INTERNATIONAL STANDARD ISO 13584-42:1998
TECHNICAL CORRIGENDUM 1

Published 2003-01-15

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION

Industrial automation systems and integration — Parts library —

Part 42: Description methodology: Methodology for structuring part families

TECHNICAL CORRIGENDUM 1

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

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

RECTIFICATIF TECHNIQUE 1

Technical Corrigendum 1 to ISO 13584-42:1998 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 4, *Industrial data.*

Introduction

This document corrects ISO 13584-42:1998; *Industrial automation systems and integration – Parts Library – Part 42: Methodology for structuring part families*.

This corrigendum addresses a number of issues regarding inaccuracies and areas of ambiguity within this standard.

ICS 25.040.40 **Ref. No. ISO 13584-42:1998/Cor.1:2003(E)**

Modifications to ISO 13584-42:1998

Clause 2, p. 2

Add the following normative reference:

ISO 639:1988 *Code for the representation of names of languages.*

Clause 3, p. 3

Add the following new definition:

3.4.8
item
a thing that can be captured by a class structure and a set of properties

Clause 3.4.4, p. 5

Change the data element type definition as follows:

unit of data for which the identification, description and value representation have been specified

Clause 7.2.21, p. 20

Change the Descr: clause as follows to make clear that property version numbers shall be assigned in ascending order but not necessarily in sequential order :

Descr: A string that contains a natural number to indicate the different versions of a property during the life cycle. Version numbers shall be issued in ascending order. A new version of the property shall be generated according to the definitions given in Table 1 (see 7.3).

Clause 8.2.20, p. 20

Change the Descr: clause as follows to make clear that class version numbers shall be assigned in ascending order but not necessarily in sequential order and to correct the size that should not appear in the defintion and which has be changed (see Clause D.3.2, p. 42):

Descr: A string that contains a natural number to indicate the different versions of a family during the life cycle. Version numbers shall be issued in ascending order. A new version number of a family shall be generated according to the definitions given in table 3 (see 8.3).

Table 2, p. 23

Change a reference to footnote as follows:

property_DET.domain\non_quantitative_int_type.domain.its_values[13]

Table 3, p. 32

Change all the references to footnotes as follows:

Attribute	Add	Modify	Delete
Code	–	X	X
Superclass	V[15]	V[15]	V[15]
Preferred Name	–	R	X
Short Name	–	R	X
Visible Types	V	V[16]	X
Applicable Types	V	V[16]	X
Sub-Class Selection Properties	V	V[16]	X
Synonymous name	R	R	R
Visible Properties	V	V[16]	X
Applicable Properties	V	V[16]	X
Class Value Assignment	V	V[16]	X
Definition	–	R[17]	X
Source Document of Definition	R	R	R
Note	R	R	R
Remark	R	R	R
Simplified Drawing	–	R	R
Date of Original Definition	–	X	X
Date of current version	–	V	X
Date of Current Revision	–	R / V	X
Version number	–	V	X
Revision number	–	R	X

Clause D.3.2, p. 42

Following Charleston's discussions, some constant values haves been redefined. Change it as follows:

```
CONSTANT
...
      supplier_code_len: INTEGER := 70;
...
      pref_name_len: INTEGER := 70;
...
      short_name_len: INTEGER := 30 ;
END_CONSTANT;
```

Following discussions about updating process, it has been decided to increase the size of the version string from 3 digits to 9 digits.

```
CONSTANT
...
      version_len: INTEGER := 9;
...
END_CONSTANT;
```

Clause D.3.5.1.3, p. 54

Where rules 4 and 5 contents have been changed as follows:

```
ENTITY class
  ABSTRACT SUPERTYPE
  SUBTYPE OF (class_and_property_elements);
  ...
  WHERE
  ...
   wr4 : check_properties_applicability(SELF);
   wr5 : check_datatypes_applicability(SELF);
 END_ENTITY;
```

Then, two new functions have been introduced in clauses D.3.9.17 and D.3.9.18.

Clause D.3.6.1, p. 57

In where rule one, the unnecessary qualified attribute shall be removed as follows:

```
  WR1: QUERY ( c <* describes_classes |
      NOT ( is_subclass (c, name_scope.definition[1])))= [];
```

*A consequence of the modification done in **clause D.3.9.6**, the where rule 1 formal propositon of the **property_bsu** entity shall be modified as follows:*

WR1: any class referenced by the **describes_classes** attribute of a **property_BSU** either is the class referenced by its **name_scope** attribute, or it is a subclass of this class.

