PROPERTY_VALUE(INTEGER(0), #210);
#3207=PROPERTY_VALUE(#2501, #180);
#3210=LIB_F_MODEL_INSTANCE(#130, (#3211, #3218, #3219, #3212, #3213,
#3214, #3215, #3216, #3217), ());
#3211=PROPERTY_VALUE(REAL_VALUE(13.0), #90);
#3218=PROPERTY_VALUE(REAL_VALUE(2.0), #100);
#3219=PROPERTY_VALUE(REAL_VALUE(19.5), #110);
#3212=PROPERTY_VALUE(INTEGER(2), #170);
#3213=PROPERTY_VALUE(INTEGER(1), #150);
#3214=PROPERTY_VALUE(INTEGER(2), #160);
#3215=PROPERTY_VALUE(INTEGER(1), #200);
#3216=PROPERTY_VALUE(INTEGER(0), #210);
#3217=PROPERTY_VALUE(#2502, #180);
#3220=LIB_F_MODEL_INSTANCE(#130, (#3221, #3228, #3229, #3222, #3223,
#3224, #3225, #3226, #3227), ());
#3221=PROPERTY_VALUE(REAL_VALUE(13.0), #90);
#3228=PROPERTY_VALUE(REAL_VALUE(2.0), #100);
#3229=PROPERTY_VALUE(REAL_VALUE(19.5), #110);
#3222=PROPERTY_VALUE(INTEGER(3), #170);
#3223=PROPERTY_VALUE(INTEGER(1), #150);
#3224=PROPERTY_VALUE(INTEGER(2), #160);
#3225=PROPERTY_VALUE(INTEGER(1), #200);
#3226=PROPERTY_VALUE(INTEGER(0), #210);
#3227=PROPERTY_VALUE(#2503, #180);
#3230=LIB_F_MODEL_INSTANCE(#130, (#3231, #3238, #3239, #3232, #3233,
#3234, #3235, #3236, #3237), ());
#3231=PROPERTY_VALUE(REAL_VALUE(13.0), #90);
#3238=PROPERTY_VALUE(REAL_VALUE(2.0), #100);
#3239=PROPERTY_VALUE(REAL_VALUE(19.5), #110);
#3232=PROPERTY_VALUE(INTEGER(4), #170);
#3233=PROPERTY_VALUE(INTEGER(1), #150);
#3234=PROPERTY_VALUE(INTEGER(2), #160);
#3235=PROPERTY_VALUE(INTEGER(1), #200);
```

```
#3236=PROPERTY_VALUE(INTEGER(0), #210);
#3237=PROPERTY_VALUE(#2504, #180);
#3240=LIB_F_MODEL_INSTANCE(#130, (#3241, #3248, #3249, #3242, #3243,
#3244, #3245, #3246, #3247), ());
#3241=PROPERTY_VALUE(REAL_VALUE(13.0), #90);
#3248=PROPERTY_VALUE(REAL_VALUE(2.0), #100);
#3249=PROPERTY_VALUE(REAL_VALUE(19.5), #110);
#3242=PROPERTY_VALUE(INTEGER(5), #170);
#3243=PROPERTY_VALUE(INTEGER(1), #150);
#3244=PROPERTY_VALUE(INTEGER(2), #160);
#3245=PROPERTY_VALUE(INTEGER(1), #200);
#3246=PROPERTY_VALUE(INTEGER(0), #210);
#3247=PROPERTY_VALUE(#2505, #180);
#3250=LIB_F_MODEL_INSTANCE(#130, (#3251, #3258, #3259, #3252, #3253,
#3254, #3255, #3256, #3257), ());
#3251=PROPERTY_VALUE(REAL_VALUE(13.0), #90);
#3258=PROPERTY_VALUE(REAL_VALUE(2.0), #100);
#3259=PROPERTY_VALUE(REAL_VALUE(19.5), #110);
#3252=PROPERTY_VALUE(INTEGER(6), #170);
#3253=PROPERTY_VALUE(INTEGER(1), #150);
#3254=PROPERTY_VALUE(INTEGER(2), #160);
#3255=PROPERTY_VALUE(INTEGER(1), #200);
#3256=PROPERTY_VALUE(INTEGER(0), #210);
#3257=PROPERTY_VALUE(#2506, #180);

#3300=LIB_F_MODEL_INSTANCE(#130, (#3301, #3308, #3309, #3302, #3303,
#3304, #3305, #3306, #3307), ());
#3301=PROPERTY_VALUE(REAL_VALUE(17.0), #90);
#3308=PROPERTY_VALUE(REAL_VALUE(3.0), #100);
#3309=PROPERTY_VALUE(REAL_VALUE(25.5), #110);
#3302=PROPERTY_VALUE(INTEGER(1), #170);
#3303=PROPERTY_VALUE(INTEGER(1), #150);
#3304=PROPERTY_VALUE(INTEGER(2), #160);
#3305=PROPERTY_VALUE(INTEGER(1), #200);
#3306=PROPERTY_VALUE(INTEGER(0), #210);
#3307=PROPERTY_VALUE(#2501, #180);
#3310=LIB_F_MODEL_INSTANCE(#130, (#3311, #3318, #3319, #3312, #3313,
#3314, #3315, #3316, #3317), ());
#3311=PROPERTY_VALUE(REAL_VALUE(17.0), #90);
#3318=PROPERTY_VALUE(REAL_VALUE(3.0), #100);
#3319=PROPERTY_VALUE(REAL_VALUE(25.5), #110);
#3312=PROPERTY_VALUE(INTEGER(2), #170);
#3313=PROPERTY_VALUE(INTEGER(1), #150);
#3314=PROPERTY_VALUE(INTEGER(2), #160);
#3315=PROPERTY_VALUE(INTEGER(1), #200);
#3316=PROPERTY_VALUE(INTEGER(0), #210);
#3317=PROPERTY_VALUE(#2502, #180);
#3320=LIB_F_MODEL_INSTANCE(#130, (#3321, #3328, #3329, #3322, #3323,
#3324, #3325, #3326, #3327), ());
#3321=PROPERTY_VALUE(REAL_VALUE(17.0), #90);
#3328=PROPERTY_VALUE(REAL_VALUE(3.0), #100);
#3329=PROPERTY_VALUE(REAL_VALUE(25.5), #110);
```



```
#3322=PROPERTY_VALUE(INTEGER(3), #170);
#3323=PROPERTY_VALUE(INTEGER(1), #150);
#3324=PROPERTY_VALUE(INTEGER(2), #160);
#3325=PROPERTY_VALUE(INTEGER(1), #200);
#3326=PROPERTY_VALUE(INTEGER(0), #210);
#3327=PROPERTY_VALUE(#2503, #180);
#3330=LIB_F_MODEL_INSTANCE(#130, (#3331, #3338, #3339, #3332, #3333,
#3334, #3335, #3336, #3337), ());
#3331=PROPERTY_VALUE(REAL_VALUE(17.0), #90);
#3338=PROPERTY_VALUE(REAL_VALUE(3.0), #100);
#3339=PROPERTY_VALUE(REAL_VALUE(25.5), #110);
#3332=PROPERTY_VALUE(INTEGER(4), #170);
#3333=PROPERTY_VALUE(INTEGER(1), #150);
#3334=PROPERTY_VALUE(INTEGER(2), #160);
#3335=PROPERTY_VALUE(INTEGER(1), #200);
#3336=PROPERTY_VALUE(INTEGER(0), #210);
#3337=PROPERTY_VALUE(#2504, #180);
#3340=LIB_F_MODEL_INSTANCE(#130, (#3341, #3348, #3349, #3342, #3343,
#3344, #3345, #3346, #3347), ());
#3341=PROPERTY_VALUE(REAL_VALUE(17.0), #90);
#3348=PROPERTY_VALUE(REAL_VALUE(3.0), #100);
#3349=PROPERTY_VALUE(REAL_VALUE(25.5), #110);
#3342=PROPERTY_VALUE(INTEGER(5), #170);
#3343=PROPERTY_VALUE(INTEGER(1), #150);
#3344=PROPERTY_VALUE(INTEGER(2), #160);
#3345=PROPERTY_VALUE(INTEGER(1), #200);
#3346=PROPERTY_VALUE(INTEGER(0), #210);
#3347=PROPERTY_VALUE(#2505, #180);
#3350=LIB_F_MODEL_INSTANCE(#130, (#3351, #3358, #3359, #3352, #3353,
#3354, #3355, #3356, #3357), ());
#3351=PROPERTY_VALUE(REAL_VALUE(17.0), #90);
#3358=PROPERTY_VALUE(REAL_VALUE(3.0), #100);
#3359=PROPERTY_VALUE(REAL_VALUE(25.5), #110);
#3352=PROPERTY_VALUE(INTEGER(6), #170);
#3353=PROPERTY_VALUE(INTEGER(1), #150);
#3354=PROPERTY_VALUE(INTEGER(2), #160);
#3355=PROPERTY_VALUE(INTEGER(1), #200);
#3356=PROPERTY_VALUE(INTEGER(0), #210);
#3357=PROPERTY_VALUE(#2506, #180);

#3400=LIB_F_MODEL_INSTANCE(#130, (#3401, #3408, #3409, #3402, #3403,
#3404, #3405, #3406, #3407), ());
#3401=PROPERTY_VALUE(REAL_VALUE(19.0), #90);
#3408=PROPERTY_VALUE(REAL_VALUE(4.0), #100);
#3409=PROPERTY_VALUE(REAL_VALUE(28.5), #110);
#3402=PROPERTY_VALUE(INTEGER(1), #170);
#3403=PROPERTY_VALUE(INTEGER(1), #150);
#3404=PROPERTY_VALUE(INTEGER(2), #160);
#3405=PROPERTY_VALUE(INTEGER(1), #200);
#3406=PROPERTY_VALUE(INTEGER(0), #210);
#3407=PROPERTY_VALUE(#2501, #180);
```

```
#3410=LIB_F_MODEL_INSTANCE(#130, (#3411, #3418, #3419, #3412, #3413,
#3414, #3415, #3416, #3417), ());
#3411=PROPERTY_VALUE(REAL_VALUE(19.0), #90);
#3418=PROPERTY_VALUE(REAL_VALUE(4.0), #100);
#3419=PROPERTY_VALUE(REAL_VALUE(28.5), #110);
#3412=PROPERTY_VALUE(INTEGER(2), #170);
#3413=PROPERTY_VALUE(INTEGER(1), #150);
#3414=PROPERTY_VALUE(INTEGER(2), #160);
#3415=PROPERTY_VALUE(INTEGER(1), #200);
#3416=PROPERTY_VALUE(INTEGER(0), #210);
#3417=PROPERTY_VALUE(#2502, #180);
#3420=LIB_F_MODEL_INSTANCE(#130, (#3421, #3428, #3429, #3422, #3423,
#3424, #3425, #3426, #3427), ());
#3421=PROPERTY_VALUE(REAL_VALUE(19.0), #90);
#3428=PROPERTY_VALUE(REAL_VALUE(4.0), #100);
#3429=PROPERTY_VALUE(REAL_VALUE(28.5), #110);
#3422=PROPERTY_VALUE(INTEGER(3), #170);
#3423=PROPERTY_VALUE(INTEGER(1), #150);
#3424=PROPERTY_VALUE(INTEGER(2), #160);
#3425=PROPERTY_VALUE(INTEGER(1), #200);
#3426=PROPERTY_VALUE(INTEGER(0), #210);
#3427=PROPERTY_VALUE(#2503, #180);
#3430=LIB_F_MODEL_INSTANCE(#130, (#3431, #3438, #3439, #3432, #3433,
#3434, #3435, #3436, #3437), ());
#3431=PROPERTY_VALUE(REAL_VALUE(19.0), #90);
#3438=PROPERTY_VALUE(REAL_VALUE(4.0), #100);
#3439=PROPERTY_VALUE(REAL_VALUE(28.5), #110);
#3432=PROPERTY_VALUE(INTEGER(4), #170);
#3433=PROPERTY_VALUE(INTEGER(1), #150);
#3434=PROPERTY_VALUE(INTEGER(2), #160);
#3435=PROPERTY_VALUE(INTEGER(1), #200);
#3436=PROPERTY_VALUE(INTEGER(0), #210);
#3437=PROPERTY_VALUE(#2504, #180);
#3440=LIB_F_MODEL_INSTANCE(#130, (#3441, #3448, #3449, #3442, #3443,
#3444, #3445, #3446, #3447), ());
#3441=PROPERTY_VALUE(REAL_VALUE(19.0), #90);
#3448=PROPERTY_VALUE(REAL_VALUE(4.0), #100);
#3449=PROPERTY_VALUE(REAL_VALUE(28.5), #110);
#3442=PROPERTY_VALUE(INTEGER(5), #170);
#3443=PROPERTY_VALUE(INTEGER(1), #150);
#3444=PROPERTY_VALUE(INTEGER(2), #160);
#3445=PROPERTY_VALUE(INTEGER(1), #200);
#3446=PROPERTY_VALUE(INTEGER(0), #210);
#3447=PROPERTY_VALUE(#2505, #180);
#3450=LIB_F_MODEL_INSTANCE(#130, (#3451, #3458, #3459, #3452, #3453,
#3454, #3455, #3456, #3457), ());
#3451=PROPERTY_VALUE(REAL_VALUE(19.0), #90);
#3458=PROPERTY_VALUE(REAL_VALUE(4.0), #100);
#3459=PROPERTY_VALUE(REAL_VALUE(28.5), #110);
#3452=PROPERTY_VALUE(INTEGER(6), #170);
#3453=PROPERTY_VALUE(INTEGER(1), #150);
#3454=PROPERTY_VALUE(INTEGER(2), #160);
```



```
#3455=PROPERTY_VALUE(INTEGER(1), #200);
#3456=PROPERTY_VALUE(INTEGER(0), #210);
#3457=PROPERTY_VALUE(#2506, #180);

ENDSEC;
END-ISO-10303-21;
/*
```

P.3 Capturing a parts family in ISO 13584 using the implicit modeling approach

P.3.1 Description of the PAW parts family

This description is twofold. First, concepts of PAW and of its properties are defined. Second, the allowed set of instances is precisely (but implicitly) defined.

P.3.1.1 Dictionary description: the BSU mechanism

The description of a data dictionary according to the **ISO13584_IEC61360_dictionary_schema** requires the specification of what are the identifiers of the different concepts (called basic semantic units: BSU) involved in the parts family definition. These identifiers define unambiguously and universally each concept within an ISO 13584-conformant data dictionary.

The following example (Figure P.22) outlines the resources used for the specification of these identifiers:

```
/*BSU for supplier */
/* The code of the supplier must be defined according to ISO 13584-26: Supplier identification
Here, the code doesn't follow the ISO 13584-26 requirements, because the supplier code is not
known at the moment */
#20=SUPPLIER_BSU('INA', *);


/* BSU for component_class */
/* The class BSUs defines the identification of the various classes, and who is the supplier that is
responsible of the class definitions */
#50=CLASS_BSU('BEARING', '001', #20);
#60=CLASS_BSU('PAW', '001', #20);

/* BSU for properties */
/* The properties BSUs define the properties identification and the class where these properties
are visible. */
#90=PROPERTY_BSU('d_in', '001', #50);
#100=PROPERTY_BSU('d_out', '001', #50);
#110=PROPERTY_BSU('e', '001', #60);

/* BSU for table */
/* The table BSU defines the identification of the table, and the class where this table is visible */
#120=TABLE_BSU('T1', '001', #50);
```

Figure P.22 — Identifiers of the concepts involved in the PAW family

P.3.1.2 Dictionary description: the dictionary element definition

A BSU only identifies a concept. A **dictionary_element** provides a computer-sensible and human-readable definition of the concept. This relationship between these two levels is presented in Figure P.23.

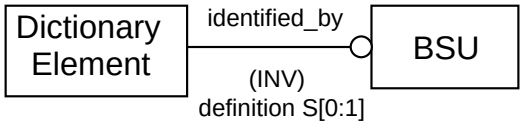


Figure P.23 — The BSU / Dictionary element relationship

Figure P.24 outlines the main structure of the **dictionary_element**s corresponding to the previous basic semantic units identifiers.

```
/* Supplier definition */
#21=SUPPLIER_ELEMENT(
          #20, /* reference to its BSU */
          $, '001',
          #22, #23); /* organisation and address */
/* Property definition */
#91=NON_DEPENDENT_P_DET(
          #90, /* reference to its BSU */
          $, '001',
          #92, /* item_names (human-readable name of the concept,
                    with possible translations) */
          TEXT('inner diameter'), $, $, $, $, (), $,
          'TO3', /* the data element type classification,
                    according to ISO 31*/
          #93, /* the specific data type of the property
               (not represented: measure in mm) */
          $);
/* Class definition*/
#71=COMPONENT_CLASS(
          #50, /* reference to its BSU */
          $, '001',
          #72, /* item_names */
          TEXT('Class associated...'), /* definition */
          $, $, $, $,
          (#90, #100),  /* the list of the properties that may be
          used to describe an instance of this class
          (applicability of the properties) */
          (), $, (),(), $);

/* Table definition*/
#121=RDB_TABLE_ELEMENT(
          #120, /* reference to its BSU */
          $, '001',
          #122, /* item_names */
          TEXT('This table...'), $, $, *,
          (#96, #116), /* the columns meaning: variable_semantics
               entities that refer to the previous property_BSU
               (not represented) */
```



```
          (#96), /* the key of the table */
          );
/* Class table relationship */
#122=CLASS_TABLE_RELATIONSHIP(#81, (#120));
```

Figure P.24 — Dictionary_element of the concepts involved in the PAW family

The **dictionary_element** of the table is similar to the other elements of the data dictionary. Here, the table is defined as a relational database table: that means that the type of the elements belonging to each columns are simple data types (number, Boolean or string data type).

Moreover, theclass/table relationship permits to define the applicability level of the table in the data dictionary. It plays the same role as the **described_by** attribute for the applicability of properties in the class **dictionary_element**.

P.3.1.3 Library specification: description of the class extension

The PAW family being a catalogue defined family, only some instances are really allowed. This set of instances is implicitly defined:

c) the choice (by the library data supplier) of the identification characteristics of the family (in this example: the **d_in** property),

d) the definition of the allowed set of values for these (here: this) identification characteristics by means of a **domain_restriction** (here: a **table_defined_domain**), and

e) the definition of the derivation functions that specify the values of the other parts characteristics as a set of functions whose domains consist of derived characteristics (here: **d_out** is specified by means of an algebraic expression and **e** by means of a table whose key corresponds to the identification characteristics **d_in**).

P.3.1.3.1 Overall architecture

The relationship between a dictionary element and an associated library specification **content_item** is outlined in Figure P.25:

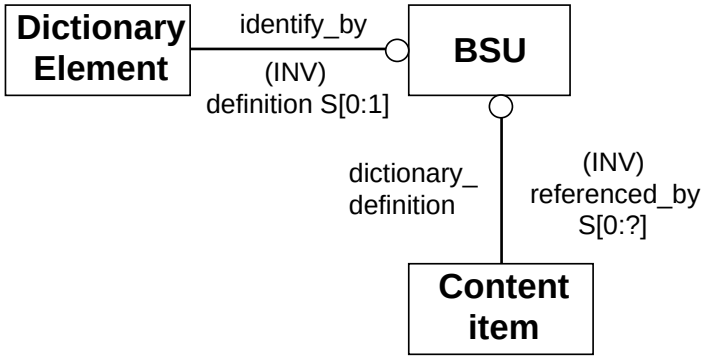


Figure P.25 — The Dictionary Element / Library Content relationship

Each content item shall be consistent with the corresponding dictionary element definition (respect of the type / value relationship).

P.3.1.3.2 Representation of the SELF variables

When defining expressions, it is required to model the variables that are involved as operands.

In ISO 13584, a variable is modeled by a threefold structure:

— a syntactical level (**generic_variable**): definition of the name (entity name) and of the type of the variable.

EXAMPLE 1 In the EXPRESS language (ISO 10303-11), this level is captured by the following syntactical notation:

x: INTEGER;

— a semantic level (**variable_semantics**): definition of the meaning of the variable, and of the mechanism that associates it with a value:

EXAMPLE 2 In the EXPRESS language (ISO 10303-11), an example of semantic level is provided by the SELF notation:

WHERE SELF.x = 1;

An association between syntax and semantics is done using the classical entity/relationship mechanism as shown in Figure P.26.

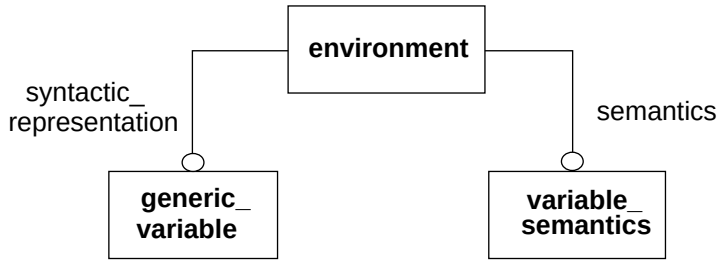


Figure P.26 — Syntax / Semantics variable association

Figure P.27 shows this association for the inner diameter (i.e., **d_in**) part characteristics..

```
/* Properties syntax definition */
#97=REAL_NUMERIC_VARIABLE(); /* a real variable named #97 */
/* Definition of the properties semantics */
#96=SELF_PROPERTY_VALUE_SEMANTICS(#90, $); /* the meaning of the
       variable #97 is the current value associated with the #90 BSU
       property for the SELF class instance */
/* Syntax / semantics association */
#98=ENVIRONMENT(#97, #96); /* the real variable identified
       by #97 stands for the SELF value of the inner diameter (d_in)*/
```

Figure P.27 — Data model of the variable that stands of the inner diameter of a PAW instance

P.3.1.3.3 Optional and displayable nature of properties

The library specification includes the definition of the optional or mandatory nature of the data dictionary properties, and the possible displayability (in a library management system) of these same properties.

```
/* Library definition of the properties */
#900=OPT_OR_MAND_PROPERTY_BSU(
           #90,
           .F., /* is not optional */
           .T.); /* is displayable */
```

Figure P.28 — Displayable and optional properties

P.3.1.3.4 Table content

The library specification of a table (**RDB_table_content**) enables to define the real content of this table that shall be consistent with the data types defined in the data dictionary (**RDB_table_element**). Figure P.29 describes the content of the table defined in Figure P.6. It contains two columns.

```
/* Content of the table */
#910=RDB_TABLE_CONTENT(
            #120, /* reference to its BSU */
            *,
            (#911, #912), /* the list of the referenced columns */
            '001', '2000-09-06');
#911=REAL_COLUMN(
            (10.0, 11.0, 13.0, 17.0, 19.0),/* the values of the column */
            'NR2..3.3' /* the format of the values of the column */
            );
```

Figure P.29 — Content of a table

P.3.1.3.5 Implicit description of the allowed instances: Domain

The identification characteristics of a part (its inner diameter **d_in** in our example) shall belong to some well defined value domains. Such a constraint is specified through a **domain_restriction** whose structure is outlined in Figure P.30.

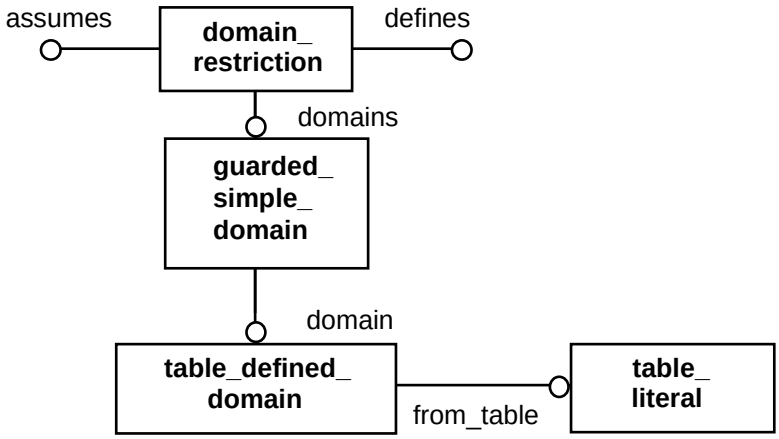


Figure P.30 — Domain restriction description

This construct specifies the allowed values for the properties belonging to the **defines** attribute list, assuming that the values of the properties referenced in the **assumes** list are known. If the considered domain is a table, the particular **domain_restriction** used is a **table_defined_domain** (a specialisation of the **simple_domain**) that refers to the considered table.