Clause D.3.6.2, p. 58

Update the synonymous_symbol declaration with:

```
ENTITY property_DET
ABSTRACT SUPERTYPE OF(ONEOF(
      condition_DET, dependent_P_DET, non_dependent_P_DET))
SUBTYPE OF(class_and_property_elements);
...
      synonymous_symbols: SET [0:?] OF mathematical_string;
...
DERIVE
      describes_classes: SET [0:?] OF class
            := identified_by.describes_classes;
END_ENTITY; -- property_DET
```

Clause D.3.6.3.2, p. 61

*In **dependent_P_DET** change in WR1 " (p.definition)" by "(p.definition[1])". Thus the new WR1 is as follows:*

```
ENTITY dependent_P_DET
SUBTYPE OF(property_DET);
      depends_on: SET [1:?] OF property_BSU;
WHERE
      WR1: QUERY(p <* depends_on | NOT(definition_available_implies(
            p, ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CONDITION_DET'
            IN TYPEOF(p.definition[1])))))  = [];
END_ENTITY; -- dependent_P_DET
```

Clause D.3.7.1.1, p. 64

defining_class *inverse attribute specification changed.*

```
ENTITY data_type_BSU
...
INVERSE
      defining_class: SET OF class FOR defined_types;
...
END_ENTITY;
```

Clause D.3.7.2.18, p. 71

*The **entity_instance_type** shall not be declared as ABSTRACT SUPERTYPE because it should be allowed to instanciate it directly using the dictionary schema. Change its EXPRESS specification as follows:*

```
ENTITY entity_instance_type
  SUBTYPE OF (complex_type);
  type_name : SET OF STRING;
END_ENTITY;
```

Clause D.3.7.3.1, p. 74

*In where rule 1, replace 'XOR' with 'OR'. Add where rule 3 for specifying if no language is specified, no translation can be provided for **dic_value**s, and change size of its_values from* [2:?] *to* [1:?] *:*

```
ENTITY value_domain;
      its_values: LIST [1:?] OF dic_value;
      source_doc_of_value_domain: OPTIONAL document;
      languages: OPTIONAL present_translations;
      terms: LIST [0:?] OF item_names;
WHERE
      WR1: NOT EXISTS(languages) OR (QUERY(v <* its_values |
            languages :<>: v.meaning.languages) = []);
      WR2: codes_are_unique(its_values);
      WR3: EXISTS(languages) OR (QUERY(v <* its_values |
            EXISTS(v.meaning.languages)) = []);
END_ENTITY;
```

And add the following formal proposition:
WR3: if no **languages** is provided, the value meanings shall not be assigned any language.

Clause D.3.8.1.2, p. 78

*Because a **code_type** is used to build an unambiguous absolute identifier, it shall not be an empty string. As a new where rule to the type definition as follows:*

```
TYPE code_type = identifier;
WHERE
...
      WR4: NOT(SELF = '');
END_TYPE; -- code_type
```

And add the following formal proposition:
WR4: a **code_type** shall not be an empty string.

Clause D.3.8.1.18, p. 83

Change the version_type specification as follows:

```
*)
TYPE version_type = code_type;
WHERE
      WR1: LENGTH(SELF) <= version_len;
      WR2: EXISTS(VALUE(SELF)) AND ('INTEGER' IN TYPEOF(VALUE(SELF)))
            AND (VALUE(SELF) >= 0);
END_TYPE; -- version_type
(*
```

Formal propositions:
WR1: the length of a **version_type** value shall be lower than the **version_len** (i.e., 9).
WR2: the **version_type** shall contain digits only.

Clause D.3.8.2.5, p. 84

Change the short_name attribute as being optional. Update WR1 consequently.
In where rule 1, replace 'XOR' with 'OR'.
In where rule 2, replace 'XOR' with 'OR'.
Remove where rule 3.
*Specify a new where rule 3 stipulating that if no **languages** are provided, **preferred_name**, **short_name** and **synonymous_names** shall not be translated.*

```
ENTITY item_names;
      preferred_name: pref_name_type;
      synonymous_names: SET OF syn_name_type;
      short_name: OPTIONAL short_name_type;
      languages: OPTIONAL present_translations;
      icon : OPTIONAL graphics;
WHERE
         WR1: NOT (EXISTS(languages )) OR  (
              ('ISO13584_IEC61360_LANGUAGE_RESOURCE_SCHEMA'
              + '.TRANSLATED_LABEL' IN TYPEOF(preferred_name) )
              AND (languages :=: preferred_name\translated_label.languages)
              AND (NOT (EXISTS(short_name) )
              OR ('ISO13584_IEC61360_LANGUAGE_RESOURCE_SCHEMA'
              + '.TRANSLATED_LABEL' IN TYPEOF(short_name) )
              AND (languages :=: short_name\translated_label.languages ) )
              AND (QUERY     (s <* synonymous_names
                   | NOT ('ISO13584_IEC61360_DICTIONARY_SCHEMA'
                   + '.LABEL_WITH_LANGUAGE' IN TYPEOF(s) ) ) = [ ] ) );
         WR2: NOT EXISTS(languages) OR (QUERY(s <* synonymous_names |
              EXISTS(s.language) AND NOT(s.language IN
              QUERY(l <* languages.language_codes | TRUE
              ))) = []);
         WR3: EXISTS(languages) OR
              (('SUPPORT_RESOURCE_SCHEMA.LABEL' IN
              TYPEOF(preferred_name))
              AND (NOT(EXISTS(short_name)) OR
              ('SUPPORT_RESOURCE_SCHEMA.LABEL' IN
              TYPEOF(short_name)))
              AND (QUERY(s <* synonymous_names |
              'ISO13584_IEC61360_DICTIONARY_SCHEMA.LABEL_WITH_LANGUAGE' IN
              TYPEOF(s)) = []));
```

```
END_ENTITY; -- item_names
```

And update the WR3 formal proposition with:

WR3: if no **languages** are provided, **preferred_name**, **short_name** and **synonymous_names** shall not be translated.

Clause D.3.9.1, p. 87

*The **current.definition[1]** attribute is unnecessarily qualified (twice). Change it as follows:*

```
FUNCTION acyclic_superclass_relationship(
      current: class_BSU; visited: SET OF class): LOGICAL;

IF SIZEOF(current.definition) = 1 THEN
      IF current.definition[1] IN visited THEN
...
      ELSE
            IF EXISTS(current.definition[1]\class.its_superclass)
            THEN
                  RETURN (acyclic_superclass_relationship(
                        current.definition[1]\class.its_superclass,
                        visited + current.definition[1]));
            ELSE
                  RETURN (TRUE);
            END_IF;
      END_IF;
ELSE
...
END_IF;

END_FUNCTION; -- acyclic_superclass_relationship
```

Clause D.3.9.2, p. 87

*Remove the **at_most_two_synonyms_per_language** function.*

Clause D.3.9.4, p. 88

*The existing function does only work for **non_quantitative_code_type**. Therefore, in order to make it working for **non_quantitative_int_type**, change its content as follows:*

```
FUNCTION codes_are_unique(values: LIST OF dic_value): BOOLEAN;
LOCAL
    ls: SET OF STRING := [];
    li: SET OF INTEGER := [];
END_LOCAL;

IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.VALUE_CODE_TYPE' IN
TYPEOF(values[1].value_code))
THEN
      REPEAT i := 1 TO  SIZEOF(values);
      ls := ls + values[i].value_code;
      END_REPEAT;

    RETURN(SIZEOF(values) = SIZEOF(ls));
ELSE

      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.INTEGER_TYPE' IN
TYPEOF(values[1].value_code))
      THEN
```

```
            REPEAT i := 1 TO SIZEOF(values);
            li := li + values[i].value_code;
            END_REPEAT;

          RETURN(SIZEOF(values) = SIZEOF(li));
    ELSE
        RETURN(?);
    END_IF;

END_IF;

END_FUNCTION;
```

Clause D.3.9.6, p. 89

The **is_subclass** function does not take into account the fact that the **sub** and **super** parameters may be the same (especially to assert the applicability of a property in a given class, defined as visible in one of its superclass or in the current class itself).
Add as a second IF statement:

```
FUNCTION is_subclass(sub,super : class) : LOGICAL;
...
        IF sub = super
        THEN
                RETURN (TRUE);
        END_IF;
        IF NOT EXISTS(sub.its_superclass)
...
END_FUNCTION;
```

*In the **is_subclass** function, **its_superclass.definition[1]** is unnecessary qualified by **\class** in two different locations. Change it as follows:*

```
  IF (sub.its_superclass.definition[1] = super) THEN
      RETURN ( TRUE );
  ELSE
      RETURN(is_subclass(sub.its_superclass.definition[1],
            super));
  END_IF;
```

Clause D.3.9.7, p. 90

The EXPRESS FORMAT function is not used correctly (twice). Change it as follows:

```
FUNCTION string_for_derived_unit (u: derived_unit): STRING;

      FUNCTION string_for_derived_unit_element
...
      LOCAL
            result: STRING;
      END_LOCAL;

      result := string_for_named_unit(u.unit);
      IF (u.exponent <> 0) THEN
            IF (u.exponent > 0) OR NOT neg_exp THEN
                  result:=result+'**' + FORMAT (ABS(u.exponent), '2I')[2];
            ELSE
                  result := result + '**' + FORMAT (u.exponent, '2I')[2];
            END_IF;
```

```
      END_IF;
      RETURN (result);
      END_FUNCTION; -- string_for_derived_unit_element
...
END_FUNCTION; -- string_for_derived_unit
```

Clause D.3.9.8, p. 91

*The '**u**' formal parameter is unnecessarily qualified when used in the function body. Change it as follows:*

```
FUNCTION string_for_named_unit (u: named_unit): STRING;

IF 'MEASURE_SCHEMA.SI_UNIT' IN TYPEOF(u) THEN
      RETURN (string_for_SI_unit (u));
ELSE
...
END_IF;
END_FUNCTION; -- string_for_named_unit
```