```
/* Property domain */
#901=DOMAIN_RESTRICTION(
          (#96), /* the property domain that is defined */
          (), /* the required properties for evaluating the
```

```
                property domain ('assumes' list)*/
            (#902),   /* the guarded domain */
            $);
#902=GUARDED_SIMPLE_DOMAIN(
            #903, /* the guard */
            #904 /* the domain */
            );
#903=OTHERS(); /* this guard means 'in every cases' */
#904=TABLE_DEFINED_DOMAIN(#905); /* the table domain */
#905=TABLE_LITERAL(#120); /* the referenced table */
```

Figure P.31 — Specification of a domain as a table

P.3.1.3.6 Description of the derivation functions

d_in being the only identification characteristics of a PAW instance, there exists a functional dependency between **d_in** and **d_out** and between **d_in** and **e**. The general structure of a derivation function is outlined Figure P.32.

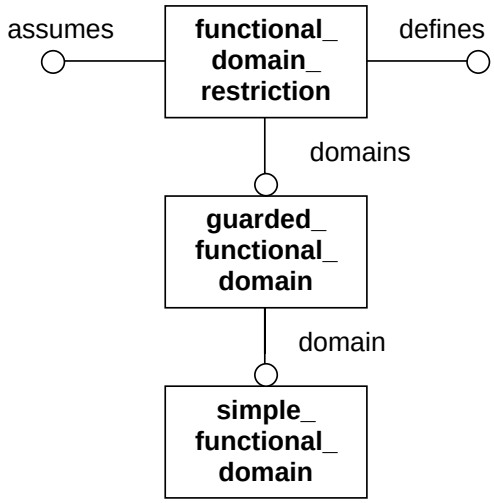


Figure P.32 — Derivation

It is built on the same template as the **domain_restriction** presented function. A derivation function is represented as a **functional_domain_restriction** that references some **assumes** properties (the parameters of the derivation relation) and **defines** properties (the derived properties). The **domains** attribute defines the associated guarded domains. These domains are specialised as some **simple_functional_domain**s (i.e. mathematical functions).

P.3.1.3.6.1 Algebraic derivation function

For the **d_out** property, the derivation function is defined by a specialisation of the **simple_functional_domain**s that is a **library_expression_defined_value** as presented in Figure P.33.

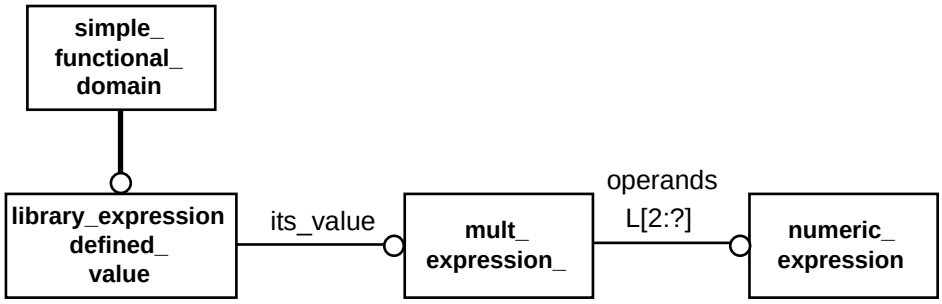


Figure P.33 — Derivation by algebraic expressions

In the physical file, the derivation function is described as a multiplication between a real constant and a real variable (associated with the inner diameter **d_in**). The result of this expression specifies the value of the **d_out** SELF property.

```
/* Definition of the derivation function 'd_out = d_in * 1.5' */
#1001=FUNCTIONAL_DOMAIN_RESTRICTION(
          (#106), /* the property of which the value is defined */
          (#96), /* the self properties that define the parameters
                    of the derivation function */
          (#1002), /* domains */
          $);
#1002=GUARDED_FUNCTIONAL_DOMAIN(
          #1003, /* the guard */
          #1004, /* the derivation function */
          );
#1003=OTHERS();
#1004=LIBRARY_EXPRESSION_DEFINED_VALUE(#1005);
#1005=MULT_EXPRESSION((#1006, #97)); /* the operands */
#1006=REAL_LITERAL(1.5); /* a literal value */
```

Figure P.34 — Specification of a property value by an algebraic expression

P.3.1.3.6.2 Table-defined derivation function

The derivation table is defined by a specialisation of the **simple_functional_domain**s that is a **table_defined_value** as presented in the Figure P.35.

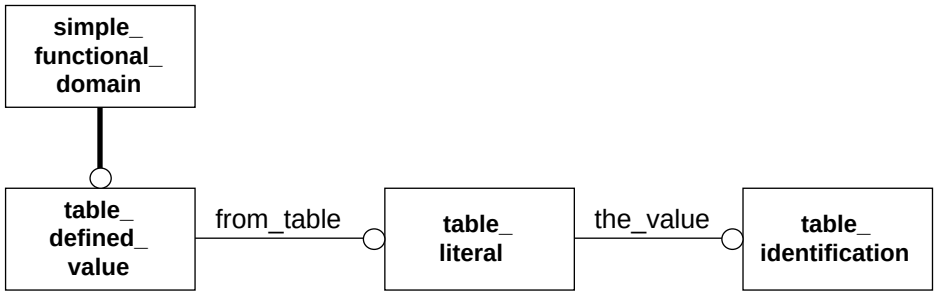


Figure P.35 — Derivation table

In the physical file, the derivation table defines the value of the thickness **e** of the part from the inner diameter **d_in**. The derivation table is referred to by the EXPRESS entity **table_defined_value**.

```
#1101=FUNCTIONAL_DOMAIN_RESTRICTION(
            (#116), /* the property domain that is defined
                       (belonging to the table) */
            (#96), /* the parameter of the derivation function
                       (belonging to the table)*/
            (#1102), $);
#1102=GUARDED_FUNCTIONAL_DOMAIN(#1103, #1104);
#1103=OTHERS();
#1104=TABLE_DEFINED_VALUE(#1105);
#1105=TABLE_LITERAL(#120); /* the referenced table */
```

Figure P.36 — Specification of a property value by a table

P.3.1.3.7 Class extension

The final description of the PAW class extension is a specialisation of the **content_item** presented in Figure P.37. It associates to the class all the derivation functions and constraints defined previously.

```
/* Class extension */
#8000 = ITEM_CLASS_EXTENSION(
          #60, /* reference to its BSU */
          (), (), (), '001', '001', (), (), *, *, *,
          (#901), /* the constrained domain */
          (#1001, #1101), /* the derivations */
          (), (),
          (#900), /* the identification characteristics */
          (#1000, #1100), /* the derived characteristics */
          (), .F., $, $, $, $, (), $, (), ()
          );
```

Figure P.37 — Description of the PAW implicit class extension

P.3.2 Description of geometric representations for the PAW parts family

We assume now that some library data supplier wants to provide a geometric representation for all the instances of the PAW family. This requires the description of a functional model class.

P.3.2.1 Overview

A functional model class is intended to represent different perspectives of the different parts described in the general model class. A functional model class has to be described like a general model class, i.e., through a class definition and through a dictionary extension (library specification).

The relationship between the three kinds of classes defined in ISO 13584 is outlined in Figure P.38.

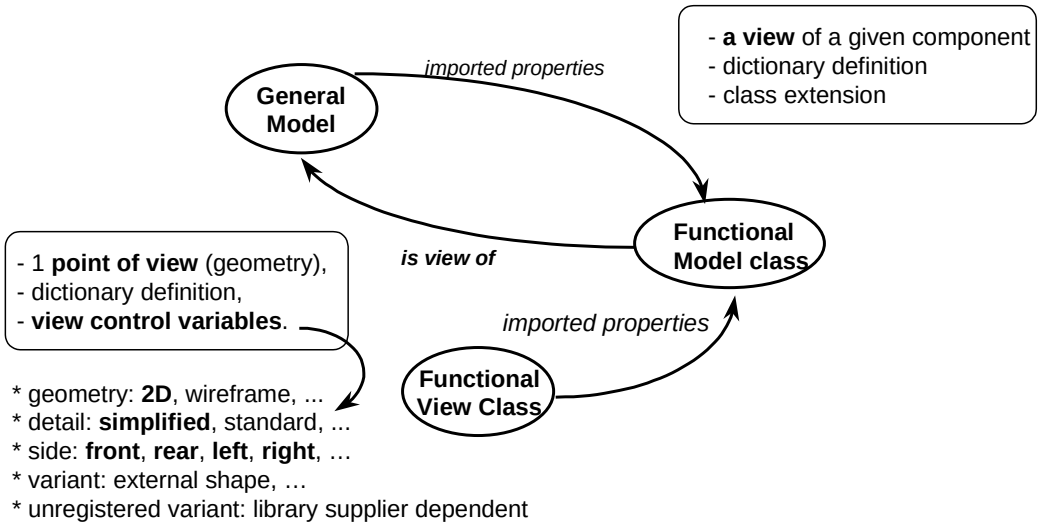


Figure P.38 — Association mechanism between a general model and a functional model

A **functional_model_class** describes a particular view (is-view-of relationship) of a given parts family (described as a general model class), according to the point of view specified by a **functional_view_class**.

In the example, a functional model class representing some kind of geometry specified by a **functional_view_class** for the PAW parts family will be defined.

P.3.2.2 The functional_view_class to be described

A **functional_view_class** is identified by a **class_BSU** (i.e., essentially a code) and further specified through some properties called the view control variables.

NOTE 1 **Functional_view_class**es are intended to be specified as standard data in the view exchange protocol series of parts of ISO 13584 and to be stored in the user libraries. Such standard **functional_view_class**es are intended to be exchanged using LIBRARY INTEGRATED INFORMATION MODEL 24-3 defined in this part of ISO 13584. Moreover, a **functional_view_class** is only described by a **dictionary_element**.

NOTE 2 The data type of any view control variable is **non_quantitative_int_type** that is a subtype of integer. The values of view control variables are therefore integer values. But, each value is associated with a human-readable (and translated) label defined in the **representation_P_DET** that constitutes the **dictionary_element** of the view control variable.

In the example, we assume that the receiving system supports the geometric view defined in the current version of ISO 13584-101.

Table P.1 — View control variables of the geometry functional view class

View control variable of the ISO 13584-101 geometry view			
code	**version**	**value**	**corresponding label** (in English)
‘geometry_level’	‘001’	1 2 3	‘2D’ ‘wireframe’ ‘solid’
‘detail_level’	‘001’	1 2 3	‘simplified’ ‘standard’ ‘extended’
‘side’	‘001’	0 1 2 3 4 5 6	'null' ‘front’ ‘rear’ ‘right’ ‘left’ ‘top’ ‘bottom’
'variant'	'001'	0 1 2 3 ... n	'null' 'external_shape' 'section' *reserved for future registration*
'unreg_variant'	'001'	0 1 ... n	'null' *library data supplier dependent*

This geometric view is:

— defined by ISO in ISO 13584-101, supplier code: ‘0112/1///13584_101_1’;

— identified by the class_BSU code: ‘basic_geometry’ and by the class_BSU version = ‘001’;

— characterised by five view control variables defined in table P.1:

P.3.2.3 Dictionary_element of a functional model class

A **fm_class_view_of** is a functional model class that refers to a well defined general model class (here the class that models the PAW family) and that provides a particular kind of representation (specified by a **functional_view_class**) for this general model class.

A functional model class is not required to provide representation for all the values of the functional view class view control variables. The range of supported values is specified by **view_control_variable_range** as shown in Figure P.39.

```
/* v_c_v range */
#155=VIEW_CONTROL_VARIABLE_RANGE(#150, 1, 1);
     /* reference to geometry_level property_bsu (#150)
     /* values range: [1:1] */
#165=VIEW_CONTROL_VARIABLE_RANGE(#160, 2, 2);
```



```
      /* reference to detail_level property_bsu (#160) */
      /* values range: [2:2] */
#175=VIEW_CONTROL_VARIABLE_RANGE(#170, 1, 6);
      /* reference to side property_bsu (#170) */
      /* values range: [1:6] */
#205 = VIEW_CONTROL_VARIABLE_RANGE (#200, 1, 1);
      /* reference to variant property_bsu (#200) */
      /* values range: [1:1] */
#215 = VIEW_CONTROL_VARIABLE_RANGE (#210, 0, 0);
      /* reference to unreg_variant property_bsu (#210) */
      /* values range: [0:0] */
```

Figure P.39 — View control variables range definition

In our example, the functional model class only provides 2D views (range 1..1, for #150 that is ‘geometry_level’), with standard representation (range 2..2, for #160 that is ‘detail-level’) for all the sides from ‘front’ to ‘bottom’ (range 1..6, for # 170 that is ‘side’) and with a standardised shape definition (range bounds to 0 for #215 that is 'unreg_variant', and range 1..1 for #205 that is 'variant').

Moreover, to be able to create the geometry, the **fm_class_view_of** needs to import some properties from the PAW family (in fact all the three defined properties). The **dictionary_element** of the **fm_class_view_of** is presented in the Figure P.40.

```
  /* Functional model class definition*/
#71=FM_CLASS_VIEW_OF(
            #130, /* reference to its BSU */
            $, '001',
            #72, /* item_names */
            TEXT('Functional model... of PAW'),
            $, $, $, $,
            (#180, #190), /* BSU of the 'prg' and of the required_side
                     properties: not represented. See P.3.4*/
            (), *, *, *, *, *,
            #140, /* the created view (reference to the BSU of the
                     functional_view_class */
            (#155, #165, #175, #205, #215), /* the v_c_v ranges */
            (#150, #160, #170, #200, #210), /* the v_c_v imported from
                     the functional_view_class */
            (), (), (),
            (), /* the possible referenced functional model classes */
            (), (), (), (), /* importation from referenced functional
                     model classes*/
            #60, /* the is-view-of semantic relationship: the reference
                     to the BSU of the described class */
            (#90, #100, #110), /* imported properties from the described
                     class of the general model */
            (),(),());
```

Figure P.40 — Specification of the view created by a functional model class

P.3.2.4 Library specification of the functional model class

P.3.2.4.1 Overview of the view creation mechanism

A **functional_model_class_extension** is the **content_item** that constitutes the library specification of a **functional_model_class** and **fm_class_view_of** subtype). It permits the definition of what are the properties that need to be valued in the context of an instance of such a class, and what are the methods associated with such a class (the main role of such methods is to call the different programs or representations associated, as external files, to such a class).

Figure P.41 outlines the different steps performed by the library management system when the user requires a particular view for some PAW instance already selected from the PAW family.

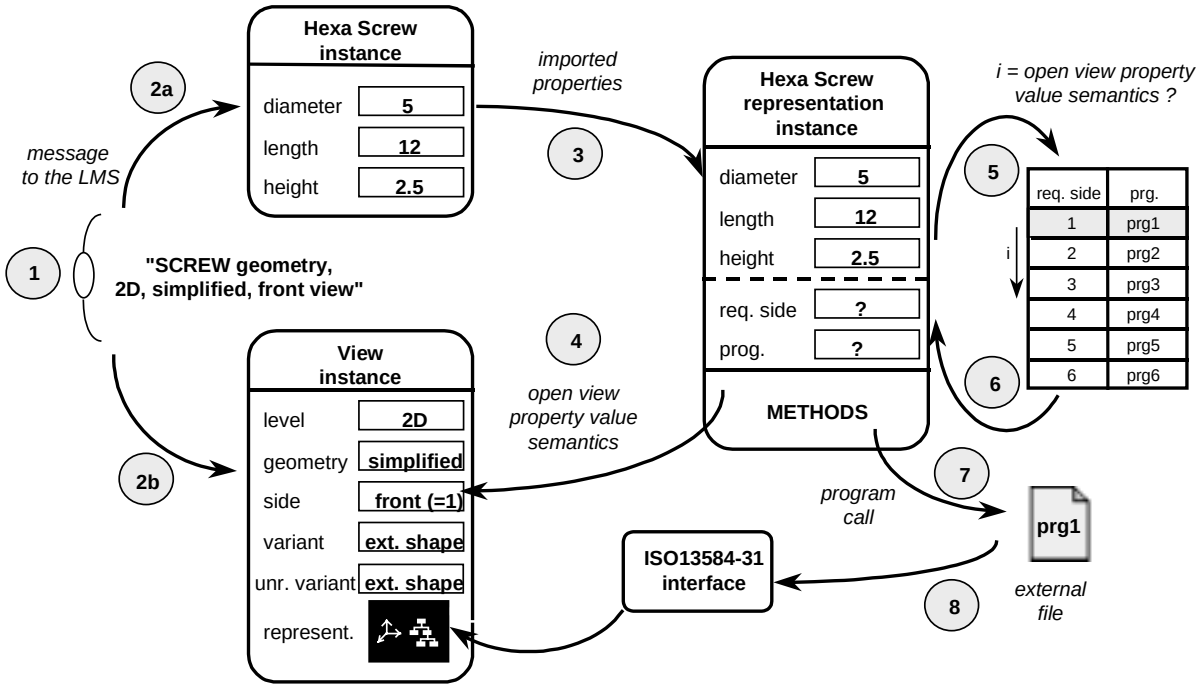


Figure P.41 — The view creation mechanism

These are the steps followed:

a) The end-user asks for a particular geometry (2D, simplified, front view) of a particular part of the PAW family (1). It is assumed at this level that this instance of PAW is already selected.

b) The following two processes are performed in parallel:

   1) a message is sent to the LMS (2a) in order to determine what is the **functional_model_class** that must be instanciated, according to the instance of the general model assumed to exist in the LMS and the view required by the user, and

   2) an instance of the **functional_view_class** is created (2b), according to the view and view control variable values defined by the end-user.

c) The **functional_model_class** instance (that is a **fm_class_view_of** subtype) is initialised by the LMS that assigns values for the properties declared as required properties from the PAW part (3). Then, the relevant instance method is triggered. Note that during the run of the method, the **functional_model_class** instance is associated with the instance of the **functional_view_class**. Now, it is assumed that the **prg** property needs to be evaluated, through a derivation table, from the **side** property. Therefore, the **functional_model_class** instance needs to know the value of the

side view_control_variable defined in the instance of the **functional_view_class** in order to compute the right **prg** program (through the **methods**).

d) Through an **open_view_property_value_semantics** the method access to the geometry view to get the **side** value of this **functional_view_class** instance (4).

e) The **prg** attribute is computed through a derivation table and a selection expression (5)(6): the table record for which the **side view_control_variable** value equals to the **required_side fm_class_view_of** property value (the table key) given in the table is selected, and permits to call the relevant program stored as an external file (7).

f) The program stored as an external file creates the **functional_view_class** instance **representation** through the ISO 13584-31 interface (8).

P.3.2.4.2 Description of methods

A **method** is defined firstly by its specifications (**method_specif**), that specifies:

— the view(s) created by the method,

— the functional model class instance properties used in the method,

and, secondly, by its body (**method_body**), that specifies the list of the statements that shall be performed.

A **method** also defines the **external_file_protocol** referenced for filling the **items** attribute of the current open view that will be used (for instance, the ISO 13584-31 Geometric Programming Interface, a CAD system specific interface, an Application Protocol from ISO 10303, ...).

The example of Figure P.42 presents a particular method that calls a program according to the view requested by the end-user.

The **method_specif** contains the following attributes: the reference to a particular functional view class, the supported **view_control_variables_range**s and the model needed properties.

The **method_body** declares the variables accessed from the method (context) and describes the different (guarded) statements that have to be performed: in our example, a program call. Note that such a statement triggers an external program whose name is stored in a table whose key is the **side** property as required by the user.

```
/* Definition of the methods */
#3000=METHOD(
          #3001, /* the specification of the method */
          #3002, /* the body of the method */
          #7); /* the used program protocol */
#3001=METHOD_SPECIF(
          #140, /* reference to the BSU of the functional_view_class */
          (#155, #165, #175; #205, #215), /* the supported v_c_v
                        ranges*/
          (#90, #100, #110), /* the properties from the functional
                        model whose values are needed */
          ());
#3002=METHOD_BODY(
          (#97, #107, #117, #177, #187, #197, #207), /* the variables
                    of the method context */
          (#3019)); /* the statement of the method */
```



```
#3019=METHOD_STATEMENT((#3020));
#3020=GUARDED_STATEMENT((#3021, #3023));
#3021=BOOLEAN_LITERAL(.T.);
#3023=CALL_PROGRAM_STATEMENT(
          #187, /* the method variable (prg) */
          #2401, /* the referenced program to be executed (specified by
               a functional domain restriction (see fig. R.12) */
          (#97, #107, #117), /*the input parameters */
          (), ());
```

Figure P.42 — Description of a method

P.3.2.5 Functional_model_class_extension

The **functional_model_class_extension** is the **content_item** associated with the **fm_class_view_of dictionary_element**. It gathers all the entities involved in the class description: the external files used, the **external_file_protocol**s required, the part characteristics needed and the **method**s it supports. The relationship between the **dictionary_element** and the library specification is done as presented in Figure P.43 (through the associated BSU).

```
#1300=FUNCTIONAL_MODEL_CLASS_EXTENSION(
          #130, /* reference to the BSU */
          *, *, *, (), (), (), (), (),
          (#2303, #2304, #2305, #2306, #2307, #2308), /* the derivation
                    table that computes the program from the side */
          (#7), /* the used program protocol: ISO 13584-31*/
          (#12), /* the used view exchange protocol */
          '001', '001', (), (), $,
          (#900, #1000, #1100), /* the required item characteristics */
          (), (),
          (#1700, #1800, #1900), /* method variables */
          (),
          (#3000), /* the associated method */
          $, $, (), $);
```

Figure P.43 — Library specification of a functional model class

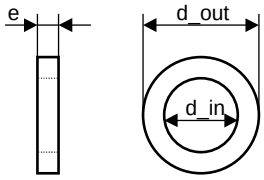
P.3.3 Resulting Physical files

In this clause, the complete examples of physical files are provided. The first physical file, conformant with LIBRARY INTEGRATED INFORMATION MODEL 24-1 defines the PAW family. The second physical file, conformant with LIBRARY INTEGRATED INFORMATION MODEL 24-2 and VIEW EXCHANGE PROTOCOL DIS 101 (for the definition of the geometry view) defines the 2D geometry of the PAW family by means of programs conformant with ISO 13584-31.

P.3.3.1 Example of a general model

This subclause contains the complete physical file presented in clause P.2.