Clause D.3.9.11, p. 94

*When called with an indeterminate value, the **all_class_descriptions_reachable** function enters in a recursive loop: Change its content as follows:*

```
FUNCTION all_class_descriptions_reachable (cl:class_BSU): BOOLEAN;

IF NOT EXISTS(cl)
THEN
      RETURN(?);
END_IF;

IF SIZEOF(cl.definition) = 0
THEN
      RETURN(FALSE);
END_IF;

IF NOT (EXISTS(cl.definition[1]\class.its_superclass))
THEN
      RETURN (TRUE);
ELSE
      RETURN(all_class_descriptions_reachable(
                              cl.definition[1]\class.its_superclass));
END_IF;

END_FUNCTION; -- all_class_descriptions_reachable
```

Clause D.3.9.12, p. 95

*The assignment of the `USEDIN` function result to the **s** variable shall be modified as follows:*

```
s:= s + USEDIN(cl,
'ISO13584_IEC61360_DICTIONARY_SCHEMA.PROPERTY_BSU.NAME_SCOPE');
```

Clause D.3.9.13, p. 95

The assignment of the `USEDIN` *function result to the* ***s*** *variable shall be modified as follows:*

```
s:= s + USEDIN(cl,
'ISO13584_IEC61360_DICTIONARY_SCHEMA.DATA_TYPE_BSU.NAME_SCOPE');
```

Clause D.3.9.15, p. 97

In the EXPRESS specification of the ***compute_known_applicable_data_types*** *function, replace:*

```
REPEAT i:=1 TO SIZEOF(cl.definition[1]\class.described_by);
s:=s+cl.definition[1]\class.defined_types[i];
END_REPEAT;
```

with

```
REPEAT i:=1 TO SIZEOF(cl.definition[1]\class.defined_types);
s:=s+cl.definition[1]\class.defined_types[i];
END_REPEAT;
```

New clause D.3.9.17, p. 98

Create a new function: ***check_properties_applicability***
(to be numbered D.3.9.16 due to deletion of D.3.9.2)

The ***check_properties_applicability*** *function checks that only those properties that are not applicable for a class by inheritance may become applicable to this class by virtue of being referenced by the* ***described_by*** *attribute.*

EXPRESS specification:

```
    *)

 FUNCTION check_properties_applicability(cl: class): LOGICAL;
 LOCAL
        inter: SET OF property_bsu := [];
 END_LOCAL;

 IF EXISTS(cl.its_superclass)
 THEN
        IF (SIZEOF(cl.its_superclass.definition)=1)
        THEN
                inter := (list_to_set(cl.described_by) * cl.its_superclass.
                definition[1]\class.known_applicable_properties);
                RETURN(inter=[]);
        ELSE
                RETURN(UNKNOWN);
        END_IF;
 ELSE
        RETURN(TRUE);
 END_IF;
 END_FUNCTION;
(*
```

New clause D.3.9.18, p. 98

Create a new function: ***check_datatypes_applicability***
(to be numbered D.3.9.17 due to deletion of D.3.9.2)

The ***check_datatypes_applicability*** *function checks that only those datatypes that are not applicable for a class by inheritance may become applicable to this class by virtue of being referenced by the* ***defined_types*** *attribute.*

EXPRESS specification:

```
    *)

FUNCTION check_datatypes_applicability(cl: class): LOGICAL;
LOCAL
        inter: SET OF data_type_bsu := [];
END_LOCAL;

IF EXISTS(cl.its_superclass)
THEN
        IF (SIZEOF(cl.its_superclass.definition) = 1)
        THEN
                inter := cl.defined_types * cl.its_superclass.
                        definition[1]\class.known_applicable_data_types;
                RETURN(inter=[]);
        ELSE
                RETURN(UNKNOWN);
        END_IF;
ELSE
        RETURN(TRUE);
END_IF;
END_FUNCTION;
```

Clause D.4, p. 98

Change Figure D.12 as follows:

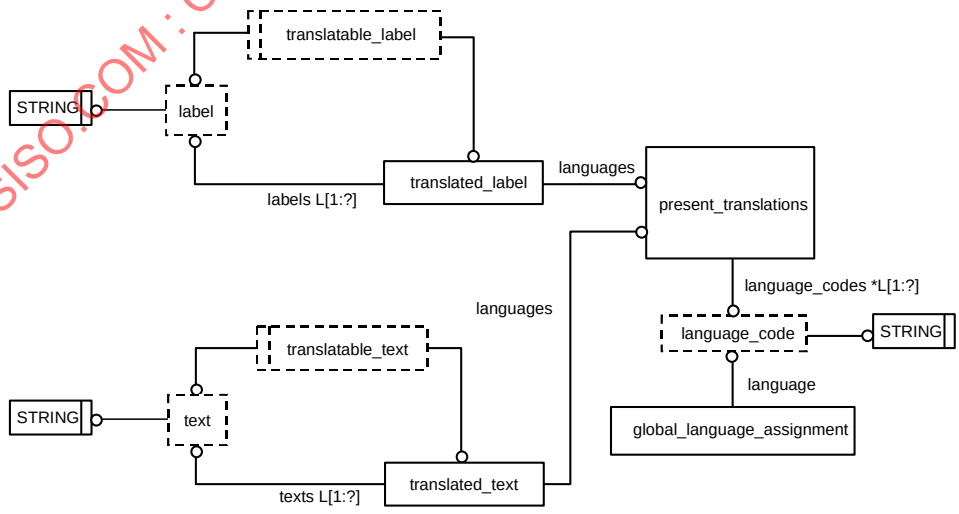

Figure D.1 — ISO13584_IEC61360_language_resource_schema and support_resource_schema

Clause D.5, p. 103

The supplier code assigned to IEC 61360-4 is not well-formed: '01122//61360-4'
Change with:
'112/2///61360_4_1'

The supplier code assigned to ICS is not well-formed: '01123//-00'
Change with:
'112/3///_00'

Moreover, there is a lot of errors inside the physical file:
- *attributes for which the type is redefined shall not appear with a star ('*') in the physical file:*
  Unnecessary "" have been removed*
- *the second attribute of the **item_names** entity data type is : `synonymous_names: SET OF syn_name_type;`*
  *If this attribute is not intended to be defined for an **item_names** instance, it shall be valued with '()', and not '$'.*
  *All the **item_names** instances have been corrected accordingly.*

- The #356 instance is not correctly built:

Change:
#356=DIC_UNIT(#357, 'm');
#357=SI_UNIT(*, $, .METRE.);

By:
#356=DIC_UNIT(#357, #358);
#357=SI_UNIT(*, $, .METRE.);
#358=MATHEMATICAL_STRING('m','m');

The correct content is as follows:

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('Example physical file'), '2;1');
FILE_NAME('example.spf', '2001-01-29', ('IEC SC3D WG2'), (),
'Version 1', '', '');
FILE_SCHEMA(('example_schema'));
ENDSEC;
DATA;
/*
D.5.1.1  Supplier data
*/
#1=SUPPLIER_BSU('112/2///61360_4_1', *); /*according to ISO 13584-26*/
#2=SUPPLIER_ELEMENT(#1, #3, '01', #4, #5);
#3=DATES('1994-09-16', '1994-09-16', $);
#4=ORGANIZATION('IEC', 'IEC Maintenance Agency', 'The IEC Maintenance Agency as described in IEC
61360-3: "Maintenance and Validation Procedures"');
#5=ADDRESS('to be determined', $, $, $, $, $, $, $, $, $, $, $);
#10=SUPPLIER_BSU('112/3///_00', *);   /* ISO/IEC ICS */
/*

D.5.1.2  Root class data
The AAA000 IEC root class provides a name scope corresponding to the whole future IEC 61360-4 standard.
It covers two trees, one for materials, one for components, therefore the class is defined as an item_class. It is
a subtype of ICS root.
*/
#90=CLASS_BSU('OO', '001', #10);       /* ICS root */
#100=CLASS_BSU('AAA000', '001', #1);
#101=ITEM_CLASS(#100, #3, '01', #102, TEXT('IEC root class that provides a name scope corresponding to
the whole IEC 61360-4 standard. It covers two trees, one for materials, one for components'), $, $, $, #90,
(#110), (), $, (#110), (), $);
#102=ITEM_NAMES(LABEL('IEC root class'), (), LABEL('IEC root'), $, $);
```

```
#110=PROPERTY_BSU('AAE000', '001', #100);
#111=NON_DEPENDENT_P_DET(#110, #3, '01', #112, TEXT('the type of tree: material or component'), $, $, $, $, (), $, $, #113, $);
#112=ITEM_NAMES(LABEL('type of tree'), (), LABEL('tree type'), $, $);
#113=NON_QUANTITATIVE_CODE_TYPE('A..8', #114);
#114=VALUE_DOMAIN((#120, #122), $, $, ());
#120=DIC_VALUE(VALUE_CODE_TYPE('MATERIAL'), #121, $);
#121=ITEM_NAMES(LABEL('material tree'), (), LABEL('mat tree'), $, $);
#122=DIC_VALUE(VALUE_CODE_TYPE('COMPONS'), #123, $);
#123=ITEM_NAMES(LABEL('component tree'), (), LABEL('comp tree'), $, $);
/*
```

D.5.1.3 Material data

```
*/
#200=CLASS_BSU('AAA218', '001', #1);
#201=MATERIAL_CLASS(#200, #3, '01', #202, TEXT('root class of the materials tree'), $, $, $, #100, (#210, #230), (), $, (#210), (#205), 'MATERIAL');
#202=ITEM_NAMES(LABEL('materials root class'), (), LABEL('materials root'), $, $);
#205=CLASS_VALUE_ASSIGNMENT(#110, 'MATERIAL');
#210=PROPERTY_BSU('AAF311', '005', #100);
#211=NON_DEPENDENT_P_DET(#210, #3, '01', #212, TEXT('code of the type of material'), $, $, $, $, (), $, 'A57', #213, $);
#212=ITEM_NAMES(LABEL('material type'), (), LABEL('material type'), $, $);
#213=NON_QUANTITATIVE_CODE_TYPE('M..3', #214);
#214=VALUE_DOMAIN((#220, #222, #224, #226), $, $, ());
#220=DIC_VALUE(VALUE_CODE_TYPE('ACO'), #221, $);
#221=ITEM_NAMES(LABEL('acoustical'), (), LABEL('acoustical'), $, $);
#222=DIC_VALUE(VALUE_CODE_TYPE('MG'), #223, $);
#223=ITEM_NAMES(LABEL('magnetic'), (), LABEL('magnetical'), $, $);
#224=DIC_VALUE(VALUE_CODE_TYPE('OP'), #225, $);
#225=ITEM_NAMES(LABEL('optical'), (), LABEL('optical'), $, $);
#226=DIC_VALUE(VALUE_CODE_TYPE('TH'), #227, $);
#227=ITEM_NAMES(LABEL('thermal-electric'), (), LABEL('th-electric'), $, $);
#230=PROPERTY_BSU('AAF286', '005', #100);
#231=NON_DEPENDENT_P_DET(#230, #3, '01', #232, TEXT('The nominal density (in kg/m**3) of a material.'), $, $, $, #233, (), $, 'K02', #234, $);
#232=ITEM_NAMES(LABEL('density'), (), LABEL('density'), $, $);
#233=MATHEMATICAL_STRING('$r_d', '&rho;<sub>d</sub>');
#234=REAL_MEASURE_TYPE('NR3..3.3ES2', #235);
#235=DIC_UNIT(#236, $);
#236=DERIVED_UNIT((#237, #239));
#237=DERIVED_UNIT_ELEMENT(#238, 1.0);
#238=SI_UNIT(*, .KILO., .GRAM.);
#239=DERIVED_UNIT_ELEMENT(#240, -3.0);
#240=SI_UNIT(*, $, .METRE.);
/*
```

D.5.1.4 Component data

```
*/
#300=CLASS_BSU('EEE000', '001', #1);
#301=COMPONENT_CLASS(#300, #3, '01', #302, TEXT('root class of the components tree'), $, $, $, #100, (#310, #330, #350), (), $, (#310), (#305), 'COMPONS');
#302=ITEM_NAMES(LABEL('components root class'), (), LABEL('components root'), $, $);
#305=CLASS_VALUE_ASSIGNMENT(#110, 'COMPONS');
#310=PROPERTY_BSU('AAE001', '005', #100);
#311=NON_DEPENDENT_P_DET(#310, #3, '01', #312, TEXT('Code of the main functional class to which a component belongs'), $, $, $, $, (), $, 'A52', #313, $);
#312=ITEM_NAMES(LABEL('main class of component'), (), LABEL('main class'), $, $);
#313=NON_QUANTITATIVE_CODE_TYPE('M..3', #314);
#314=VALUE_DOMAIN((#320, #322, #324, #326), $, $, ());
```

```
#320=DIC_VALUE(VALUE_CODE_TYPE('EE'), #321, $);
#321=ITEM_NAMES(LABEL('EE (electric / electronic)'), (), LABEL('EE'), $, $);
#322=DIC_VALUE(VALUE_CODE_TYPE('EM'), #323, $);
#323=ITEM_NAMES(LABEL('electromechanical'), (), LABEL('electromech'), $, $);
#324=DIC_VALUE(VALUE_CODE_TYPE('ME'), #325, $);
#325=ITEM_NAMES(LABEL('mechanical'), (), LABEL('mechanical'), $, $);
#326=DIC_VALUE(VALUE_CODE_TYPE('MP'), #327, $);
#327=ITEM_NAMES(LABEL('magnetic part'), (), LABEL('magnetic'), $, $);
#330=PROPERTY_BSU('AAF267', '005', #100);
#331=NON_DEPENDENT_P_DET(#330, #3, '01', #332, TEXT('The nominal distance (in m) between the inside of the two tapes used for taped products with axial leads'), $, $, $, #333, (), $, 'T03', #334, $);
#332=ITEM_NAMES(LABEL('inner tape spacing'), (), LABEL('inner tape spac'), $, $);
#333=MATHEMATICAL_STRING('b_tape', 'b<sub>tape</sub>');
#334=LEVEL_TYPE((.NOM.), #335);
#335=REAL_MEASURE_TYPE('NR3..3.3ES2', #336);
#336=DIC_UNIT(#337, $);
#337=SI_UNIT(*, $, .METRE.);
#350=PROPERTY_BSU('AAE022', '005', #100);
#351=NON_DEPENDENT_P_DET(#350, #3, '01', #352, TEXT('The value as specified by level (miNoMax) of the outside diameter (in m) of a component with a body of circular cross-section'), $, $, $, #353, (), $, 'T03', #354, $);
#352=ITEM_NAMES(LABEL('outside diameter'), (), LABEL('outside diam'), $, $);
#353=MATHEMATICAL_STRING('d_out', 'd<sub>out</sub>');
#354=LEVEL_TYPE((.MIN.,.NOM.,.MAX.), #355);
#355=REAL_MEASURE_TYPE('NR3..3.3ES2', #356);
#356=DIC_UNIT(#357, #358);
#357=SI_UNIT(*, $, .METRE.);
#358=MATHEMATICAL_STRING('m','m');
/*
```

D.5.1.5 Electric / Electronic component data