```
/*
The PAW family
```

d_in	e
10	1
11	1
13	2
17	3
19	4

d_out = d_in * 1.5

```
  */

ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('PLIB GENERAL MODEL EXAMPLE'), '1');
FILE_NAME('P24_gm_implicit.p21', '2000-06-05T02:38:14',
          (''),
          ('LISI/ENSMA'),
          'ECCO RUNTIME SYSTEM BUILT-IN PREPROCESSOR V2.3.3',
          'ECCO RUNTIME SYSTEM V2.3.3',
          '');
FILE_SCHEMA(('ISO13584_G_M_IIM_LIBRARY_IMPLICIT_SCHEMA'));
ENDSEC;

DATA;
```

/* Global library description */

```
#2=LIBRARY_IN_STANDARD_FORMAT($, $, $, $, (), #20, #11, (), (), (#20),
(#50, #60), (), #3, $, $, ());
#3=ITEM_NAMES(LABEL('General model example'), (), LABEL(''), $, $);
#10=GLOBAL_LANGUAGE_ASSIGNMENT ('en');
#11=LIBRARY_IIM_IDENTIFICATION($, 'IS', 'ISO_13584_24_1', 2001, '3', $,
());
```

/* DICTIONARY DESCRIPTION */

/* BSU for supplier */

```
#20=SUPPLIER_BSU('INA', *);
```

/* BSU for component_class */

```
#50=CLASS_BSU('BEARING', '001', #20);
#60=CLASS_BSU('PAW', '001', #20);
```

/* BSU for properties */

```
#90=PROPERTY_BSU('d_in', '001', #50);
#100=PROPERTY_BSU('d_out', '001', #50);
#110=PROPERTY_BSU('e', '001', #60);
```

/* BSU for table */

```
#120=TABLE_BSU('T1', '001', #60);
```

/* Dictionary properties description */

/* supplier description */

```
#21=SUPPLIER_ELEMENT(#20, $, '001', #22, #23);
#22=ORGANIZATION($, 'INA', '');
#23=ADDRESS($, $, $, $, $, $, $, 'GERMANY', $, $, $, $);

/* d_in */
#91=NON_DEPENDENT_P_DET(#90, $, '001', #92, TEXT('inner diameter'),
$, $, $, $, (), $, 'TO3', #93, $);
#92=ITEM_NAMES(LABEL('inner diameter'), (), LABEL(''), $, $);
#93=REAL_MEASURE_TYPE('NR2..3.3', #94);
#94=DIC_UNIT(#95, $);
#95=SI_UNIT(*, .MILLI., .METRE.);

/* d_out */
#101=NON_DEPENDENT_P_DET(#100, $, '001', #102, TEXT('outer
diameter'), $, $, $, $, (), $, 'TO3', #93, $);
#102=ITEM_NAMES(LABEL('outer diameter'), (), LABEL(''), $, $);
#103=REAL_MEASURE_TYPE('NR2..3.3', #104);
#104=DIC_UNIT(#105, $);
#105=SI_UNIT(*, .MILLI., .METRE.);

/* e */
#111=NON_DEPENDENT_P_DET(#110, $, '001', #112, TEXT('thickness'), $,
$, $, $, (), $, 'TO3', #93, $);
#112=ITEM_NAMES(LABEL('thickness'), (), LABEL(''), $, $);
#113=REAL_MEASURE_TYPE('NR2..3.3', #114);
#114=DIC_UNIT(#115, $);
#115=SI_UNIT(*, .MILLI., .METRE.);

/* Dictionary class description */
/* Part class */
#71=COMPONENT_CLASS(#50, $, '001', #72, TEXT('Class associated to
the generic bearing family'), $, $, $, $, (#90, #100), (), $, (),(),
$);
#72=ITEM_NAMES(LABEL('Generic bearing family'), (), LABEL('Bearing
family'), $, $);

/* PAW class */
#81=COMPONENT_CLASS(#60, $, '001', #82, TEXT('Class associated to
the PAW part family'), $, $, $, #50, (#110), (), $, (),(), $);
#82=ITEM_NAMES(LABEL('PAW family'), (), LABEL('PAW'), $, $);

/* class / table relationship */
#1422=CLASS_TABLE_RELATIONSHIP(#81, (#120));

/* Dictionary table description */
#121=RDB_TABLE_ELEMENT(#120, $, '001', #122, TEXT('This table
defines the relationship (derivation) between the inner diameter and
the thickness of PAW'), $, $, *, (#96, #116), (#96));
#122=ITEM_NAMES (LABEL('d_in / e table'), (), LABEL(''), $, $);

/* Properties semantics definition */
#96=SELF_PROPERTY_VALUE_SEMANTICS(#90, $);
```

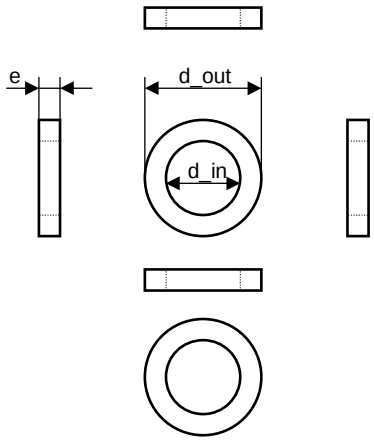


```
#106=SELF_PROPERTY_VALUE_SEMANTICS(#100, $);
#116=SELF_PROPERTY_VALUE_SEMANTICS(#110, $);

/* Properties syntax definition */
#97=REAL_NUMERIC_VARIABLE();
#107=REAL_NUMERIC_VARIABLE();
#117=REAL_NUMERIC_VARIABLE();

/* Syntax / semantics association */
#98=ENVIRONMENT(#97, #96);
#108=ENVIRONMENT(#107, #106);
#118=ENVIRONMENT(#117, #116);

/* LIBRARY DESCRIPTION */

/* Library definition of the properties */
#900=OPT_OR_MAND_PROPERTY_BSU(#90, .F., .T.);
#1000=OPT_OR_MAND_PROPERTY_BSU(#100, .F., .T.);
#1100=OPT_OR_MAND_PROPERTY_BSU(#110, .F., .T.);

/* Dictionary extension */
#8000=ITEM_CLASS_EXTENSION(#60, (), (), (), '001', '001', (), (), *, *,
*, (#901), (#1001, #1101), (), (), (#900), (#1000, #1100), (), .F., $,
$, $, $, (), $, (), ());

/* Property domain */
#901=DOMAIN_RESTRICTION((#96), (), (#902), $);
#902=GUARDED_SIMPLE_DOMAIN(#903, #904);
#903=OTHERS();
#904=TABLE_DEFINED_DOMAIN(#905);
#905=TABLE_LITERAL(#120);

/* Extension of the table */
#910=RDB_TABLE_CONTENT(#120, *, (#911, #912), '001', '1997-12-19');
#911=REAL_COLUMN((10.0, 11.0, 13.0, 17.0, 19.0), 'NR2..3.3');
#912=REAL_COLUMN((1.0, 1.0, 2.0, 3.0, 4.0), 'NR2..3.3');

/* Definition of the derivation function 'd_out=d_in * 1.5' */
#1001=FUNCTIONAL_DOMAIN_RESTRICTION((#106), (#96), (#1002), $);
#1002=GUARDED_FUNCTIONAL_DOMAIN(#1003, #1004);
#1003=OTHERS();
#1004=LIBRARY_EXPRESSION_DEFINED_VALUE(#1005);
#1005=MULT_EXPRESSION((#1006, #97));
#1006=REAL_LITERAL(1.5);

/* Definition of the derivation table that computes 'e' from 'd_in' */
#1101=FUNCTIONAL_DOMAIN_RESTRICTION((#116), (#96), (#1102), $);
#1102=GUARDED_FUNCTIONAL_DOMAIN(#1103, #1104);
#1103=OTHERS();
#1104=TABLE_DEFINED_VALUE(#1105);
#1105=TABLE_LITERAL(#120);
```

```
ENDSEC;
END-ISO-10303-21;

/*
```

P.3.3.2 Example of functional model class descriptions using a standard external file protocol

This subclause contains the complete physical file presented in clause P.3.



P.3.3.2.1 Example of a functional model class using a standard program protocol

```
*/ISO-10303-21;
  HEADER;
  FILE_DESCRIPTION(('PLIB FUNCTIONAL MODEL EXAMPLE 1'), '1');
  FILE_NAME('P24_fm_implicit_p101.spf',
            '2000-06-05T02:38:14',
            (''),
            ('LISI/ENSMA'),
            'ECCO RUNTIME SYSTEM BUILT-IN PREPROCESSOR V2.3.3',
            'ECCO RUNTIME SYSTEM V2.3.3',
            '');
  FILE_SCHEMA(('ISO13584_F_M_IIM_LIBRARY_IMPLICIT_SCHEMA'));
  ENDSEC;

  DATA;
```

/* Global library description */


```
#2=LIBRARY_IN_STANDARD_FORMAT($, $, $, $, (), #30, #11, (#7), (#12),
(#20, #30, #40), (#50, #60, #140, #130), (), #3, $, $, ());
#3=ITEM_NAMES(LABEL('Functional geometry model'), (), LABEL(''), $,
$);
#6=ORGANIZATION('ISO', 'International Organization for Standardization',
'');
#7=STANDARD_SIMPLE_PROGRAM_PROTOCOL(#6, $, 'ISO_13584_31', '001', '1',
#8, $, 'FORTRAN', .SOURCE., $, $, $);
#8=ITEM_NAMES(LABEL('ISO_13584_31'), (), LABEL('ISO_13584_31'), $, $);
#11=LIBRARY_IIM_IDENTIFICATION($, 'IS', 'ISO_13584_24_2', 2001, '3', $,
());
#12=VIEW_EXCHANGE_PROTOCOL_IDENTIFICATION($, 'IS', 'ISO_13584_101',
2001, '1', '1', (#7), $);
#10=GLOBAL_LANGUAGE_ASSIGNMENT('en');
```

/* DICTIONARY DESCRIPTION */

```
/* BSU for supplier */
#20=SUPPLIER_BSU('INA', *);  /* INA code unknown */
#30=SUPPLIER_BSU('9/19860073600021', *);
   /* LISI/ENSMA code in the coding scheme ICD=0009: SIRET number */
#40=SUPPLIER_BSU('0112/1///13584_101_1', *);
   /* Identification of ISO 13584-101 according to ISO 13584-26 */
```

/* BSU for component_class */

```
#50=CLASS_BSU('Bearing', '001', #20);
#60=CLASS_BSU('PAW', '001', #20);
#130=CLASS_BSU('PAW_Geometry', '001', #30);
#140=CLASS_BSU('basic_geometry', '001', #40);
```

/* BSU for properties */

```
#90=PROPERTY_BSU('d_in', '001', #50);
#100=PROPERTY_BSU('d_out', '001', #50);
#110=PROPERTY_BSU('e', '001', #50);
#150=PROPERTY_BSU('geometry_level', '001', #140);
#160=PROPERTY_BSU('detail_level', '001', #140);
#170=PROPERTY_BSU('side', '001', #140);
#180=PROPERTY_BSU('prg', '001', #130);
#190=PROPERTY_BSU('required_side', '001', #130);
#200=PROPERTY_BSU('variant', '001', #140);
#210=PROPERTY_BSU('unreg_variant', '001', #140);
```

/* BSU for table */

```
#230=TABLE_BSU('T2', '001', #130);
```

/* v_c_v range */

```
#155=VIEW_CONTROL_VARIABLE_RANGE(#150, 1, 1);
#165=VIEW_CONTROL_VARIABLE_RANGE(#160, 2, 2);
#175=VIEW_CONTROL_VARIABLE_RANGE(#170, 1, 6);
#205=VIEW_CONTROL_VARIABLE_RANGE(#200, 1, 1);
#215=VIEW_CONTROL_VARIABLE_RANGE(#210, 0, 0);
```

```
/* supplier description */
#31=SUPPLIER_ELEMENT(#30, $, '001', #32, #33);
#32=ORGANIZATION('LISI/ENSMA', 'LISI/ENSMA', '');
#33=ADDRESS($, $, $, $, $, $, $, 'FRANCE', $, $, $, $);

/* Dictionary table description */
#231=TABLE_ELEMENT(#230, $, '001', #232, TEXT('Definition of the
geometry programs according to the side of the part'), $, $, *, (#196,
#186), (#196));
#232=ITEM_NAMES(LABEL('side / prg table'), (), LABEL(''), $, $);

/* Dictionary properties description */
/* prg */
#91=REPRESENTATION_P_DET (#180, $, '001', #92, TEXT('variable used to
reference geometry programs'), $, $, $, $, (), $, 'A58', #93, $);
#92=ITEM_NAMES (LABEL('related program'), (), LABEL(''), $, $);
#93=PROGRAM_REFERENCE_TYPE ((
'ISO13584_F_M_IIM_LIBRARY_IMPLICIT_SCHEMA.PROGRAM_REFERENCE'));

/* required side */
#101=REPRESENTATION_P_DET(#190, $, '001', #102, TEXT('property used
to store the required side'), $, $, $, $, (), $, 'A58', #103, $);
#102=ITEM_NAMES(LABEL('side to be represented'), (), LABEL(''), $,
$);
#103=INT_TYPE('N 1');

/* class - table relationship */
#1424=CLASS_TABLE_RELATIONSHIP(#71, (#230));

/* Dictionary class description */
/* Functional model class view_of definition*/
#71=FM_CLASS_VIEW_OF(#130, $, '001', #72, TEXT('Functional model
class describing the 2d standard geometry of PAW'), $, $, $, $, (#180,
#190), (), *, *, *, *, *, #140, (#155, #165, #175, #205, #215), (#150,
#160, #170, #200, #210), (), (), (), (), (), (), (), #60, (#90,
#100, #110), (),(),());
#72=ITEM_NAMES(LABEL('Functional model class of PAW'), (), LABEL('fm
class of PAW'), $, $);

/* Definition of the properties semantics */
#176=OPEN_VIEW_PROPERTY_VALUE_SEMANTICS(#170, $);
#186=SELF_PROPERTY_VALUE_SEMANTICS(#180, $);
#196=SELF_PROPERTY_VALUE_SEMANTICS(#190, $);
#206=COLUMN_TRAVERSAL_VARIABLE_SEMANTICS(#2407, #196);

#96 =SELF_PROPERTY_VALUE_SEMANTICS(#90, $);
#106=SELF_PROPERTY_VALUE_SEMANTICS(#100, $);
#116=SELF_PROPERTY_VALUE_SEMANTICS(#110, $);

/* Properties syntax definition */
#177=INT_NUMERIC_VARIABLE();
#187=ENTITY_INSTANCE_VARIABLE((
```



```
'ISO13584_F_M_IIM_LIBRARY_IMPLICIT_SCHEMA.PROGRAM_REFERENCE'));
#197=INT_NUMERIC_VARIABLE();
#207=INT_NUMERIC_VARIABLE();
#97 =REAL_NUMERIC_VARIABLE();
#107=REAL_NUMERIC_VARIABLE();
#117=REAL_NUMERIC_VARIABLE();
```

/* Syntax / semantics association */

```
#178=ENVIRONMENT(#177, #176);
#188=ENVIRONMENT(#187, #186);
#198=ENVIRONMENT(#197, #196);
#208=ENVIRONMENT(#207, #206);
#98 =ENVIRONMENT(#97, #96);
#108=ENVIRONMENT(#107, #106);
#118=ENVIRONMENT(#117, #116);
```

/* LIBRARY DESCRIPTION */

/* Extension of the table */

```
#2300=TABLE_CONTENT(#230, *, (#2301, #2302), '001', '1997-12-19');
#2301=INTEGER_COLUMN((1, 2, 3, 4, 5, 6), 'NR1..1');
#2302=ENTITY_INSTANCE_COLUMN((#2303, #2304, #2305, #2306, #2307,
#2308),
('ISO13584_F_M_IIM_LIBRARY_IMPLICIT_SCHEMA.PROGRAM_REFERENCE'));
#2303=PROGRAM_REFERENCE(#7, #2313, 'Add1_PAW', 'PAW_p1', (), (), ());
#2304=PROGRAM_REFERENCE(#7, #2314, 'Add2_PAW', 'PAW_p2', (), (), ());
#2305=PROGRAM_REFERENCE(#7, #2315, 'Add3_PAW', 'PAW_p3', (), (), ());
#2306=PROGRAM_REFERENCE(#7, #2316, 'Add4_PAW', 'PAW_p4', (), (), ());
#2307=PROGRAM_REFERENCE(#7, #2317, 'Add5_PAW', 'PAW_p5', (), (), ());
#2308=PROGRAM_REFERENCE(#7, #2318, 'Add6_PAW', 'PAW_p6', (), (), ());
#2313=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2323));
#2314=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2324));
#2315=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2325));
#2316=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2326));
#2317=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2327));
#2318=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2328));
#2323=LANGUAGE_SPECIFIC_CONTENT((#2333), #2333, $);
#2324=LANGUAGE_SPECIFIC_CONTENT((#2334), #2334, $);
#2325=LANGUAGE_SPECIFIC_CONTENT((#2335), #2335, $);
#2326=LANGUAGE_SPECIFIC_CONTENT((#2336), #2336, $);
#2327=LANGUAGE_SPECIFIC_CONTENT((#2337), #2337, $);
#2328=LANGUAGE_SPECIFIC_CONTENT((#2338), #2338, $);
#2333=EXTERNAL_FILE_UNIT('PAW_p1.for', '7bit');
#2334=EXTERNAL_FILE_UNIT('PAW_p1.for', '7bit');
#2335=EXTERNAL_FILE_UNIT('PAW_p1.for', '7bit');
#2336=EXTERNAL_FILE_UNIT('PAW_p1.for', '7bit');
#2337=EXTERNAL_FILE_UNIT('PAW_p1.for', '7bit');
#2338=EXTERNAL_FILE_UNIT('PAW_p1.for', '7bit');
```

/* Library definition of the properties */

```
#900 =OPT_OR_MAND_PROPERTY_BSU(#90, .F., .T.);
```

```
#1000=OPT_OR_MAND_PROPERTY_BSU(#100, .F., .T.);
#1100=OPT_OR_MAND_PROPERTY_BSU(#110, .F., .T.);
#1700=OPT_OR_MAND_PROPERTY_BSU(#170, .F., .T.);
#1800=OPT_OR_MAND_PROPERTY_BSU(#180, .F., .F.);
#1900=OPT_OR_MAND_PROPERTY_BSU(#190, .F., .F.);

/* Functional model class extension */
#1300=FUNCTIONAL_MODEL_CLASS_EXTENSION(#130, (#2303, #2304, #2305,
#2306, #2307, #2308), (#7), (#12),'001', '001', (), (), *, *, *, (), (),
(), $, (#900, #1000, #1100), (), (), (#1700, #1800, #1900), (),
(#3000), $, $, (), $);

/* Definition of the derivation table that computes 'prg' from 'side' */
#2401=FUNCTIONAL_DOMAIN_RESTRICTION((#186), (#176, #196, #206),
(#2402), $);
#2402=GUARDED_FUNCTIONAL_DOMAIN(#2403, #2405);
#2403=OTHERS();
#2405=TABLE_DEFINED_VALUE(#2406);
#2406=SELECT_EXPRESSION((#2407, #2408));
#2407=TABLE_LITERAL(#230);
#2408=EQUALS_EXPRESSION((#207, #177));

/* Definition of the methods */
#3000=METHOD(#3001, #3002, #7);
#3001=METHOD_SPECIF(#140, (#155, #165, #175, #205, #215), (#90, #100,
#110),
());
#3002=METHOD_BODY((#97, #107, #117, #177, #187, #197, #207),
(#3019));

#3019=METHOD_STATEMENT((#3020));
#3020=GUARDED_STATEMENT(#3021, #3023);
#3021=BOOLEAN_LITERAL(.T.);
#3023=CALL_PROGRAM_STATEMENT(#187, #2401, (#97, #107, #117), (),
());

ENDSEC;
END-ISO-10303-21;

/*
```

This ISO 10303-21 conformant physical file is associated with six external files that contain the parametric specifications of the PAW geometry for the different 2D side views. This parametric specification is exchanged as parametric programs conformant with ISO 13584-31 and is supposed to be generated by some parametric geometry editor.

The following file, named Add1_PAW shows an example of such an automatically generated parametric program. This file has been automatically generated from the EBP (Example Based Programming) system developed in order to support the generation of programs (in this example, the program that generates a front view is called PAW_p1)from a user design.

```
SUBROUTINE PAW_p1 (d_out, d_int)
! implicit declarations
```



```
!
! entity types: (d)ir, (p)nt, (l)in, (c)ir,
!      (g)rp, (s)et, (e)nt, (a)rc or (a)2p
IMPLICIT INTEGER (d,p,l,c,g,s,e,a,n)
!
! transfo_type(n) are strings that will contain
! "mirror", "shift", "rotation" or "homotetia"
IMPLICIT CHARACTER*(80) (t)
!
! prefix of D.P. var used: (r)adius, (v)al
IMPLICIT DOUBLE PRECISION (r,v)
!
! global constants
INTEGER TDB, CAD
DOUBLE PRECISION ZERO_VALUE
PARAMETER (TDB = 0,CAD = 1)
PARAMETER (ZERO_VALUE = 0.001)
INTEGER lstent (1000)
INTEGER FALSE, TRUE
!
! parameter declarations:
!
DOUBLE PRECISION d_out
DOUBLE PRECISION d_int
!
! include the types of the P31 and LIB functions
include 'P31_FUNCTIONS_TYPES'
include 'LIB_FUNCTIONS_TYPES'
!
! *** Program Body ***
!
! initialise some constant used entities
grpfix = CREATE_GRP ()
CALL CLOSE_GRP ()
a2p_ref = a2p_ref_sys (TDB)
dir_x = dir_a2p_x (a2p_ref, TDB)
dir_y = dir_a2p_y (a2p_ref, TDB)
pnt_origin = PNT_CARTESIAN_ABSOLUTE(0.0D0, 0.0D0, 0.0D0, TDB)
FALSE = 0
TRUE = 1
CALL INQ_GEOMETRICAL_POWER(POWER,ERR)
three_d = (POWER .GT. 1)
if three_d then
dir_z = dir_a2p_z (a2p_ref, TDB)
end if
!
!
!
! *** START OF THE RECORDING SESSION ***
!
!
```

```
! horizontal line of a given y value
pnt1 = PNT_CARTESIAN_ABSOLUTE(0.0D0, 0.0D0, 0.0D0, TDB)
lin1C1 = lib_lin_max(LIN_PNT_LENGTH_DIR(pnt1, 1.0D0, dir_x, TDB))
CALL ADD_ENT_GRP(grpfix, lin1C1)
!
! vertical line of a given x value
pnt1 = PNT_CARTESIAN_ABSOLUTE(0.0D0, 0.0D0, 0.0D0, TDB)
lin2C1 = lib_lin_max(LIN_PNT_LENGTH_DIR(pnt1, 1.0D0, dir_y, TDB))
CALL ADD_ENT_GRP(grpfix, lin2C1)
!
! intersection of 2 lines
pntnm1 = PNT_INTERSECTION_2_ENT(lin2C1, lin1C1, TDB)
!
! circle by its centre and its radius
cir3C1 = CIRCLE_RAD_A2P ((d_out) / (2.000D0), &
lib_a2p_pnt(pntnm1),false, TDB)
CALL ADD_ENT_GRP(grpfix, cir3C1)
!
! centre of a circle
pntnm2 = PNT_CENTRE_ARC(cir3C1, TDB)
! circle by its centre and its radius
cir4C1 = CIRCLE_RAD_A2P ((d_int) / (2.000D0), &
lib_a2p_pnt(pntnm2),false, TDB)
CALL ADD_ENT_GRP(grpfix, cir4C1)
!
! arc by a circle
arc5C1 = cir3C1
CALL ADD_ENT_GRP(grpfix, arc5C1)
!
! arc by a circle
arc6C1 = cir4C1
CALL ADD_ENT_GRP(grpfix, arc6C1)
!
! *** FIX ENTITIES into CAD SYSTEM ***
!
lstent (1) = grpfix
CALL FIX_ENT (1,lstent)
RETURN
END
```

P.3.3.2.2 A second example of a functional model class using a nonstandard program protocol based on the Pro-Engineer CAD system

```
  */
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('PLIB FUNCTIONAL MODEL EXAMPLE 2'), '1');
FILE_NAME('P24_fm_implicit_p102.spf',
          '2000-06-05T02:54:41',
          (''),
          ('LISI/ENSMA'),
          'ECCO RUNTIME SYSTEM BUILT-IN PREPROCESSOR V2.3.3',
          'ECCO RUNTIME SYSTEM V2.3.3',
```



```
          '');
FILE_SCHEMA(('ISO13584_F_M_IIM_LIBRARY_IMPLICIT_SCHEMA'));
ENDSEC;

DATA;
```

/* Global library description */

```
#2=LIBRARY_IN_STANDARD_FORMAT($, $, $, $, (), #30, #11, (#7), (#12),
(#20, #30, #40), (#50, #60, #140, #130), (), #3, $, $, ());
#3=ITEM_NAMES(LABEL('Functional geometry model'), (), LABEL(''), $,
$);
#6=ORGANIZATION('Parametric Technology Corporation', 'PTC', '');
#7=NON_STANDARD_SIMPLE_PROGRAM_PROTOCOL(#6, 'United States',
'ProEngineerInterface', '001', $, #8, $, 'BINARY', .COMPILED., #6, '',
'001');
#8=ITEM_NAMES(LABEL('Pro-Engineer interface'), (), LABEL('Pro-Engineer
interface'), $, $);
#11=LIBRARY_IIM_IDENTIFICATION($, 'IS', 'ISO_13584_24_2', 2001, '3E', $,
());
#12=VIEW_EXCHANGE_PROTOCOL_IDENTIFICATION($, 'IS', 'ISO_13584_101',
2001, $, $, (#7), $);
#10=GLOBAL_LANGUAGE_ASSIGNMENT('en');
```

/* DICTIONARY DESCRIPTION */

/*BSU for supplier */

```
#20=SUPPLIER_BSU('INA', *); /* Unknown Id */
/* LISI/ENSMA code in the coding scheme ICD = 0009: SIRET number */
#30=SUPPLIER_BSU('9/19860073600021', *);
/* Identification of ISO 13584-101 according to ISO 13584-26 */
#40=SUPPLIER_BSU('0112/1///13584_101_1', *);
```

/* BSU for component_class */

```
#50=CLASS_BSU('Bearing', '001', #20);
#60=CLASS_BSU('PAW', '001', #20);
#130=CLASS_BSU('PAW_Geometry', '001', #30);
#140=CLASS_BSU('basic_geometry', '001', #40);
```

/* BSU for properties */

```
#90=PROPERTY_BSU('d_in', '001', #50);
#100=PROPERTY_BSU('d_out', '001', #50);
#110=PROPERTY_BSU('e', '001', #50);
#150=PROPERTY_BSU('geometry_level', '001', #140);
#160=PROPERTY_BSU('detail_level', '001', #140);
#170=PROPERTY_BSU('side', '001', #140);
#180=PROPERTY_BSU('prg', '001', #130);
#190=PROPERTY_BSU('required_side', '001', #130);
#200=PROPERTY_BSU('variant', '001', #140);
#210=PROPERTY_BSU('unreg_variant', '001', #140);
```

/* BSU for table */

```
#230=TABLE_BSU('T2', '001', #130);

/* v_c_v range */
#155=VIEW_CONTROL_VARIABLE_RANGE(#150, 1, 1);
#165=VIEW_CONTROL_VARIABLE_RANGE(#160, 2, 2);
#175=VIEW_CONTROL_VARIABLE_RANGE(#170, 1, 6);
#205=VIEW_CONTROL_VARIABLE_RANGE (#200, 0, 0);
#215=VIEW_CONTROL_VARIABLE_RANGE (#210, 1, 1);

/* supplier description */
#31=SUPPLIER_ELEMENT(#30, $, '001', #32, #33);
#32=ORGANIZATION('LISI/ENSMA', 'LISI/ENSMA', '');
#33=ADDRESS($, $, $, $, $, $, $, 'FRANCE', $, $, $, $);

/* Dictionary table description */
#231=TABLE_ELEMENT(#230, $, '001', #232, TEXT('Definition of the
geometry programs according to the side of the part'), $, $, *, (#196,
#186), (#196));
#232=ITEM_NAMES(LABEL('required side / prg table'), (), LABEL(''), $,
$);

/* Dictionary properties description */
/* prg */
#91=REPRESENTATION_P_DET(#180, $, '001', #92, TEXT('variable used to
reference geometry programs'), $, $, $, $, (), $, 'A58', #93, $);
#92=ITEM_NAMES (LABEL('related program'), (), LABEL('prg'), $, $);
#93=PROGRAM_REFERENCE_TYPE((
'ISO13584_F_M_IIM_LIBRARY_IMPLICIT_SCHEMA.PROGRAM_REFERENCE'));

/* required side */

#101=REPRESENTATION_P_DET(#190, $, '001', #102, TEXT('property used to
store the required side'), $, $, $, $, (), $, 'A58', #103, $);
#102=ITEM_NAMES(LABEL('side to be represented'), (), LABEL('reqside'),
$, $);
#103=INT_TYPE('N 1');

/* class - table relationship */
#1424=CLASS_TABLE_RELATIONSHIP(#71, (#230));

/* Dictionary class description */
/* Functional model class view_of definition*/
#71=FM_CLASS_VIEW_OF(#130, $, '001', #72, TEXT('Functional model
class describing the 2d standard geometry of PAW'), $, $, $, $, (#180,
#190), (), *, *, *, *, *, #140, (#155, #165, #175, #205, #215), (#150,
#160, #170, #200, #210), (), (), (), (), (), (), (), (), #60, (#90,
#100, #110), (),(),());
#72=ITEM_NAMES(LABEL('Functional model class of PAW'), (), LABEL('fm
class of PAW'), $, $);

/* Definition of the properties semantics */
#176=OPEN_VIEW_PROPERTY_VALUE_SEMANTICS(#170, $);
```



```
#186=SELF_PROPERTY_VALUE_SEMANTICS(#180, $);
#196=SELF_PROPERTY_VALUE_SEMANTICS(#190, $);
#206=COLUMN_TRAVERSAL_VARIABLE_SEMANTICS(#2407, #196);

#96=SELF_PROPERTY_VALUE_SEMANTICS(#90, $);
#106=SELF_PROPERTY_VALUE_SEMANTICS(#100, $);
#116=SELF_PROPERTY_VALUE_SEMANTICS(#110, $);
```

/* Properties syntax definition */

```
#177=INT_NUMERIC_VARIABLE();
#187=ENTITY_INSTANCE_VARIABLE((
'ISO13584_F_M_IIM_LIBRARY_IMPLICIT_SCHEMA.PROGRAM_REFERENCE'));
#197=INT_NUMERIC_VARIABLE();
#207=INT_NUMERIC_VARIABLE();
#97=REAL_NUMERIC_VARIABLE();
#107=REAL_NUMERIC_VARIABLE();
#117=REAL_NUMERIC_VARIABLE();
```

/* Syntax / semantics association */

```
#178=ENVIRONMENT(#177, #176);
#188=ENVIRONMENT(#187, #186);
#198=ENVIRONMENT(#197, #196);
#208=ENVIRONMENT(#207, #206);
#98=ENVIRONMENT(#97, #96);
#108=ENVIRONMENT(#107, #106);
#118=ENVIRONMENT(#117, #116);
```

/* LIBRARY DESCRIPTION */

/* Extension of the table */

```
#2300=TABLE_CONTENT(#230, *, (#2301, #2302), '001', '1997-12-19');
#2301=INTEGER_COLUMN((1, 2, 3, 4, 5, 6), 'NR1..1');
#2302=ENTITY_INSTANCE_COLUMN((#2303, #2304, #2305, #2306, #2307,
#2308), (
'ISO13584_F_M_IIM_LIBRARY_IMPLICIT_SCHEMA.PROGRAM_REFERENCE'));
#2303=PROGRAM_REFERENCE(#7, #2313, 'Add1_PAW', 'PAW_p1', (), (), ());
#2304=PROGRAM_REFERENCE(#7, #2314, 'Add2_PAW', 'PAW_p2', (), (), ());
#2305=PROGRAM_REFERENCE(#7, #2315, 'Add3_PAW', 'PAW_p3', (), (), ());
#2306=PROGRAM_REFERENCE(#7, #2316, 'Add4_PAW', 'PAW_p4', (), (), ());
#2307=PROGRAM_REFERENCE(#7, #2317, 'Add5_PAW', 'PAW_p5', (), (), ());
#2308=PROGRAM_REFERENCE(#7, #2318, 'Add6_PAW', 'PAW_p6', (), (), ());
#2313=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2323));
#2314=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2324));
#2315=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2325));
#2316=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2326));
#2317=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2327));
#2318=NOT_TRANSLATABLE_EXTERNAL_CONTENT((#2328));
#2323=LANGUAGE_SPECIFIC_CONTENT((#2333), #2333, $);
#2324=LANGUAGE_SPECIFIC_CONTENT((#2334), #2334, $);
#2325=LANGUAGE_SPECIFIC_CONTENT((#2335), #2335, $);
#2326=LANGUAGE_SPECIFIC_CONTENT((#2336), #2336, $);
```

```
#2327=LANGUAGE_SPECIFIC_CONTENT((#2337), #2337, $);
#2328=LANGUAGE_SPECIFIC_CONTENT((#2338), #2338, $);
#2333=EXTERNAL_FILE_UNIT('PAW_p1.ptc', '7bit');
#2334=EXTERNAL_FILE_UNIT('PAW_p1.ptc', '7bit');
#2335=EXTERNAL_FILE_UNIT('PAW_p1.ptc', '7bit');
#2336=EXTERNAL_FILE_UNIT('PAW_p1.ptc', '7bit');
#2337=EXTERNAL_FILE_UNIT('PAW_p1.ptc', '7bit');
#2338=EXTERNAL_FILE_UNIT('PAW_p1.ptc', '7bit');

/* Library definition of the properties */
#900=OPT_OR_MAND_PROPERTY_BSU(#90, .F., .T.);
#1000=OPT_OR_MAND_PROPERTY_BSU(#100, .F., .T.);
#1100=OPT_OR_MAND_PROPERTY_BSU(#110, .F., .T.);
#1700=OPT_OR_MAND_PROPERTY_BSU(#170, .F., .T.);
#1800=OPT_OR_MAND_PROPERTY_BSU(#180, .F., .F.);
#1900=OPT_OR_MAND_PROPERTY_BSU(#190, .F., .F.);

/* Functional model class extension */
#1300=FUNCTIONAL_MODEL_CLASS_EXTENSION(#130, (#2303, #2304, #2305,
#2306, #2307, #2308), (#7), (#12),'001', '001', (), (), *, *, *,
(), (), (), $, (#900, #1000, #1100), (), (), (#1700, #1800, #1900), (),
(#3000), $, $, (), $);

/* Definition of the derivation table that computes 'prg' from 'side' */
#2401=FUNCTIONAL_DOMAIN_RESTRICTION((#186), (#176, #196, #206),
(#2402), $);
#2402=GUARDED_FUNCTIONAL_DOMAIN(#2403, #2405);
#2403=OTHERS();
#2405=TABLE_DEFINED_VALUE(#2406);
#2406=SELECT_EXPRESSION((#2407, #2408));
#2407=TABLE_LITERAL(#230);
#2408=EQUALS_EXPRESSION((#207, #177));

/* Definition of the methods */
#3000=METHOD(#3001, #3002, #7);
#3001=METHOD_SPECIF(#140, (#155, #165, #175, #205, #215), (#90, #100,
#110), ());
#3002=METHOD_BODY((#97, #107, #117, #177, #187, #197, #207), (#3019));

#3019=METHOD_STATEMENT((#3020));
#3020=GUARDED_STATEMENT(#3021, #3023);
#3021=BOOLEAN_LITERAL(.T.);
#3023=CALL_PROGRAM_STATEMENT(#187, #2401, (#97, #107, #117), (),
());

ENDSEC;
END-ISO-10303-21;