```
*/
#400=CLASS_BSU('EEE001', '001', #1);
#401=COMPONENT_CLASS(#400, #3, '01', #402, TEXT('electric / electronic components'), $, $, $, #300, (#410, #470), (), $, (#410), (#405), 'EE');
#402=ITEM_NAMES(LABEL('EE components'), (), LABEL('EE components'), $, $);
#405=CLASS_VALUE_ASSIGNMENT(#310, 'EE');
#410=PROPERTY_BSU('AAE002', '005', #100);
#411=NON_DEPENDENT_P_DET(#410, #3, '01', #412, TEXT('Code of the category to which an electric/electronic component belongs.'), $, $, $, $, (), $, 'A52', #413, $);
#412=ITEM_NAMES(LABEL('category EE component'), (), LABEL('categ EE comp'), $, $);
#413=NON_QUANTITATIVE_CODE_TYPE('M..3', #414);
#414=VALUE_DOMAIN((#420, #422, #424, #426, #428, #430, #432, #434, #436, #438, #440), $, $, ());
#420=DIC_VALUE(VALUE_CODE_TYPE('AMP'), #421, $);
#421=ITEM_NAMES(LABEL('amplifier'), (), LABEL('amplifier'), $, $);
#422=DIC_VALUE(VALUE_CODE_TYPE('ANT'), #423, $);
#423=ITEM_NAMES(LABEL('antenna (aerial)'), (), LABEL('antenna (aer)'), $, $);
#424=DIC_VALUE(VALUE_CODE_TYPE('BAT'), #425, $);
#425=ITEM_NAMES(LABEL('battery'), (), LABEL('battery'), $, $);
#426=DIC_VALUE(VALUE_CODE_TYPE('CAP'), #427, $);
#427=ITEM_NAMES(LABEL('capacitor'), (), LABEL('capacitor'), $, $);
#428=DIC_VALUE(VALUE_CODE_TYPE('CND'), #429, $);
#429=ITEM_NAMES(LABEL('conductor'), (), LABEL('conductor'), $, $);
#430=DIC_VALUE(VALUE_CODE_TYPE('DEL'), #431, $);
#431=ITEM_NAMES(LABEL('delay line'), (), LABEL('delay line'), $, $);
#432=DIC_VALUE(VALUE_CODE_TYPE('DID'), #433, $);
#433=ITEM_NAMES(LABEL('diode device'), (), LABEL('diode device'), $, $);
#434=DIC_VALUE(VALUE_CODE_TYPE('FIL'), #435, $);
#435=ITEM_NAMES(LABEL('filter'), (), LABEL('filter'), $, $);
#436=DIC_VALUE(VALUE_CODE_TYPE('IC'), #437, $);
```

```
#437=ITEM_NAMES(LABEL('integrated circuit'), (), LABEL('IC'), $, $);
#438=DIC_VALUE(VALUE_CODE_TYPE('IND'), #439, $);
#439=ITEM_NAMES(LABEL('inductor'), (), LABEL('inductor'), $, $);
#440=DIC_VALUE(VALUE_CODE_TYPE('LAM'), #441, $);
#441=ITEM_NAMES(LABEL('lamp'), (), LABEL('lamp'), $, $);
/* etc. */
#470=PROPERTY_BSU('AAE754', '005', #100);
#471=NON_DEPENDENT_P_DET(#470, #3, '01', #472, TEXT('The number of electrical terminals of an electric/electronic or electromechanical component'), $, $, $, #473, (), $, 'Q56', #474, $);
#472=ITEM_NAMES(LABEL('number of terminals'), ('number of pins'), LABEL('nr of terminals'), $, $);
#473=MATHEMATICAL_STRING('N_term', 'N<sub>term</sub>');
#474=INT_TYPE('NR1..4');