/*
```


Annex Q
(informative)

Guidelines for creating functional model classes

ISO 13584-42 provides rules and guidelines for creating hierarchies of parts families, i.e., hierarchies of general model classes. When creating new representation categories, and/or new view exchange protocols, intended to be represented by new functional model classes and, possibly, by new functional view classes, the question arises whether some property shall be defined within a general model class or within a functional model class. The following three rules provide guidelines for this choice.

Q.1 RULE 1 — Representation property Vs item characteristics

When the value of some property for some item inserted in the same environment may change without changing the item identification, this property shall be defined as a representation property in some functional model class.

EXAMPLE 1 The price of a part, in the same currency, may change over the time. Therefore, this property is not a part characteristics, it is a representation property that shall be defined within a functional model class.

EXAMPLE 2 The radius of the sphere that constitutes the end of a needle in a needle bearing results from the production process of this needle. If the bearing supplier does not want to guarantee such a radius, this property is not a part characteristics. It shall be represented in the functional model of geometry.

Q.2 RULE 2 — Sharing of parts' characteristics

When a property is a part characteristics, i.e., its value (for a given part) may never change without changing the part, this property shall be defined within the general model class that models the family of this part.

EXAMPLE The threaded diameter of a screw cannot change without changing the screw. Moreover, this diameter is intended to be used in more than one representation category (for instance, geometry and any representation category that refers to mechanical behaviour, for instance stress analysis). Therefore this property shall be defined within the general model class of which one instance models this screw.

Q.3 RULE 3 — Context parameters Vs representation properties

Three criteria have been defined in order to define which properties should be considered as some context parameters or some representation properties:

a) When the value of some properties in a parts family are measured in some environmental condition, the context parameter(s) that characterise(s) this environment shall be represented, within the general model that models this parts family, as a context parameter.

EXAMPLE In the catalogue of a semi-conductor manufacturer, the value provided for the LOW-state dc input voltage, applied to a digital function of an integrated circuit, depends on two environmental conditions:the supply voltage and the temperature.

b) When a usual selection criteria of a part consist of some environmental condition that capture a user requirement, this environmental condition shall be represented as a context parameter in the general model class that models this family of part. These context parameters may be used both to filter the set of part that are proposed to the user and to compute some context-dependent characteristics that proves useful for the user selection process.

EXAMPLE When selecting a bearing, the dynamic load and the required rotation speed are usually used as selection criteria, they shall be modelled as context parameters.

NOTE A context parameter is not intended to be recorded after the selection process. Therefore, if such a context parameter is intended to be used in some functional model class, a different free model properties shall be defined in this functional model class.

c) When some environmental condition is mainly required to create a representation, in some context, of some part but not used for selection purpose, it shall be represented as a (selectable) representation property in the relevant functional model.

EXAMPLE When creating a geometric representation of a bolt + nut assembly, the distance from the nut to the head of the bolt is needed. This property shall be represented as a (selectable) representation property in the functional model of geometry. If it is also considered as a selection criteria for the assembly, a different property shall be modelled in the general model class as a context parameter.

Annex R
(informative)

EXPRESS-G diagrams

Figure R.1 through R.25 correspond to the EXPRESS schemas given in chapters 7 to 15. The diagrams use the EXPRESS-G graphical notation for the EXPRESS language. EXPRESS-G is defined in annex A of ISO 10303-11.

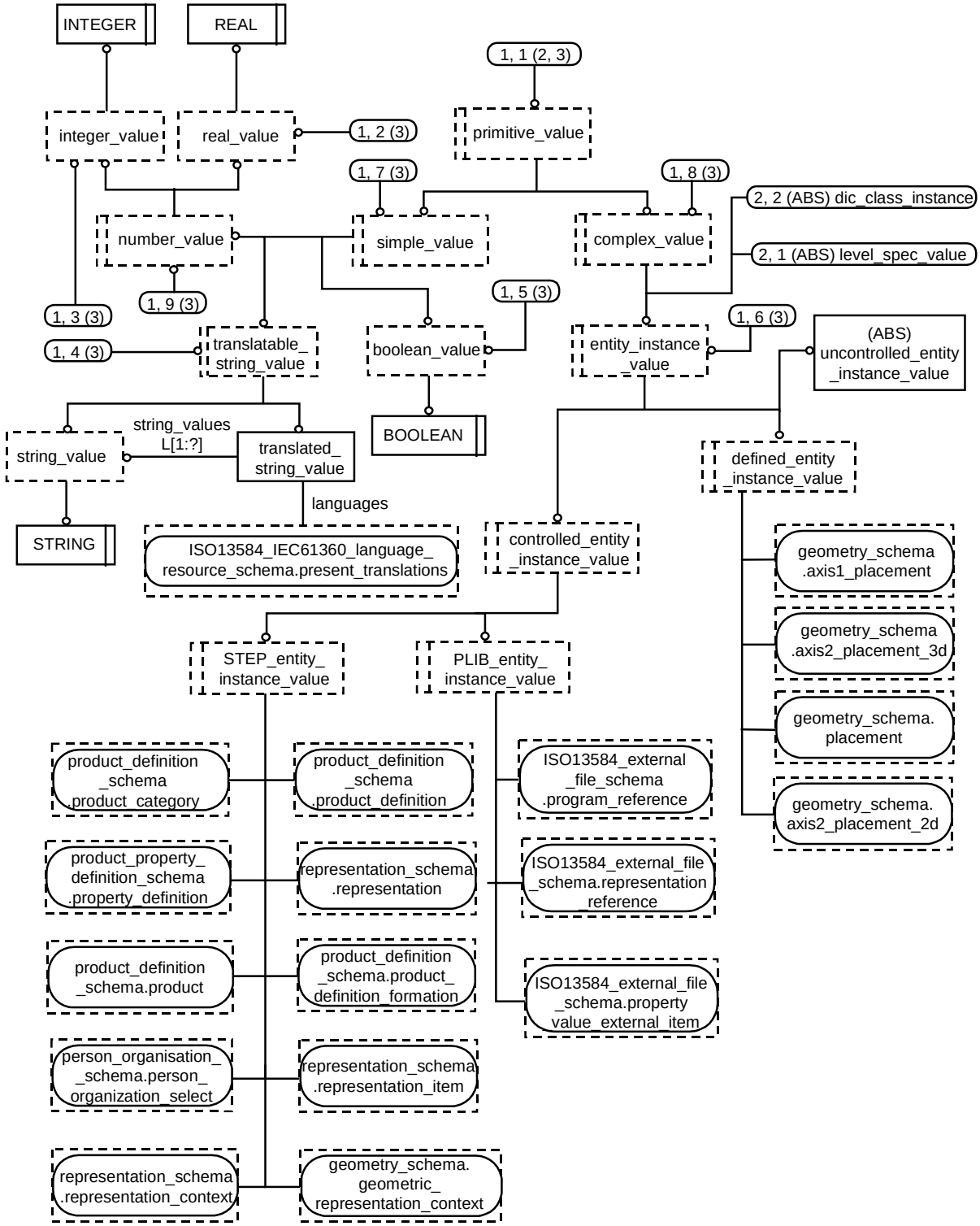


Figure R.1 — ISO13584_instance_resource_schema diagram 1 of 3

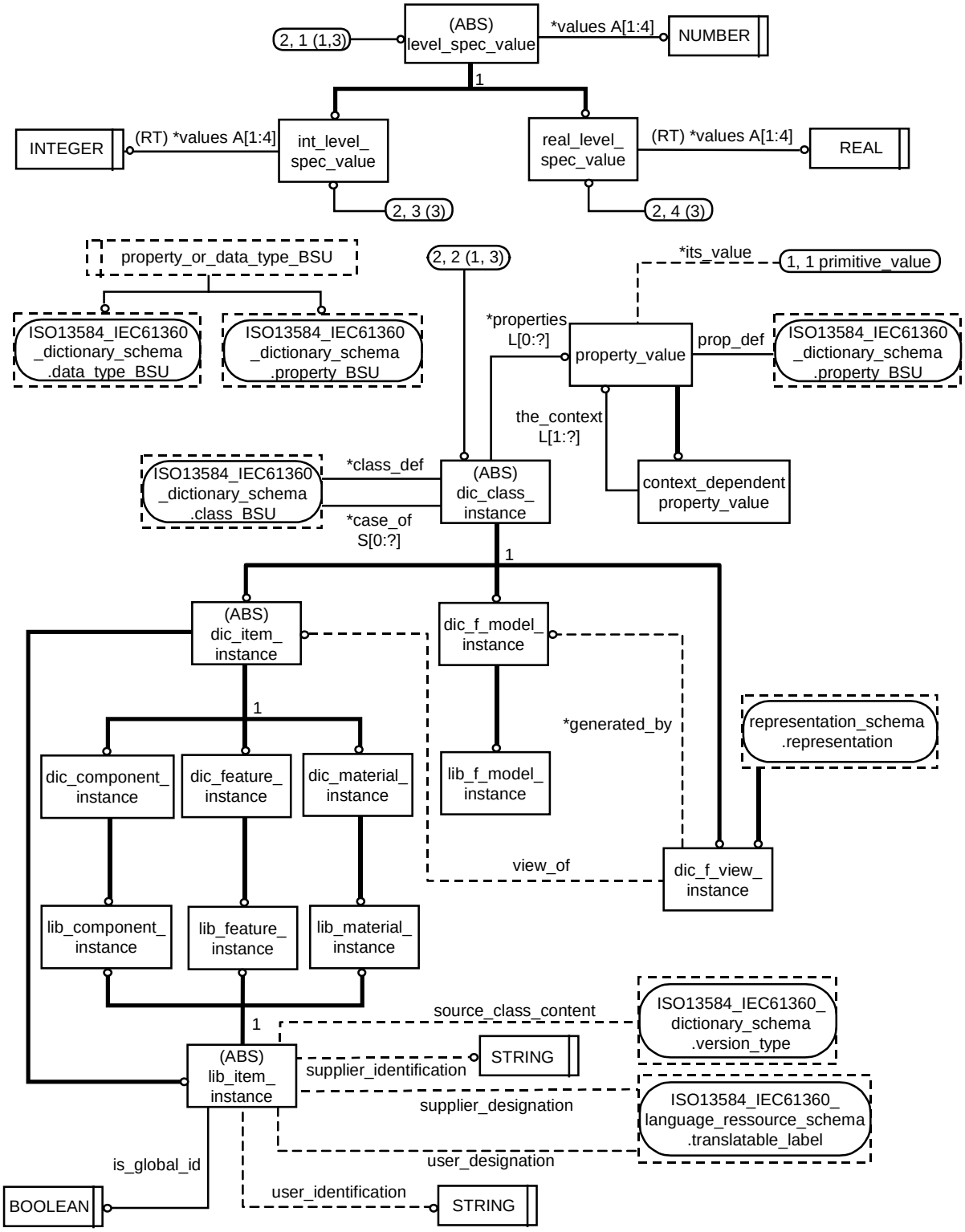


Figure R.2 — ISO13584_instance_resource_schema diagram 2 of 3

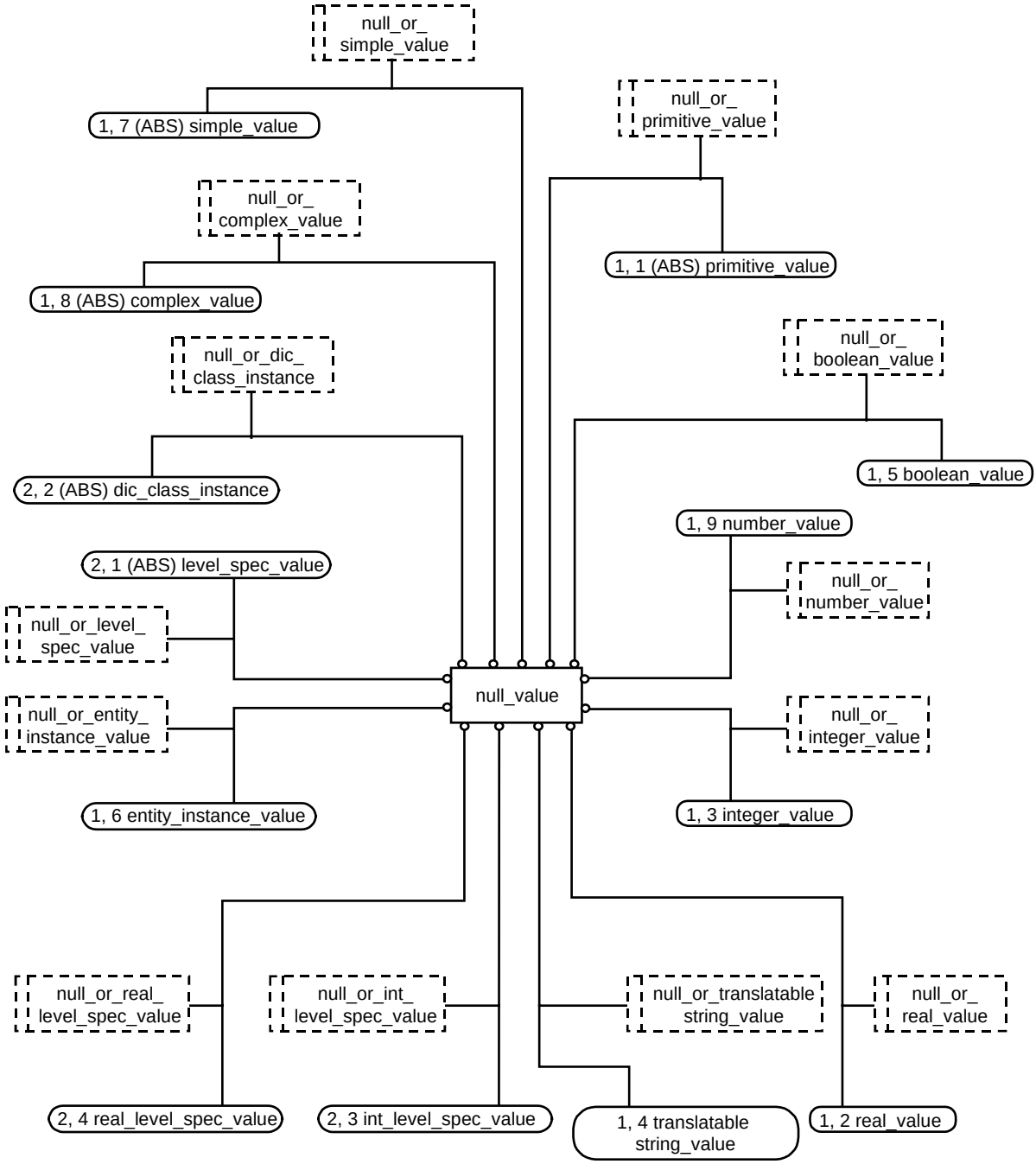


Figure R.3 — ISO13584_instance_resource_schema diagram 3 of 3

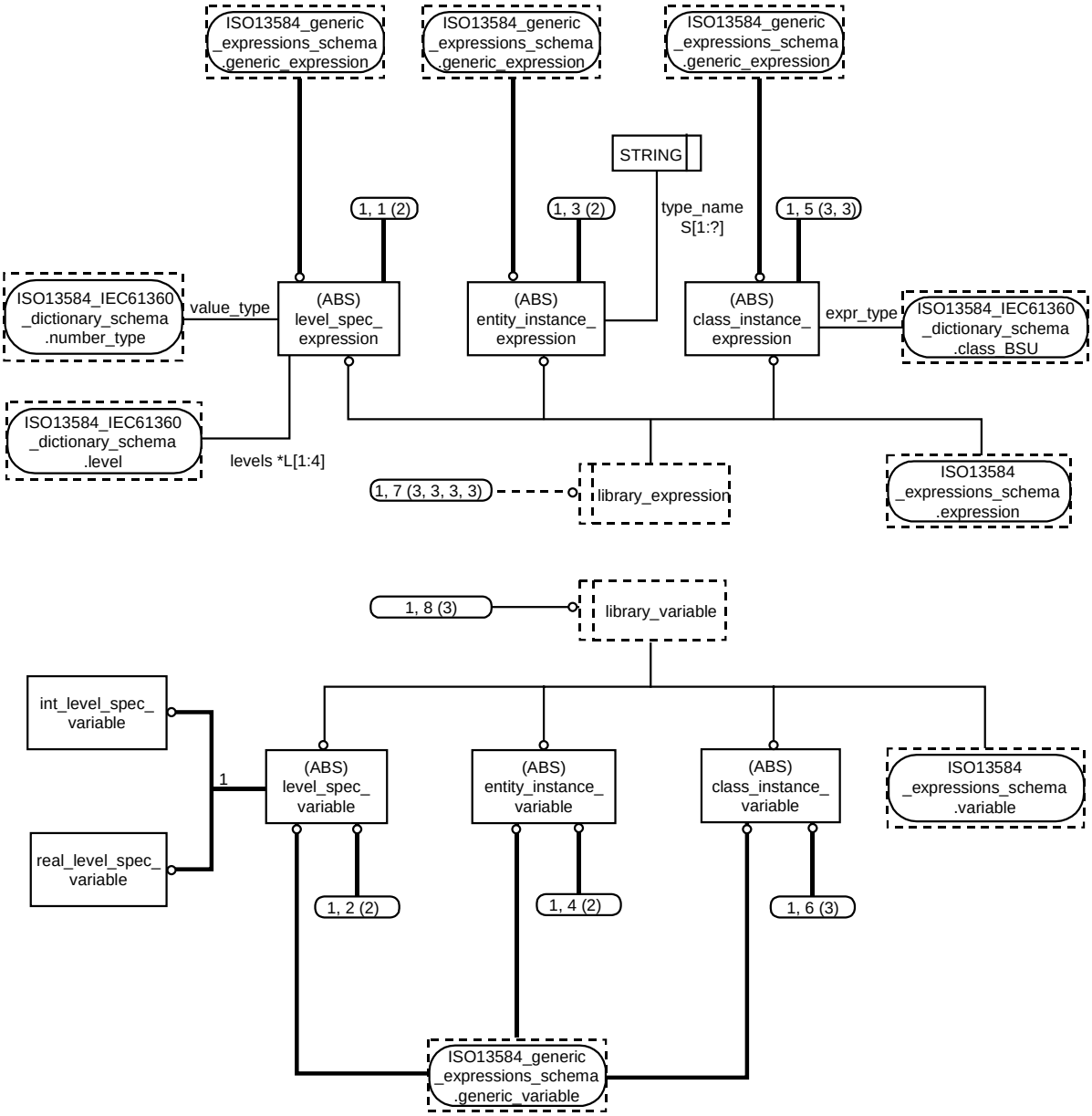


Figure R.4 — ISO13584_library_expressions_schema diagram 1 of 3

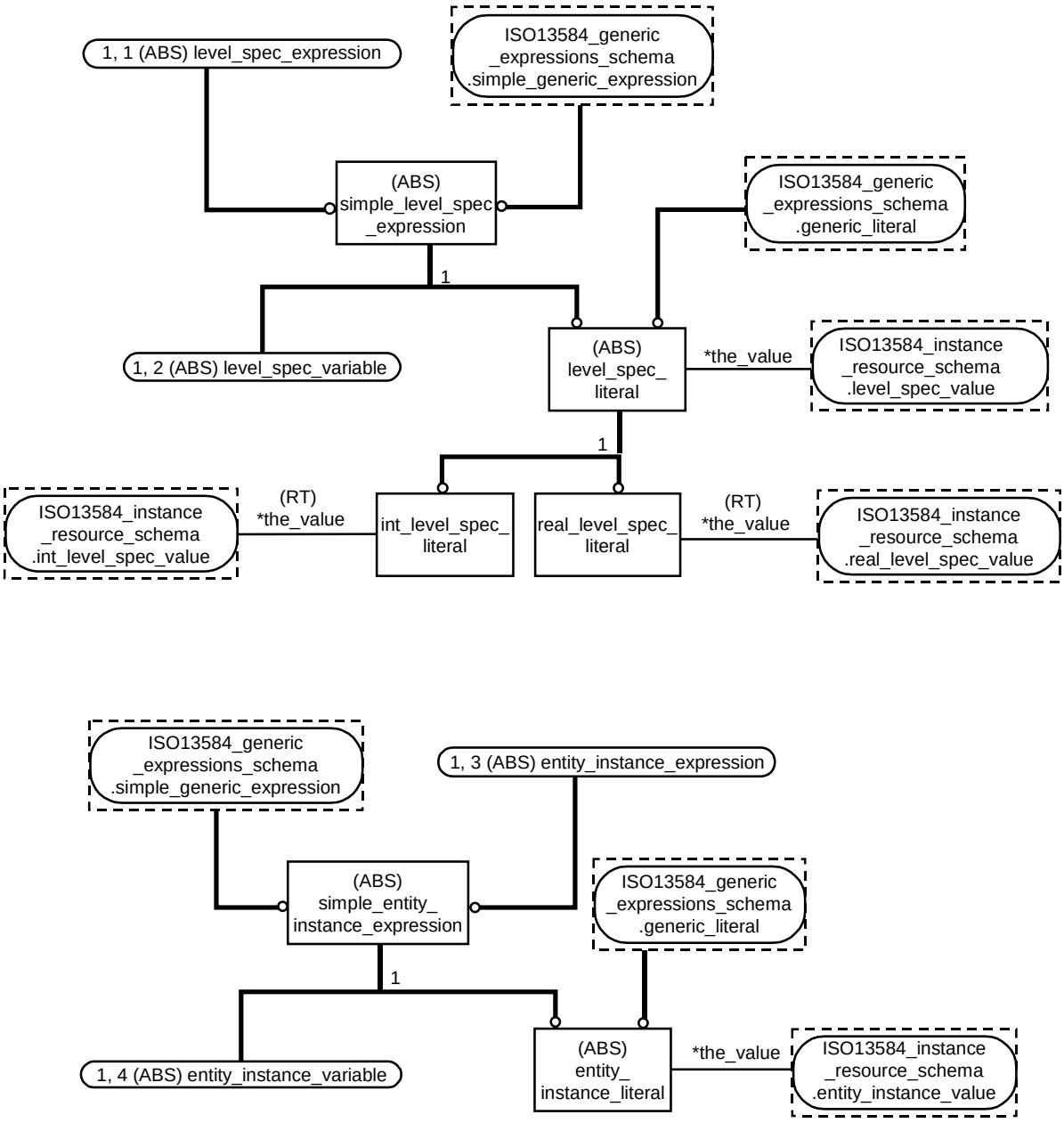


Figure R.5 — ISO13584_library_expressions_schema diagram 2 of 3

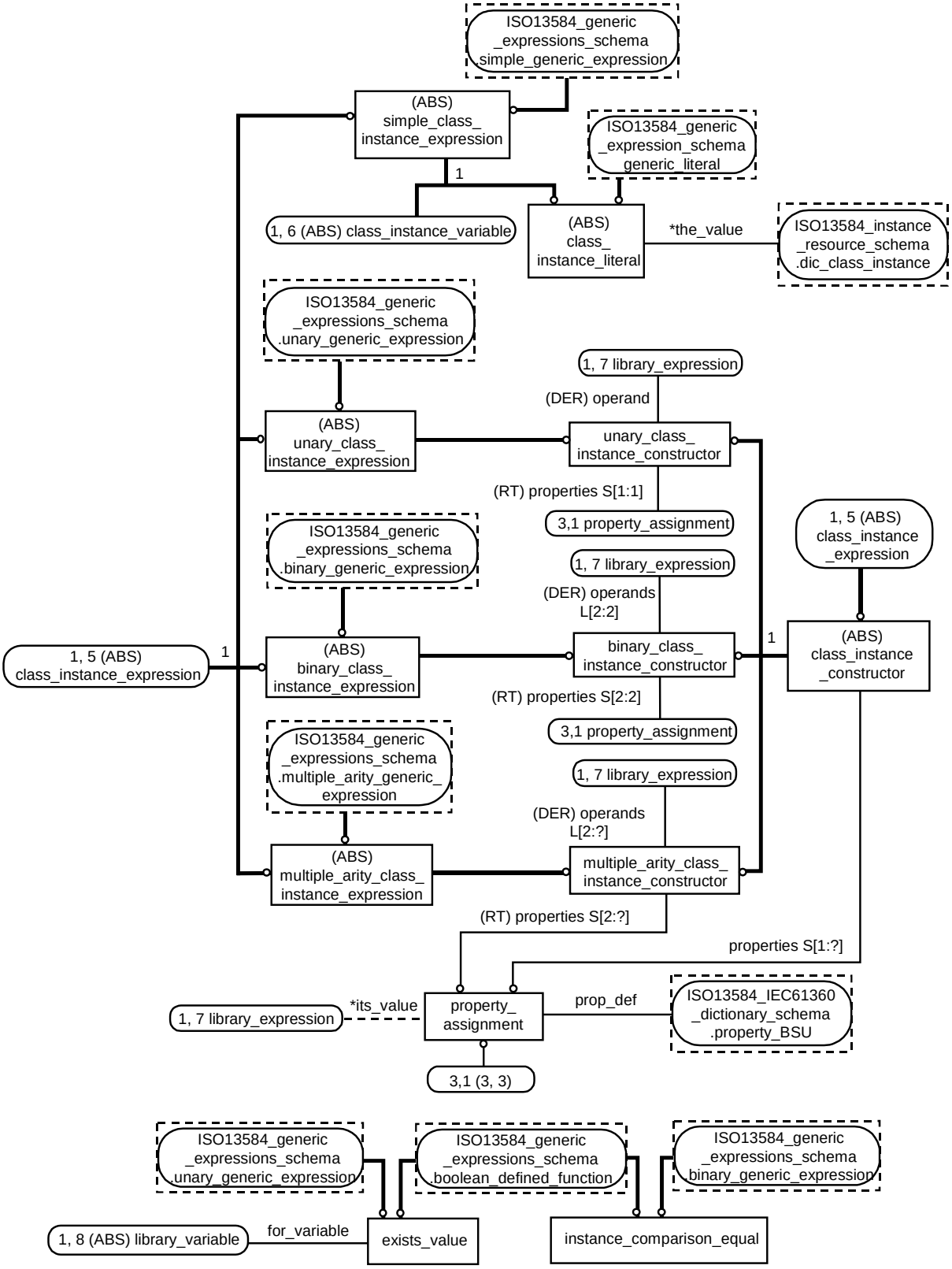


Figure R.6 — ISO13584_library_expressions_schema diagram 3 of 3

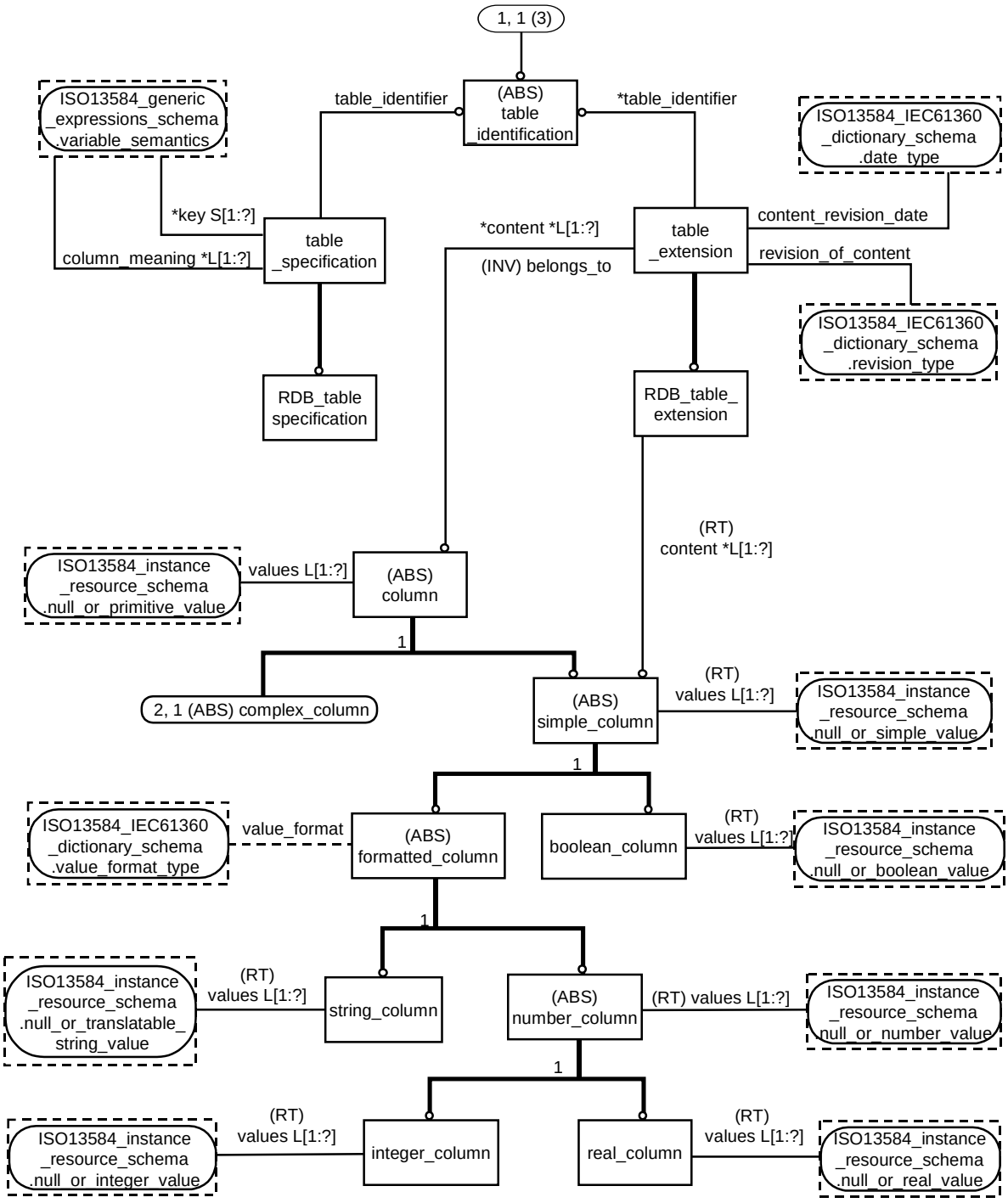


Figure R.7 — ISO13584_table_resource_schema diagram 1 of 4

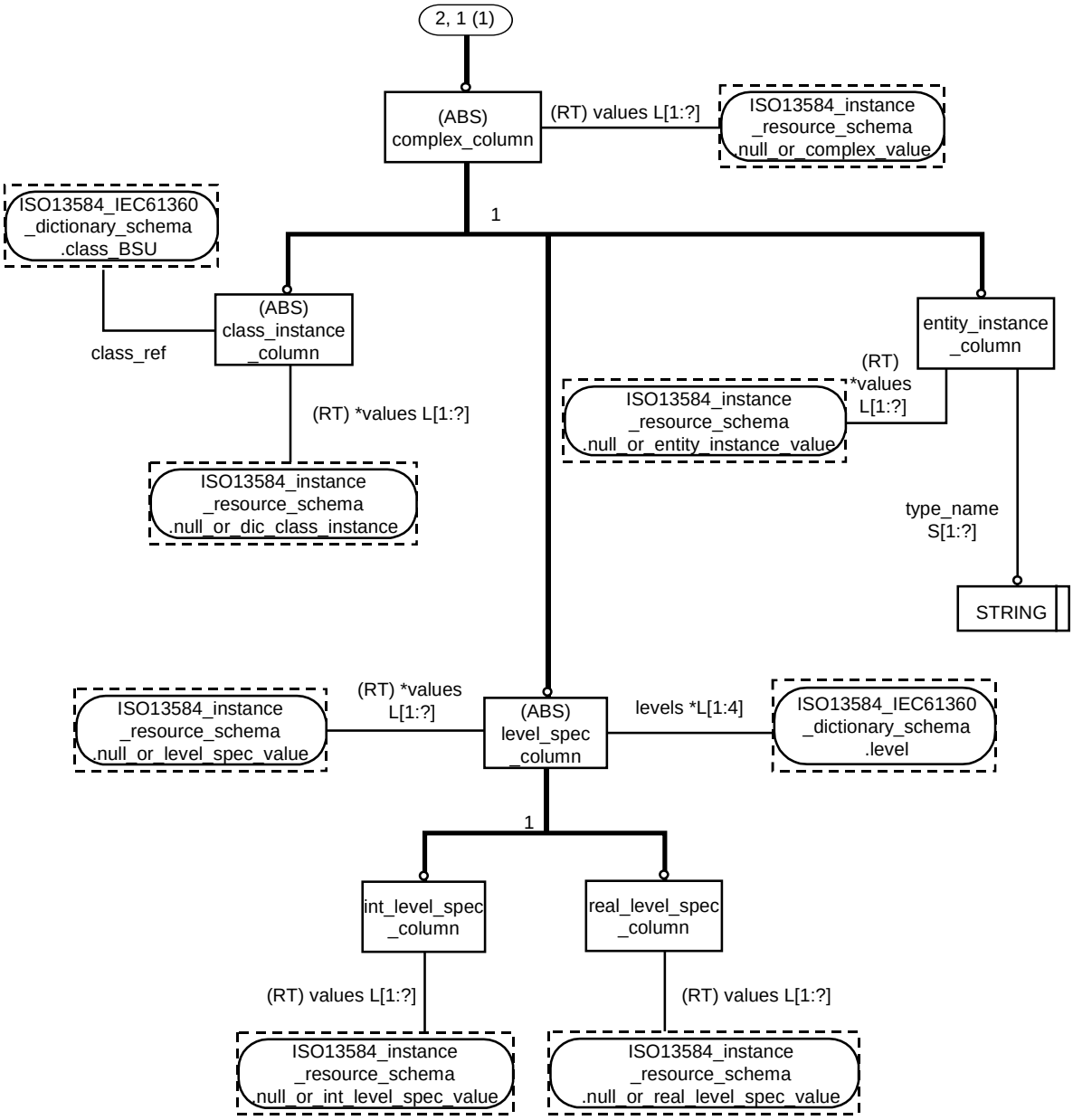


Figure R.8 — ISO13584_table_resource_schema diagram 2 of 4

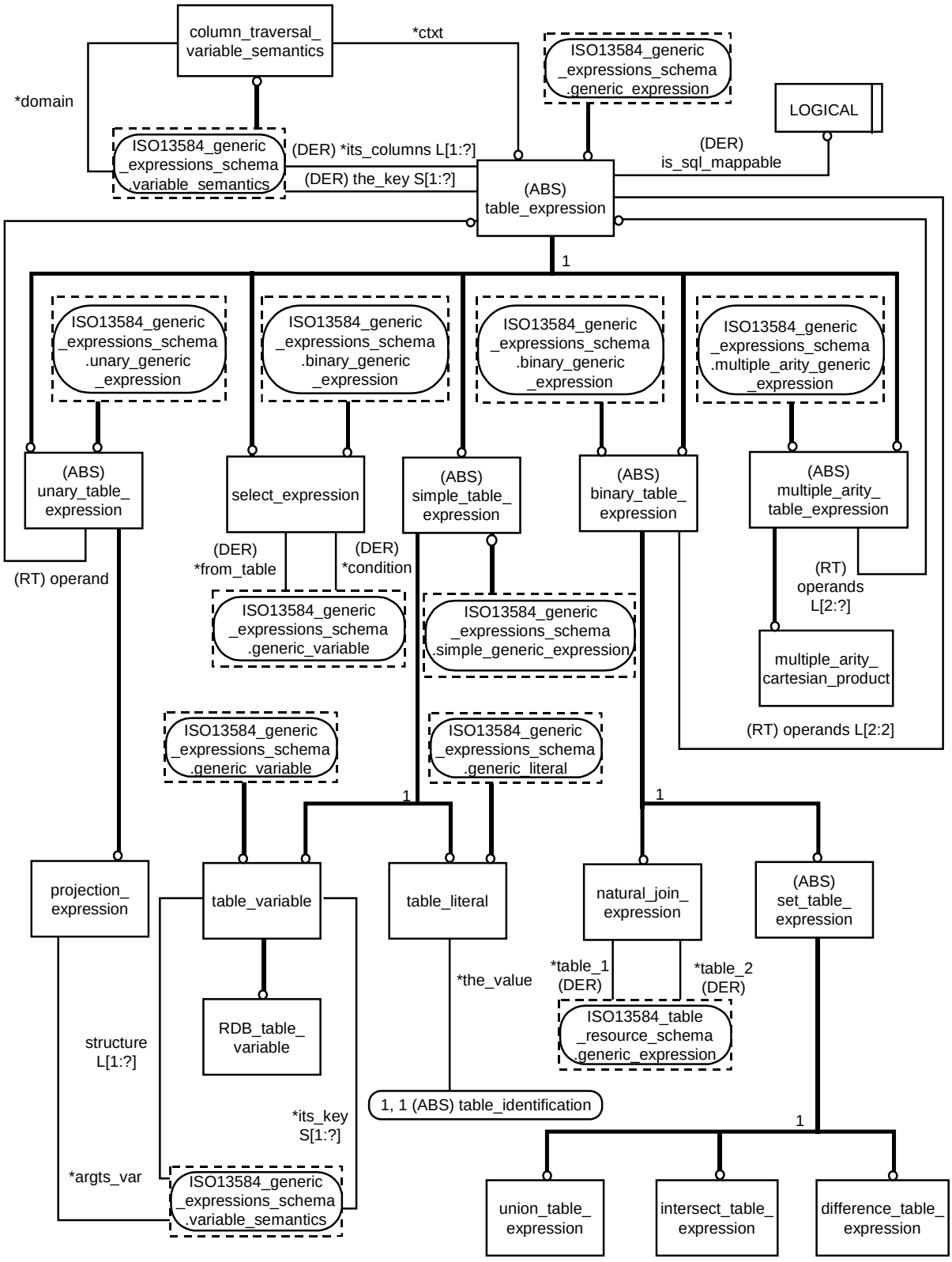


Figure R.9 — ISO13584_table_resource_schema diagram 3 of 4

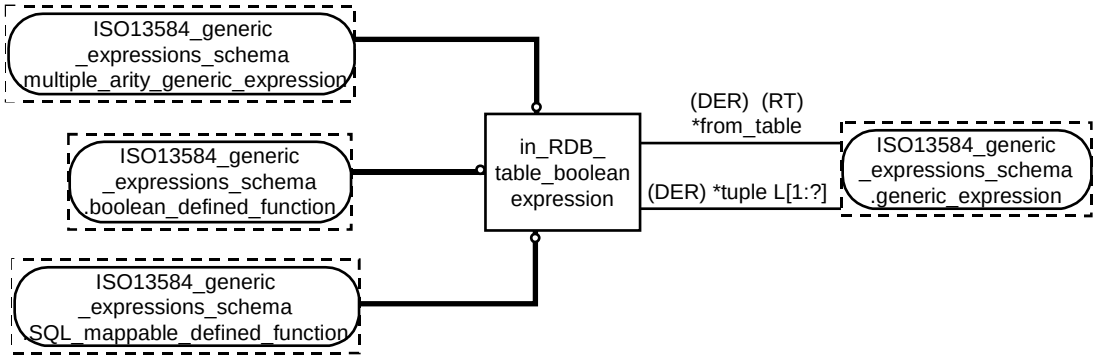


Figure R.10 — ISO13584_table_resource_schema diagram 4 of 4

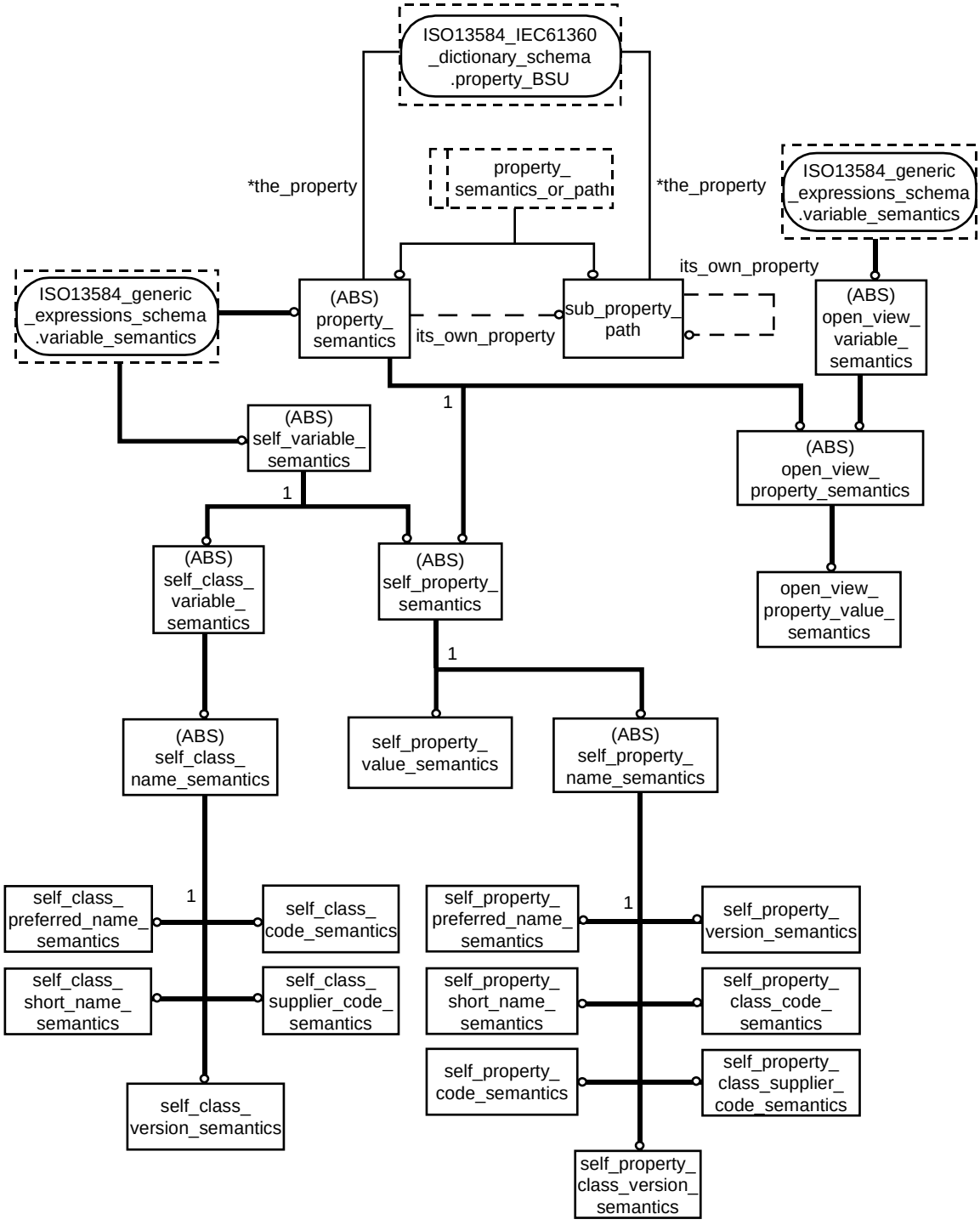


Figure R.11 — ISO13584_variable_semantics_schema diagram 1 of 1

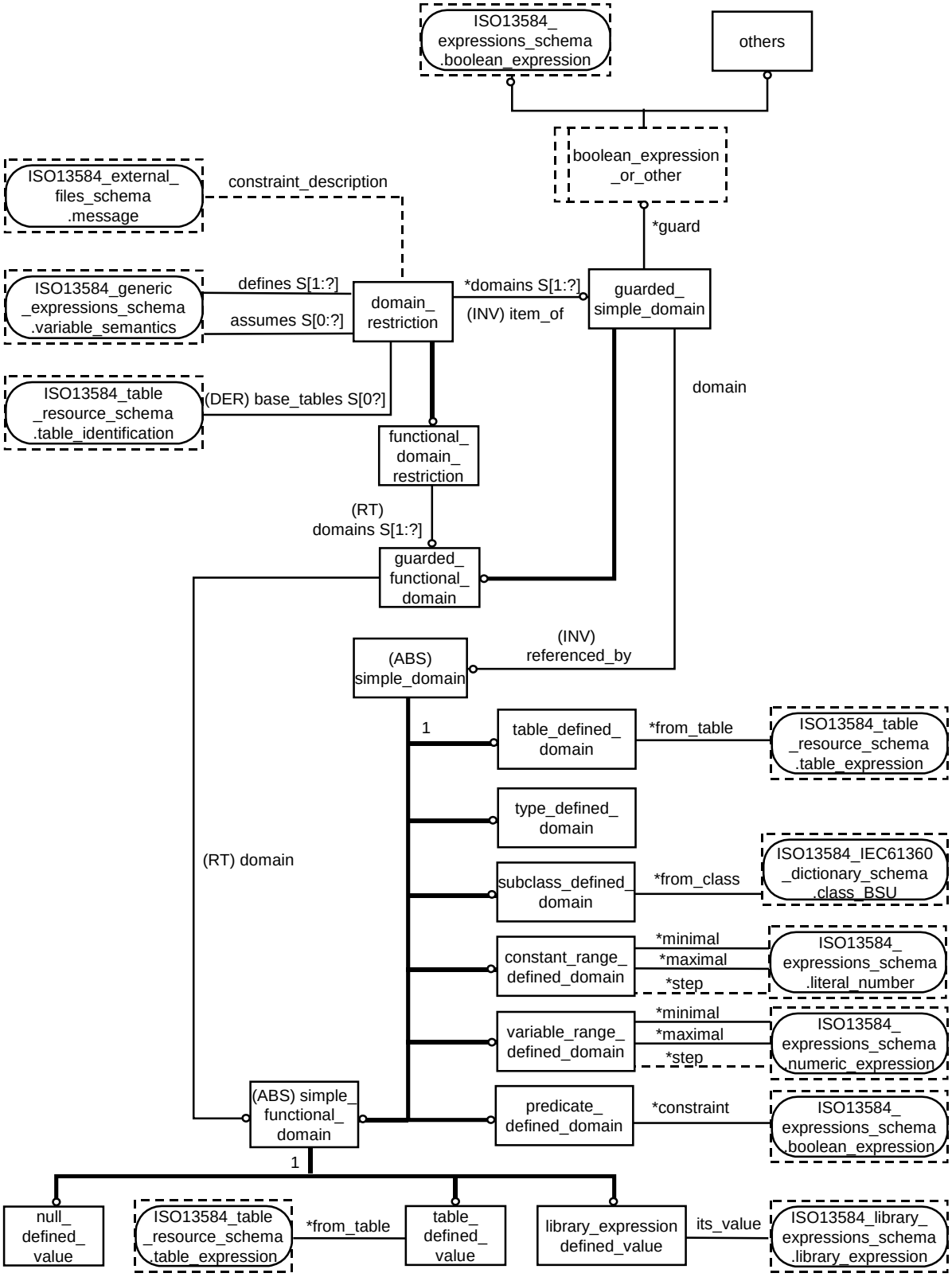


Figure R.12 — ISO13584_domain_resource_schema diagram 1 of 1

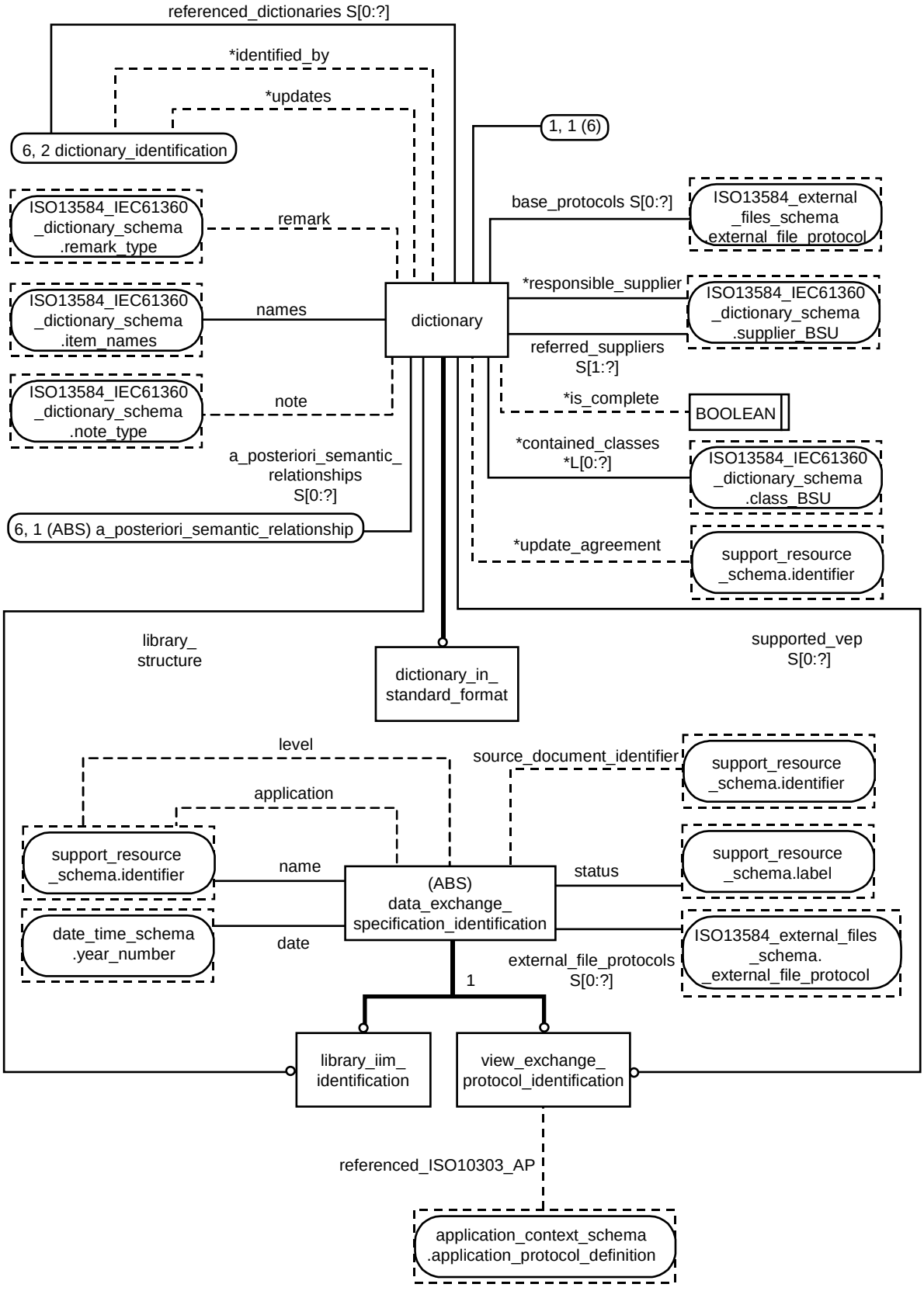


Figure R.13 — ISO13584_extended_dictionary_schema diagram 1 of 7

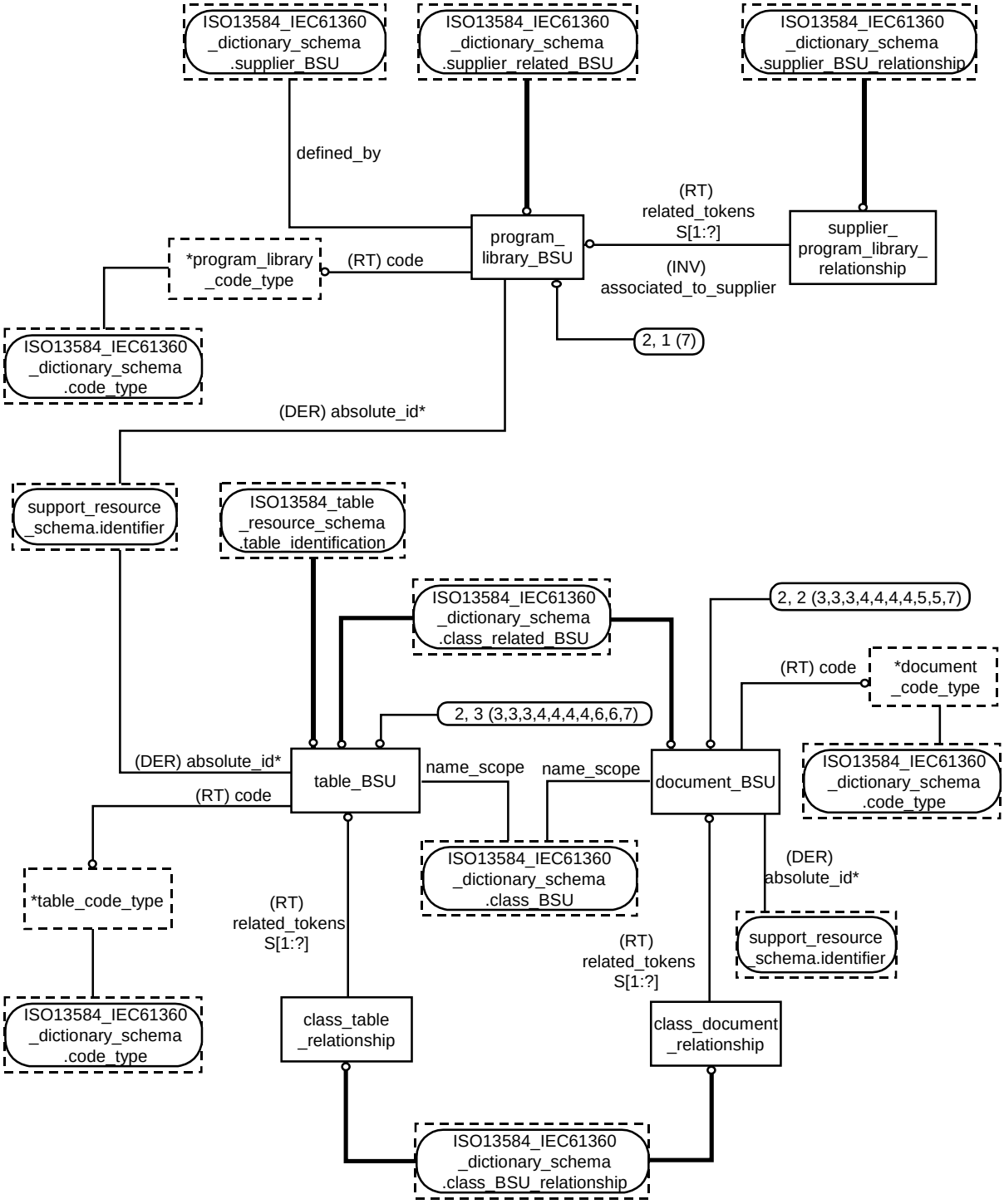


Figure R.14 — ISO13584_extended_dictionary_schema diagram 2 of 7

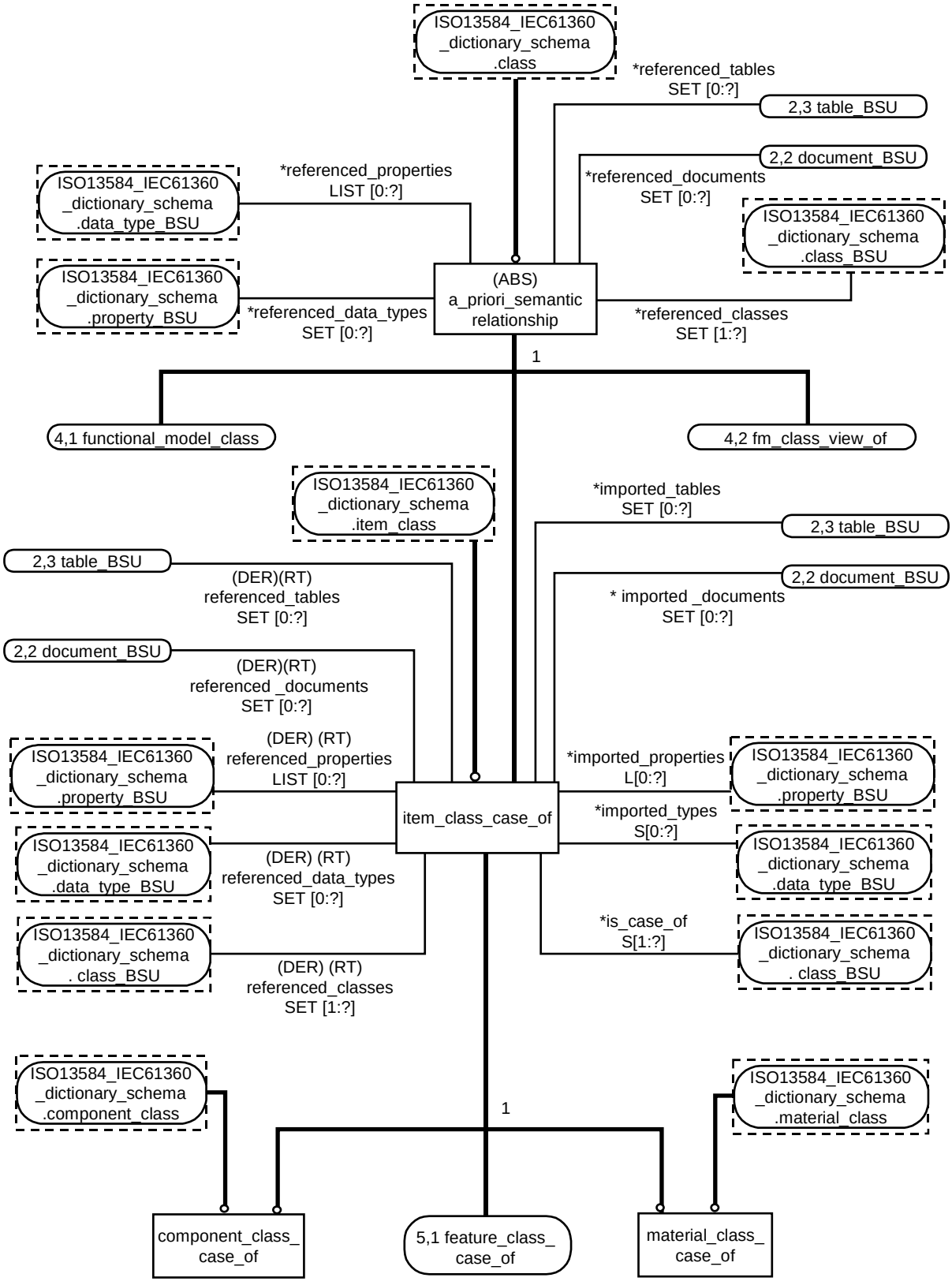


Figure R.15 — ISO13584_extended_dictionary_schema diagram 3 of 7

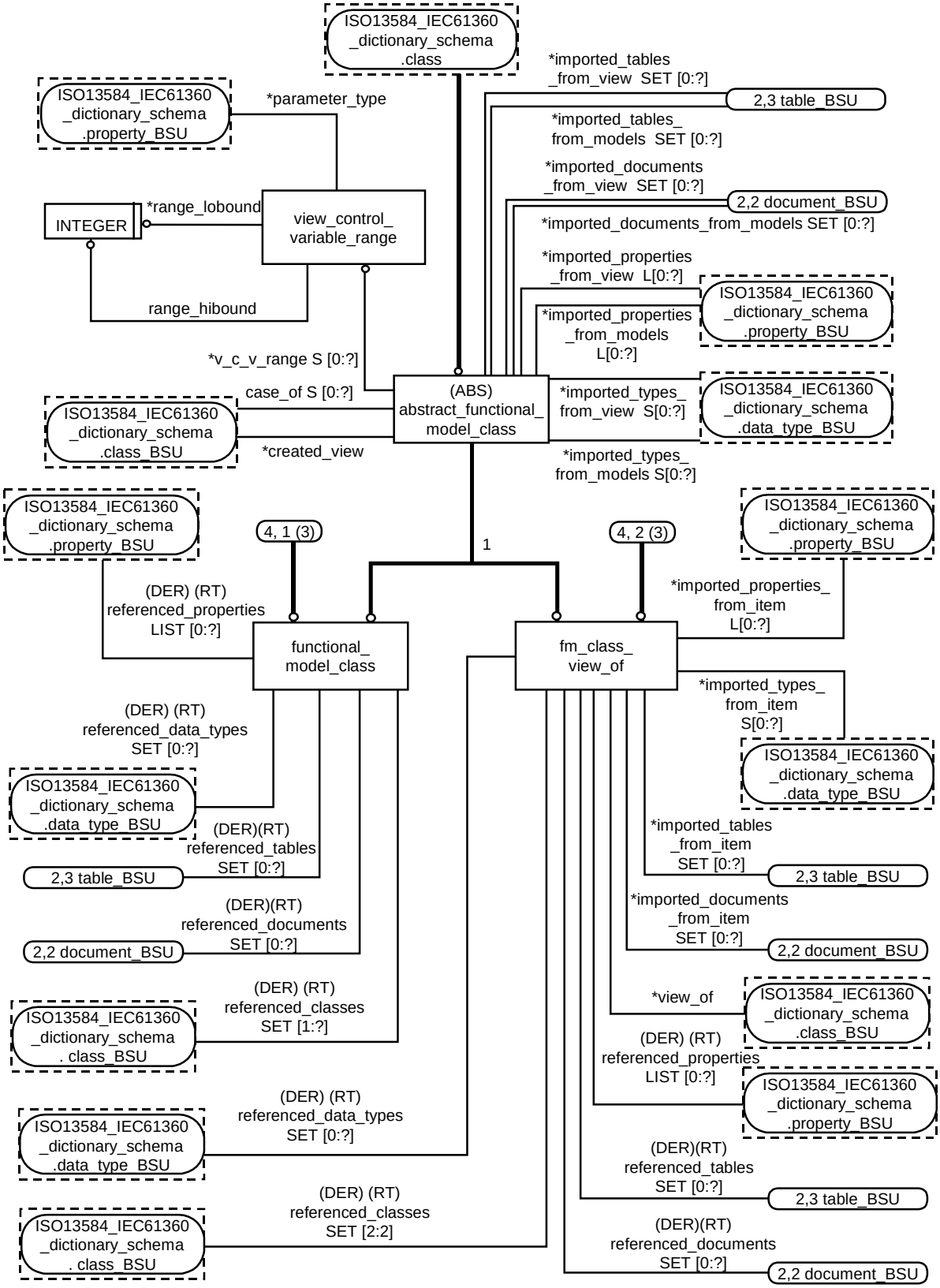


Figure R.16 — ISO13584_extended_dictionary_schema diagram 4 of 7

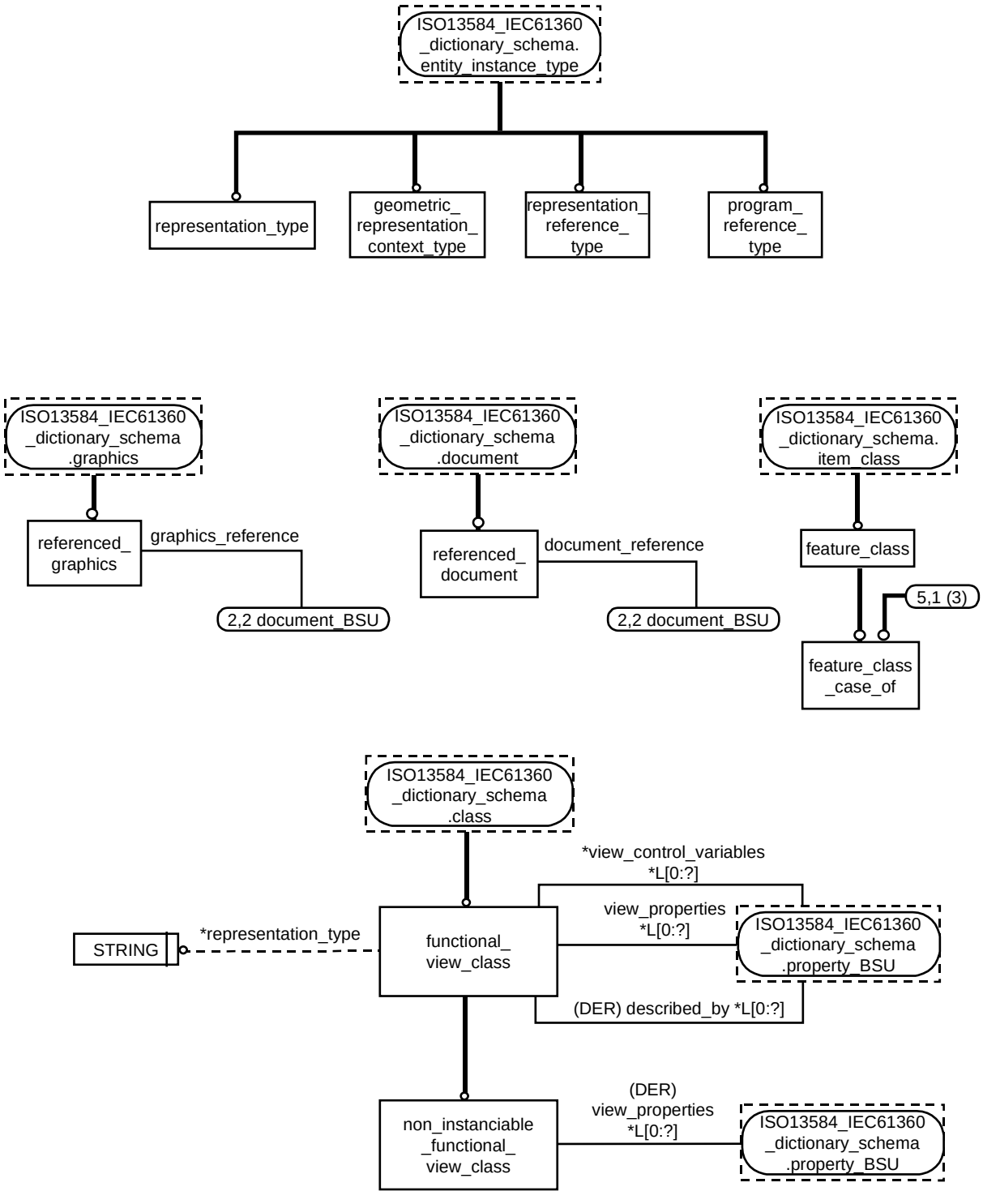


Figure R.17 — ISO13584_extended_dictionary_schema diagram 5 of 7

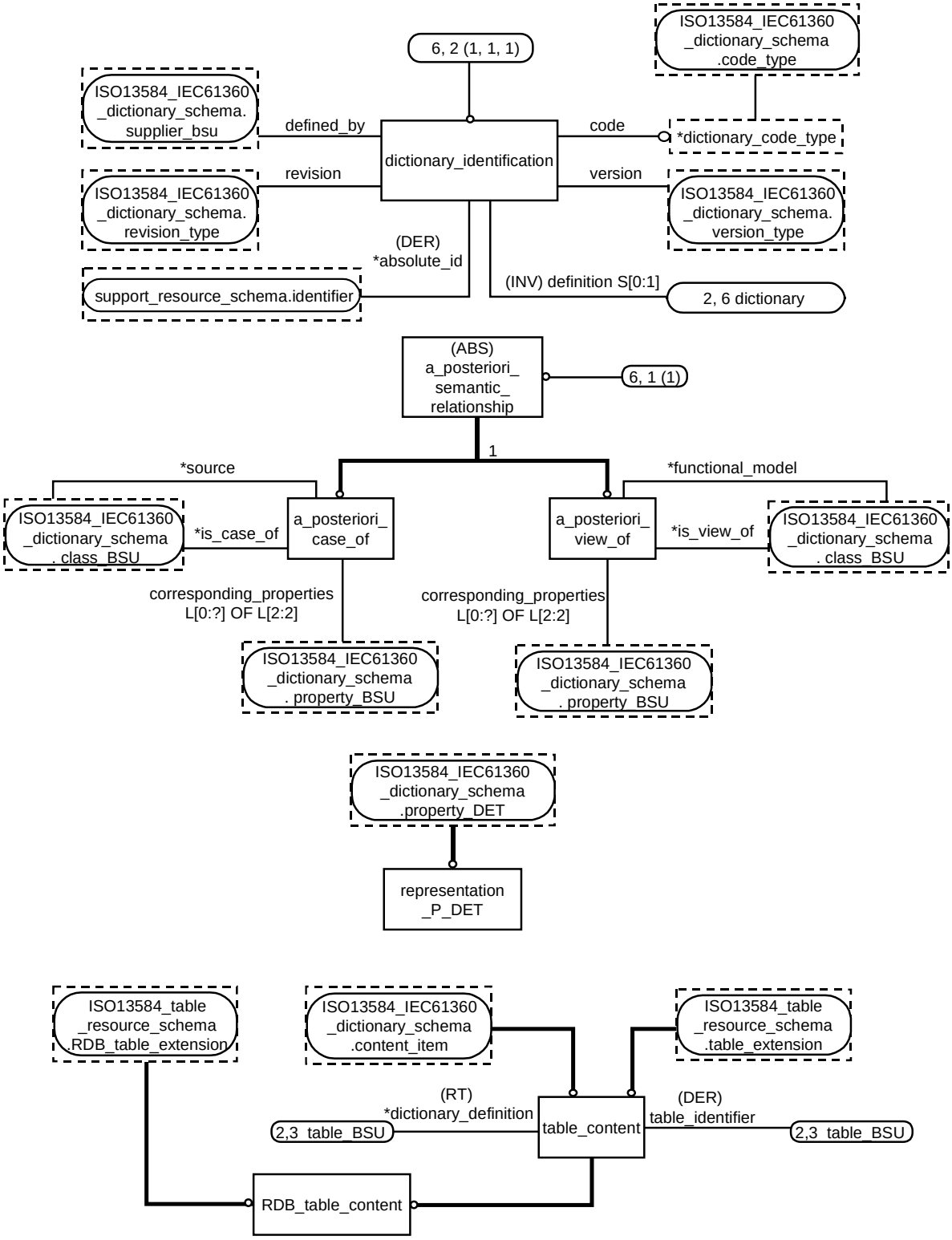


Figure R.18 — ISO13584_extended_dictionary_schema diagram 6 of 7

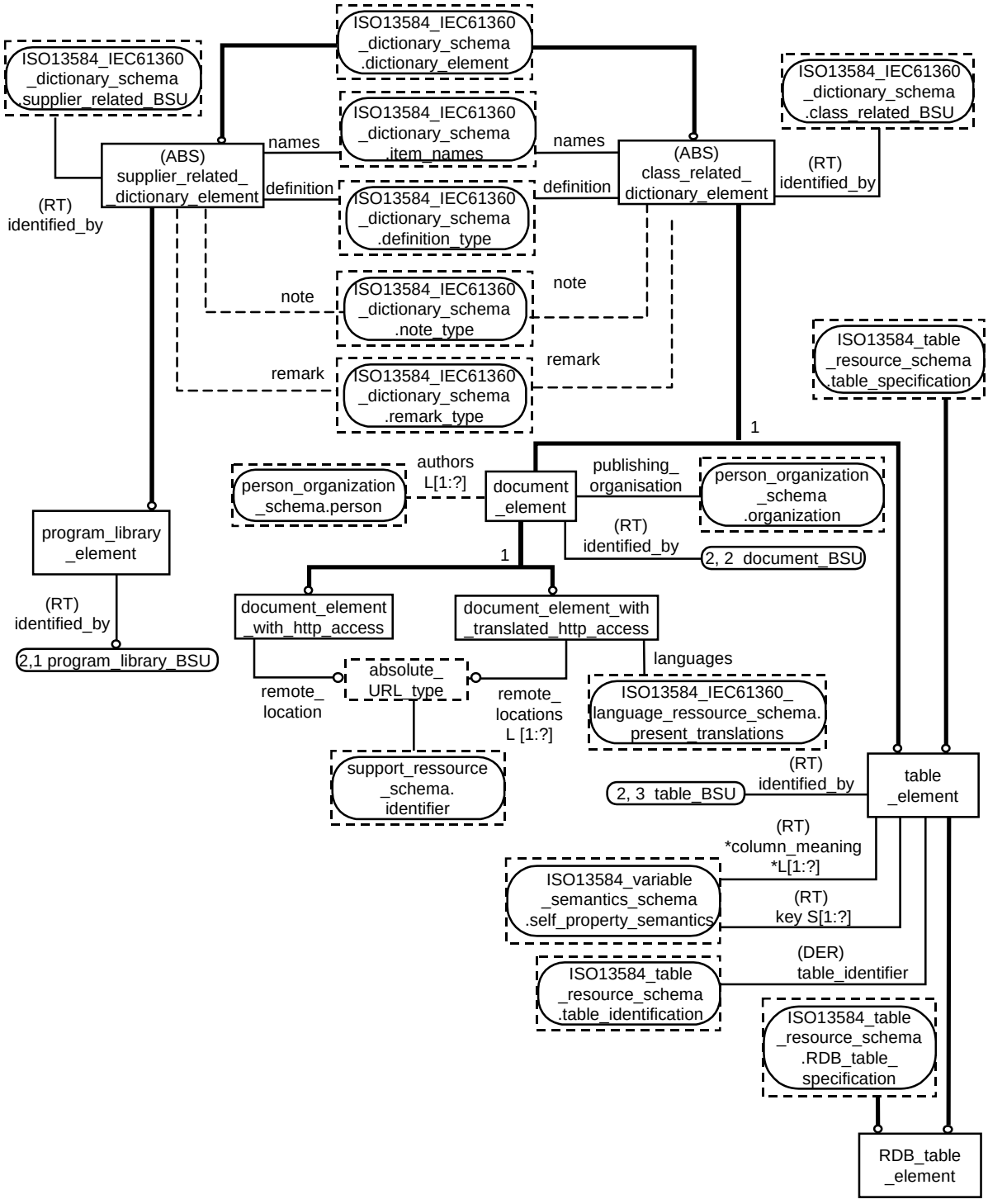


Figure R.19 — ISO13584_extended_dictionary_schema diagram 7 of 7

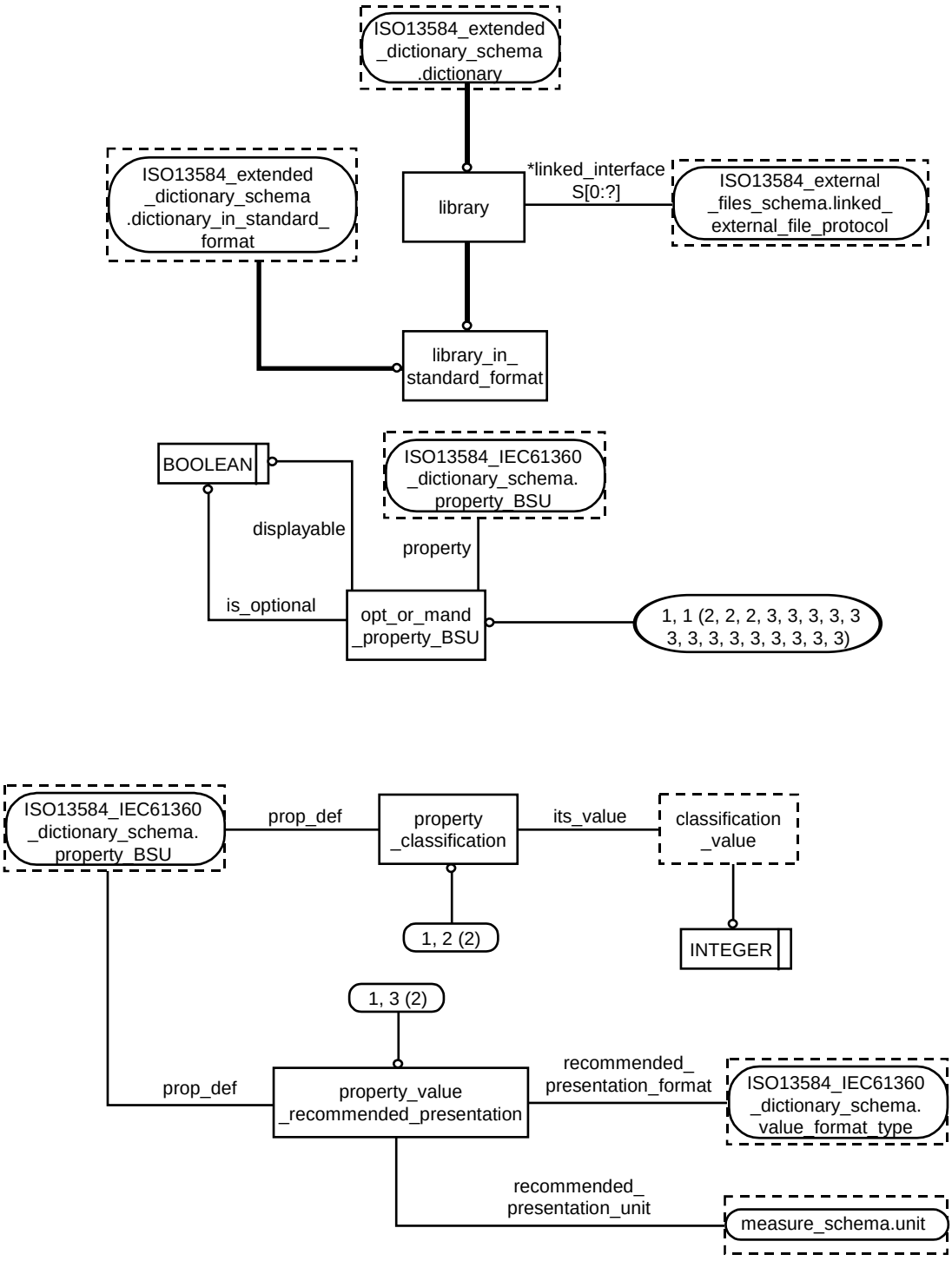


Figure R.20 — ISO13584_library_content_schema diagram 1 of 4

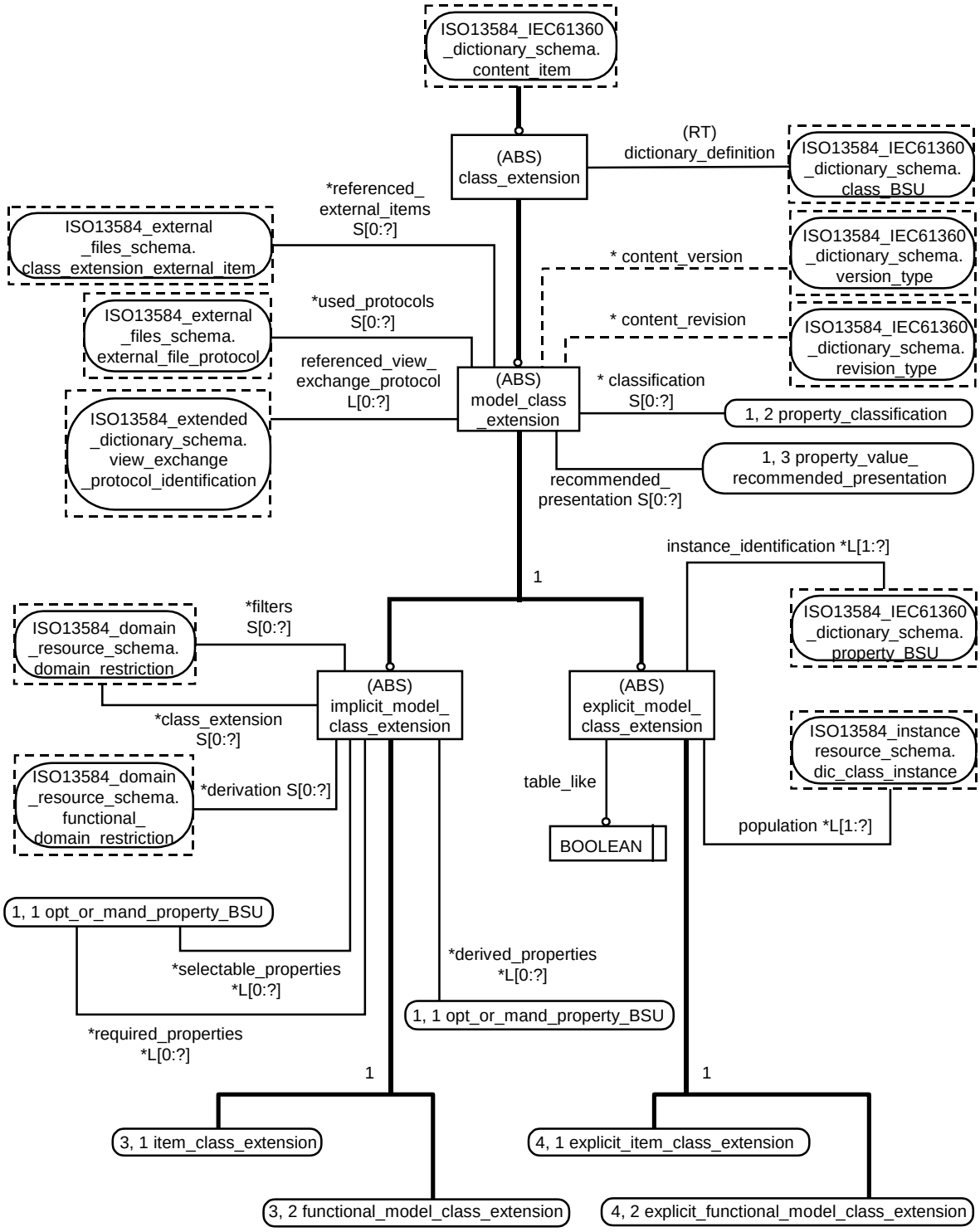
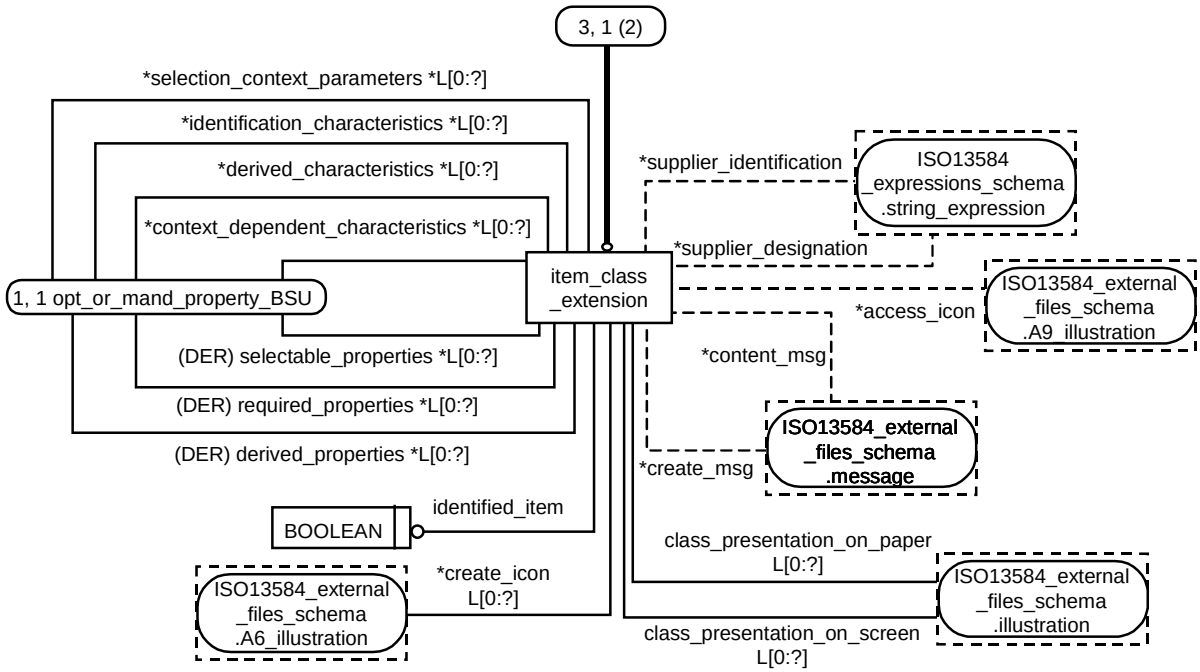


Figure R.21 — ISO13584_library_content_schema diagram 2 of 4



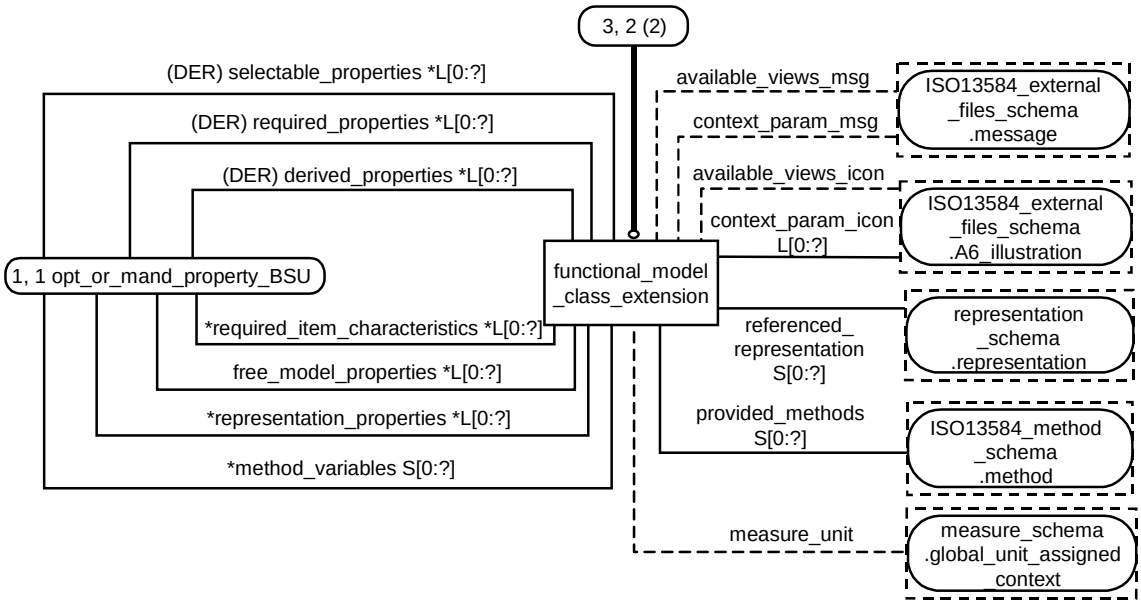


Figure R.22 — ISO13584_library_content_schema diagram 3 of 4

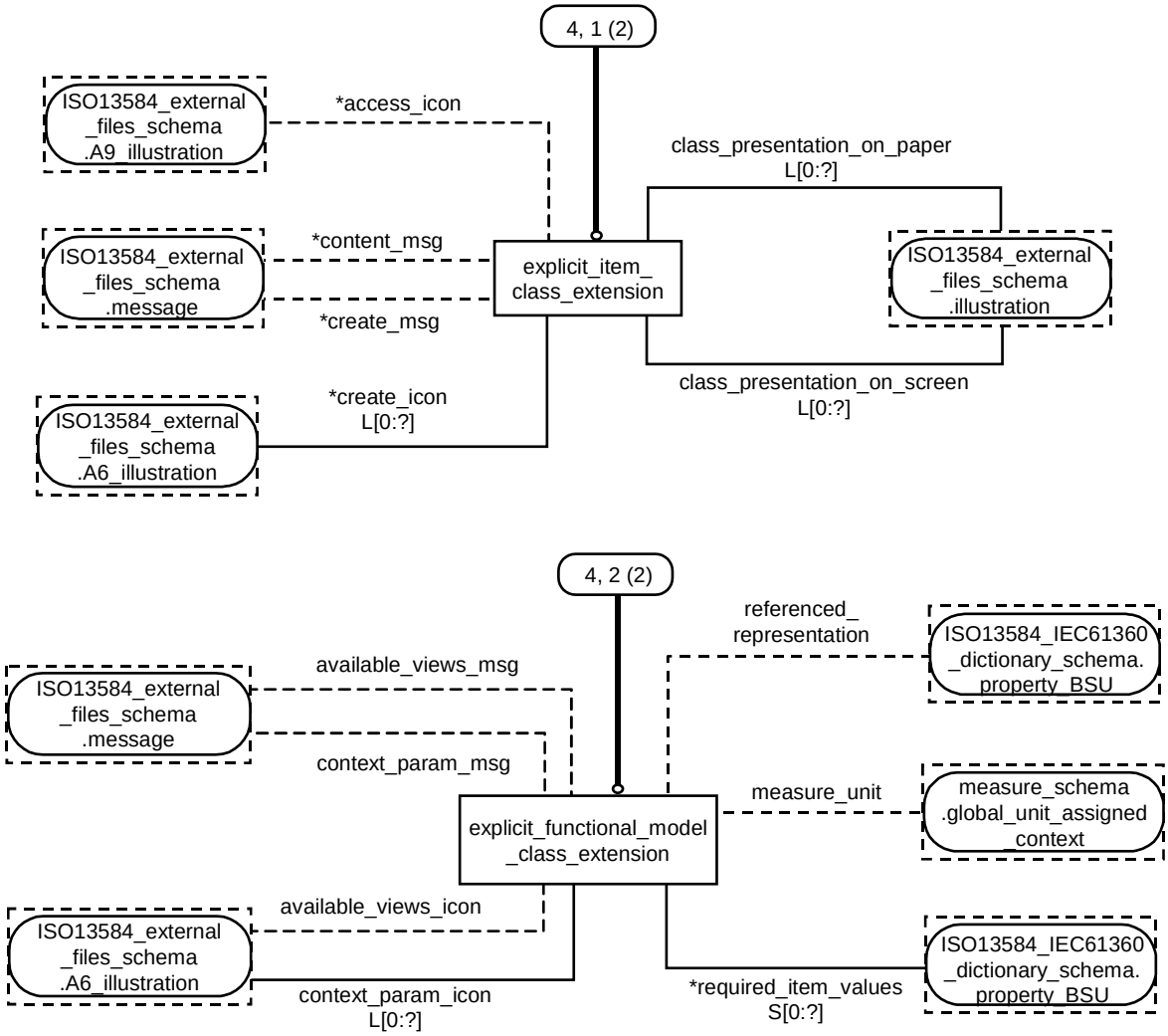


Figure R.23 — ISO13584_library_content_schema diagram 4 of 4

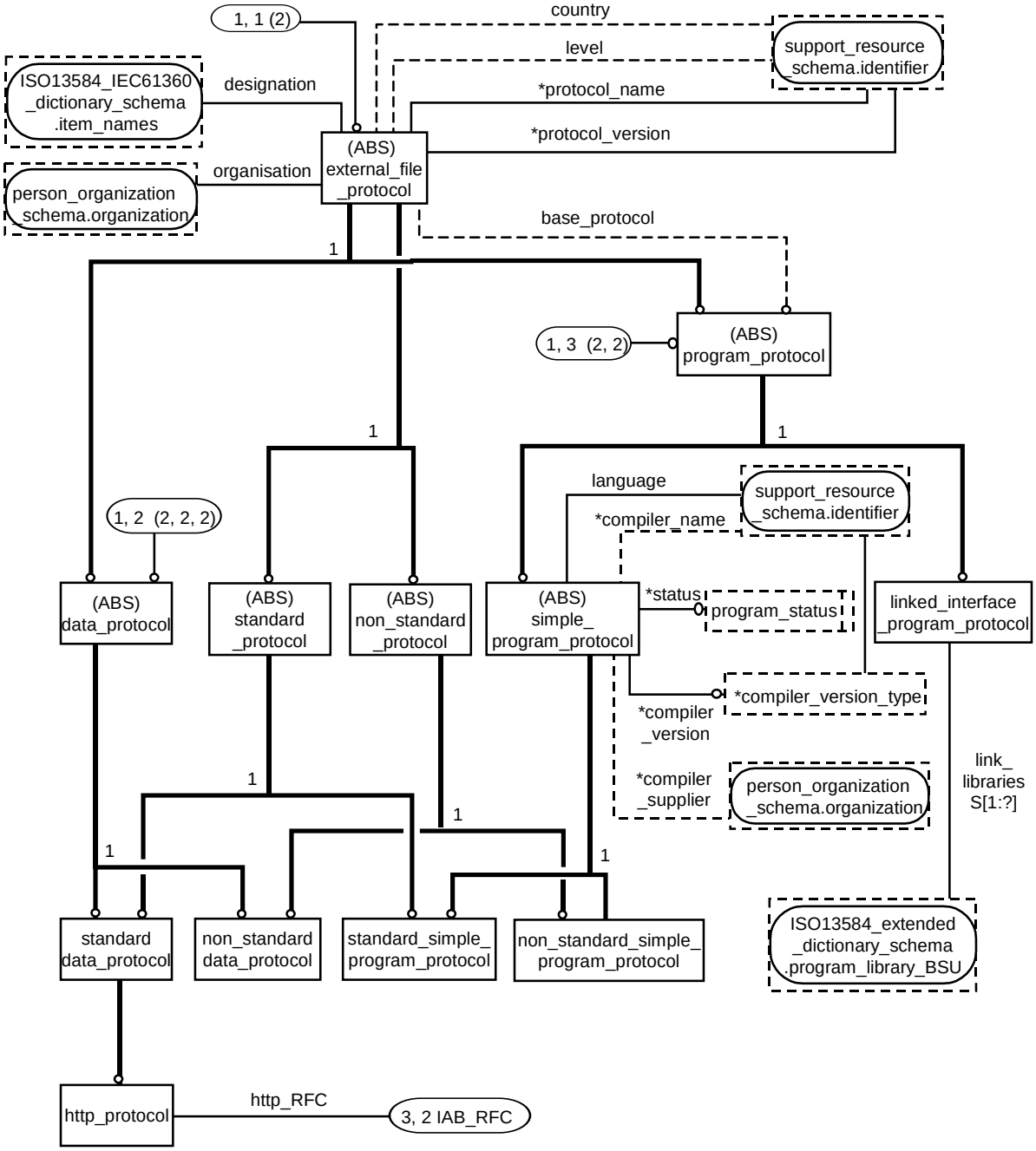


Figure R.24 — ISO13584_external_file_schema diagram 1 of 3

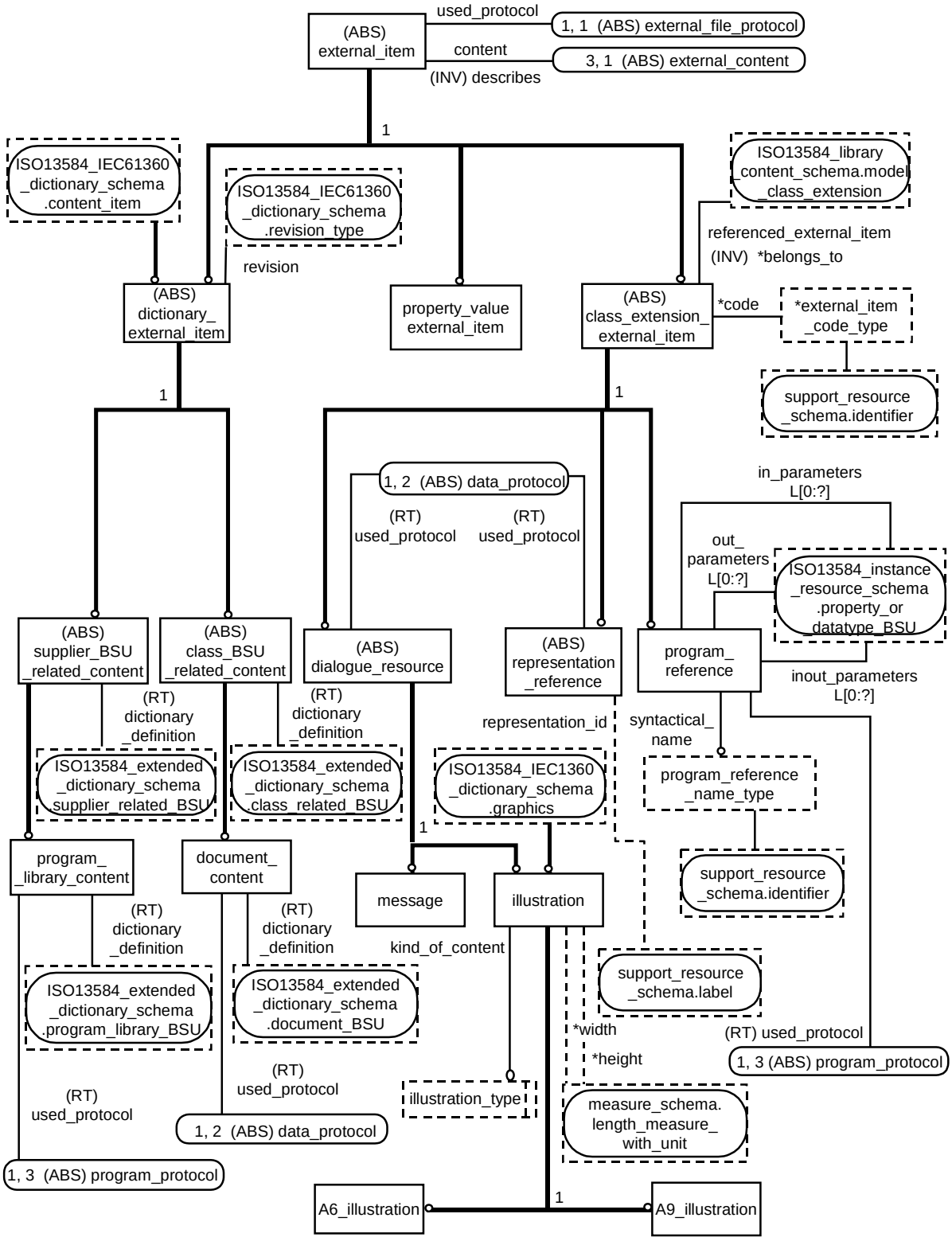


Figure R.25 — ISO13584_external_file_schema diagram 2 of 3

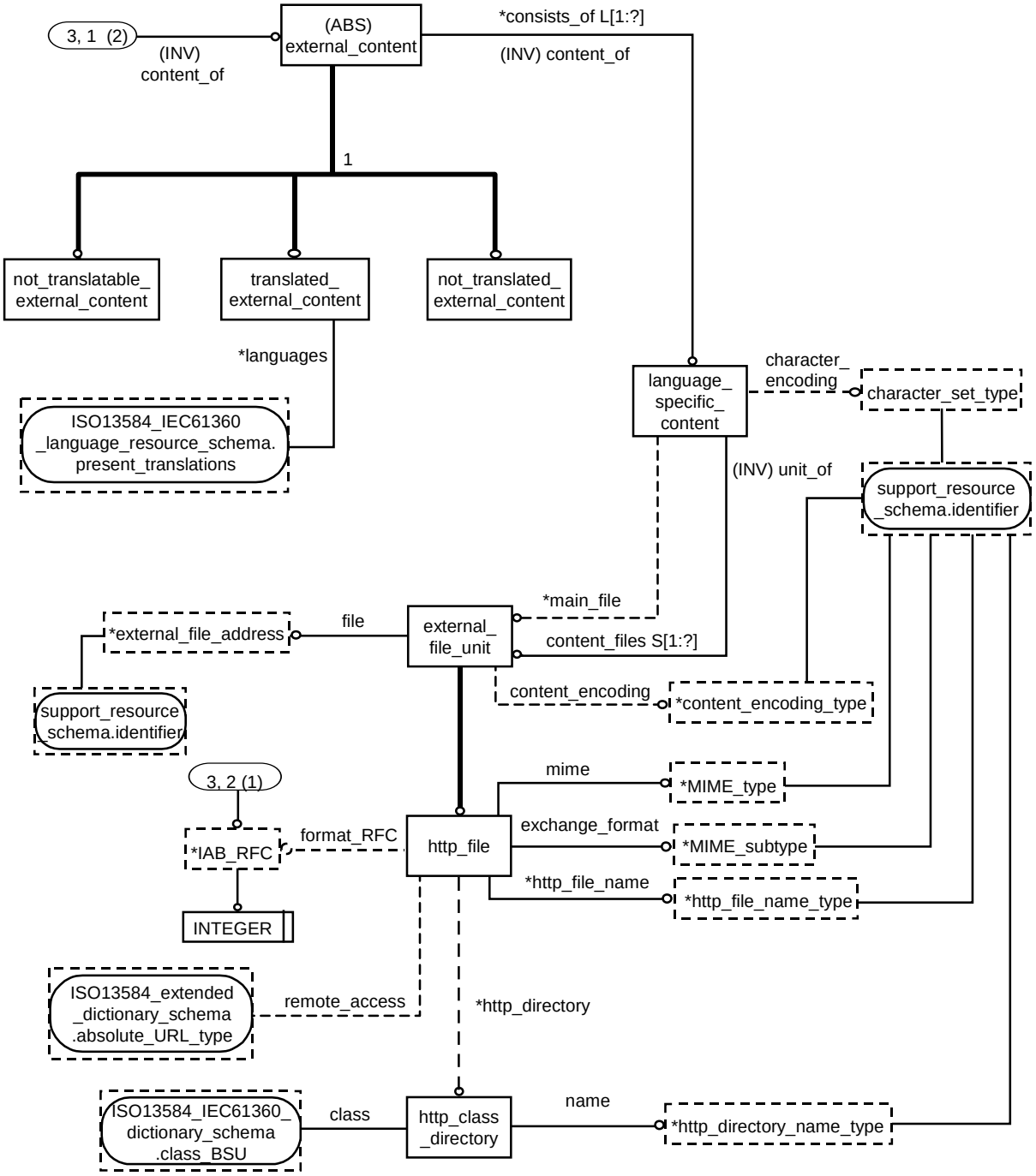


Figure R.26 — ISO13584_external_file_schema diagram 3 of 3

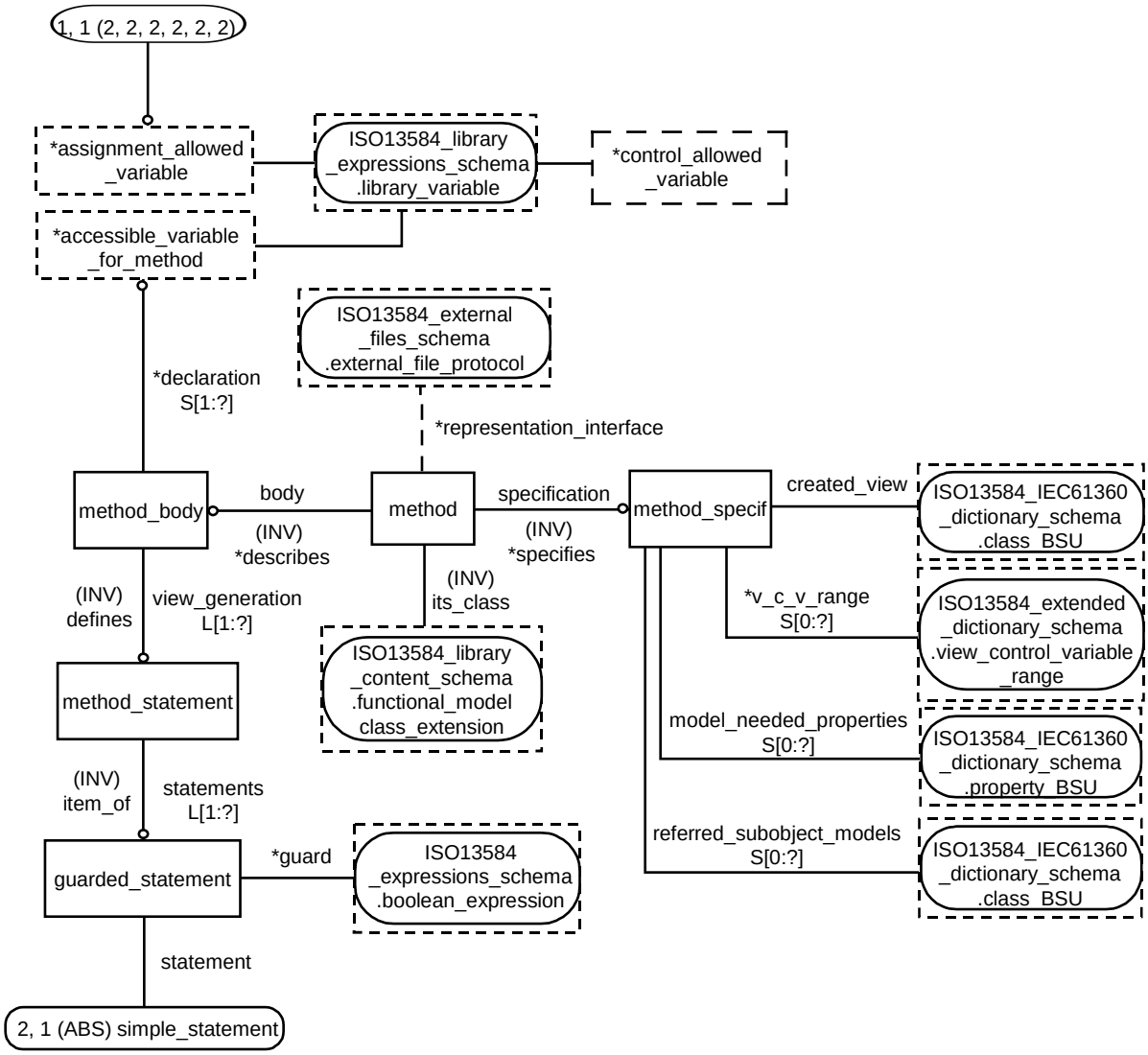


Figure R.27 — ISO13584_method_schema diagram 1 of 2

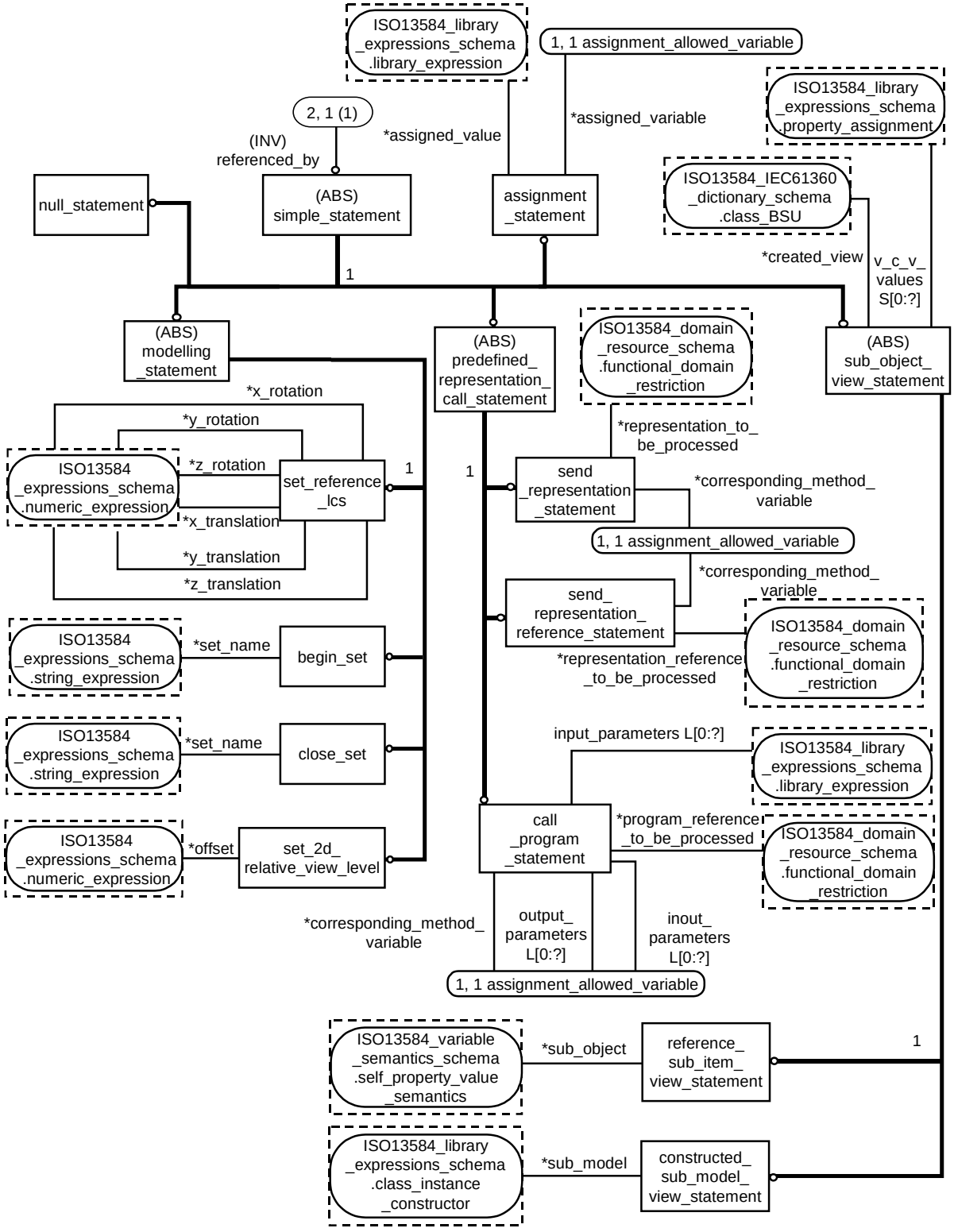


Figure R.28 — ISO13584_method_schema diagram 2 of 2

Annex S
(informative)

Notational Conventions and Generic Grammar for URL-encoded strings

This annex summarises the augmented Backus-Naur Form (BNF) notations used in the HTTP protocol, IAB RFC 2068, for syntax specification. It also uses these notations to outline the syntax of URL encoded strings as defined in IAB RFC 1808.

S.1 Augmented BNF [IAB RFC 2068]

The augmented BNF includes the following constructs.

name = definition
The name of a rule is simply the name itself (without any enclosing"<" and ">") and is separated from its definition by the equal "="character. White space is only significant in that indentation of continuation lines is used to indicate a rule definition that spans more than one line.

"literal"
Quotation marks surround literal text. Unless stated otherwise, the text is case-insensitive.

rule1 | rule2
Elements separated by a bar ("|") are alternatives. For instance, "yes | no" will accept yes or no.

(rule1 rule2)
Elements enclosed in parentheses are treated as a single element. Thus, "(elem (foo | bar) elem)" allows the token sequences "elem foo elem" and "elem bar elem".

***rule**
The character "*" preceding an element indicates repetition. The full form is "<n>*<m>element" indicating at least <n> and at most<m> occurrences of element. Default values are 0 and infinity so that "*(element)" allows any number, including zero; "1*element"requires at least one; and "1*2element" allows one or two.

[rule]
Square brackets enclose optional elements; "[foo bar]" is equivalent to "*1(foo bar)".

N rule
Specific repetition: "<n>(element)" is equivalent to"<n>*<n>(element)"; that is, exactly <n> occurrences of (element).

; comment
A semi-colon, set off some distance to the right of rule text, starts a comment that continues to the end of line. This is a simple way of including useful notes in parallel with the specifications.

S.2 URL-encoded string [IAB RFC 1808]

This is an augmented BNF description of the relative URL syntax. Note that this differs from the URL syntax defined in RFC 1738:1995 in that all schemas are required to use a single set of reserved characters and use them consistently within the major URL components.

The following rules define the allowed content of an URL as defined in IAB RFC 1808.

— URL = (absoluteURL | relativeURL) ["#" fragment]

— absoluteURL = generic-RL | (scheme ":" *(uchar | reserved))

— generic-RL = scheme ":" relativeURL

— relativeURL = net_path | abs_path | rel_path

— net_path = "//" net_loc [abs_path]

— abs_path = "/" rel_path

— rel_path = [path] [";" params] ["?" query]

— path = fsegment *("/" segment)

— fsegment = 1*pchar

— segment = *pchar

— params = param *(";" param)

— param = *(pchar | "/")

— scheme = 1*(alpha | digit | "+" | "-" | ".")

— net_loc = *(pchar | ";" | "?")

— query = *(uchar | reserved)

— fragment = *(uchar | reserved)

— pchar = uchar | ":" | "@" | "&" | "="

— uchar = unreserved | escape

— unreserved = alpha | digit | safe | extra

— escape = "%" hex hex

— hex = digit | "A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" | "d" | "e" | "f"

— alpha = lowalpha | hialpha

— lowalpha = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"

— hialpha = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"

— digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

— safe = "$" | "-" | "_" | "." | "+"

— extra = "!" | "*" | "'" | "(" | ")" | ","

— national = "{" | "}" | "|" | "\" | "^" | "~" | "[" | "]" | "`"

— reserved = ";" | "/" | "?" | ":" | "@" | "&" | "="

— punctuation = "<" | ">" | "#" | "%" | <">

Bibliography

[1] ELU, P., PIERRA, G., SARDET, E., KASAN, D., NEUMAIER, A., *Design of PLib-Compliant intelligent electronic catalogues - Supplier Oriented Methodology* - Deliverable #13/22 of the European ESPRIT project #8984, PLUS, December 1996., (available on the Internet, http://www.lisi.ensma.fr /pub/PLUS/D13_22_PLUS_PLIB_METHOD/ D13_22_PLUS_PLIB_METHOD.<doc|ps|rtf|txt>)

[2] IAB RFC 1738:1994, *Internet architecture board proposed standard protocol: Uniform Resource Locators (URL)*

[3] IAB RFC 2046:1996, *Internet architecture board draft standard protocol: MIME Media Types (MIME-MEDIA)*

[4] IAB RFC 2047:1996, *Internet architecture board draft standard protocol: MIME Message Header Extensions for Non-ASCII Text (MIME-MSG)*

[5] IAB RFC 2048:1996, *Internet Architecture Board Draft Standard Protocol: MIME Registration Procedures* (available on the Internet Official Internet Server at the following URL: http//DS.INTERNIC.NET)

[6] IAB RFC 2376:1998, *Internet Architecture Board Informational Protocol: XML Media Types (available* on the Internet Official Internet Server at the following URL: http//DS.INTERNIC.NET)

[7] ISO 10303-22:1998, *Industrial automation systems and integration — Product data representation and exchange — Part 22: Implementation methods: Standard data access interface*

Index

ICS 25.040.40

Price based on 653 pages