ENDSEC;
END-ISO-10303-21;
```

Clause D.5.2.3, p. 108

Due to change in D.3.6.2, change the cardinality of **synonymous_symbols** in the templates of the property_DET subtypes as follows:

e_i_n=CONDITION_DET(identified_by: property_BSU, ..., synonymous_symbols: **SET [0 : ?]** OF mathematical_string, ...

e_i_n=DEPENDENT_P_DET(.... synonymous_symbols: **SET [0 : ?]** OF mathematical_string, ...

e_i_n=NON_DEPENDENT_P_DET(**...** synonymous_symbols: **SET [0 : ?]** OF mathematical_string, **...**

Clause Annex G, p. 119

A lot of errors have been identified in the different Express-G diagrams. Consequently, all these diagrams have been updated.

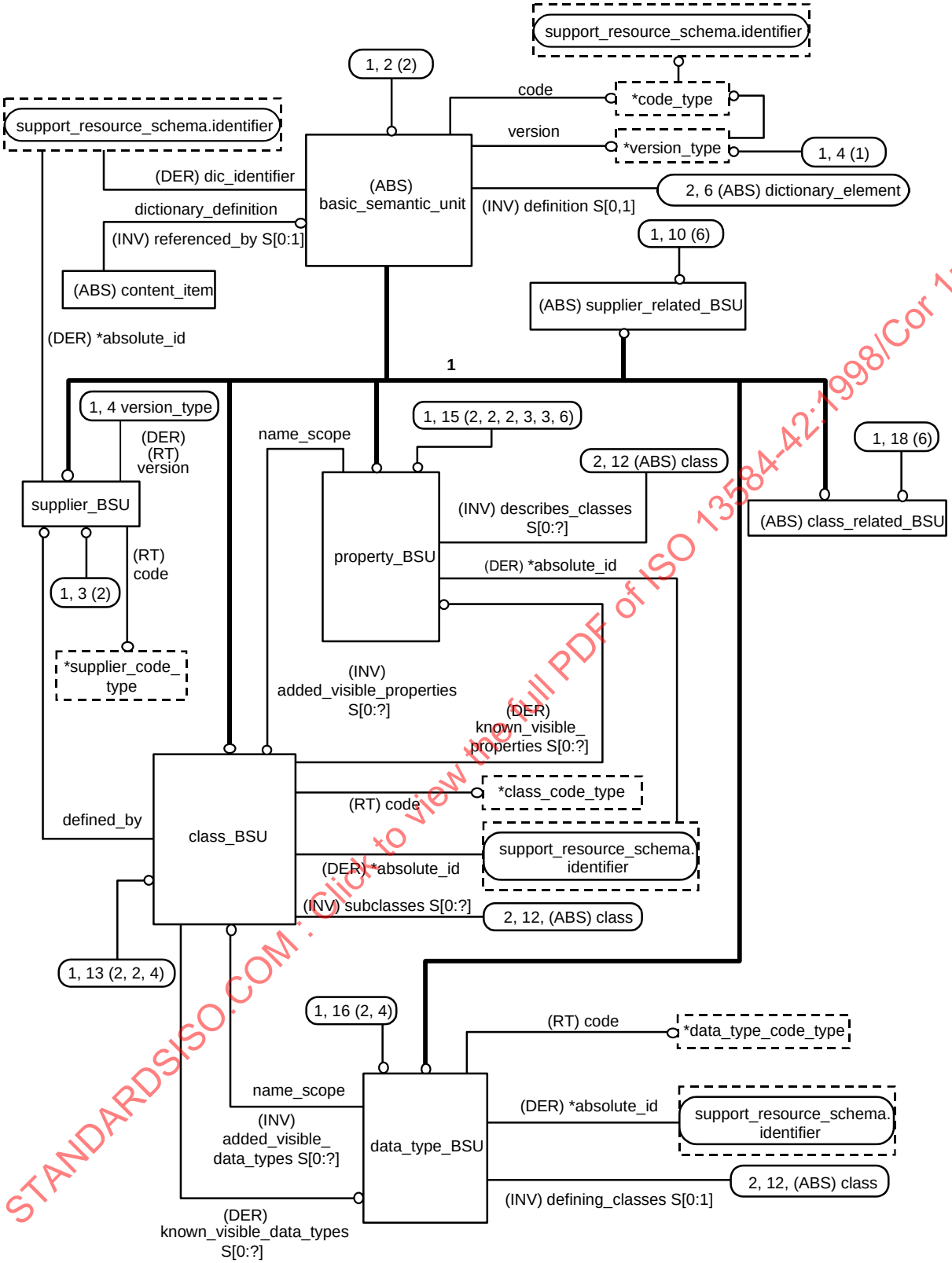

Figure G.1 — ISO13584_IEC61360_dictionary_schema - Basic semantic units - EXPRESS-G diagram 1 of 6

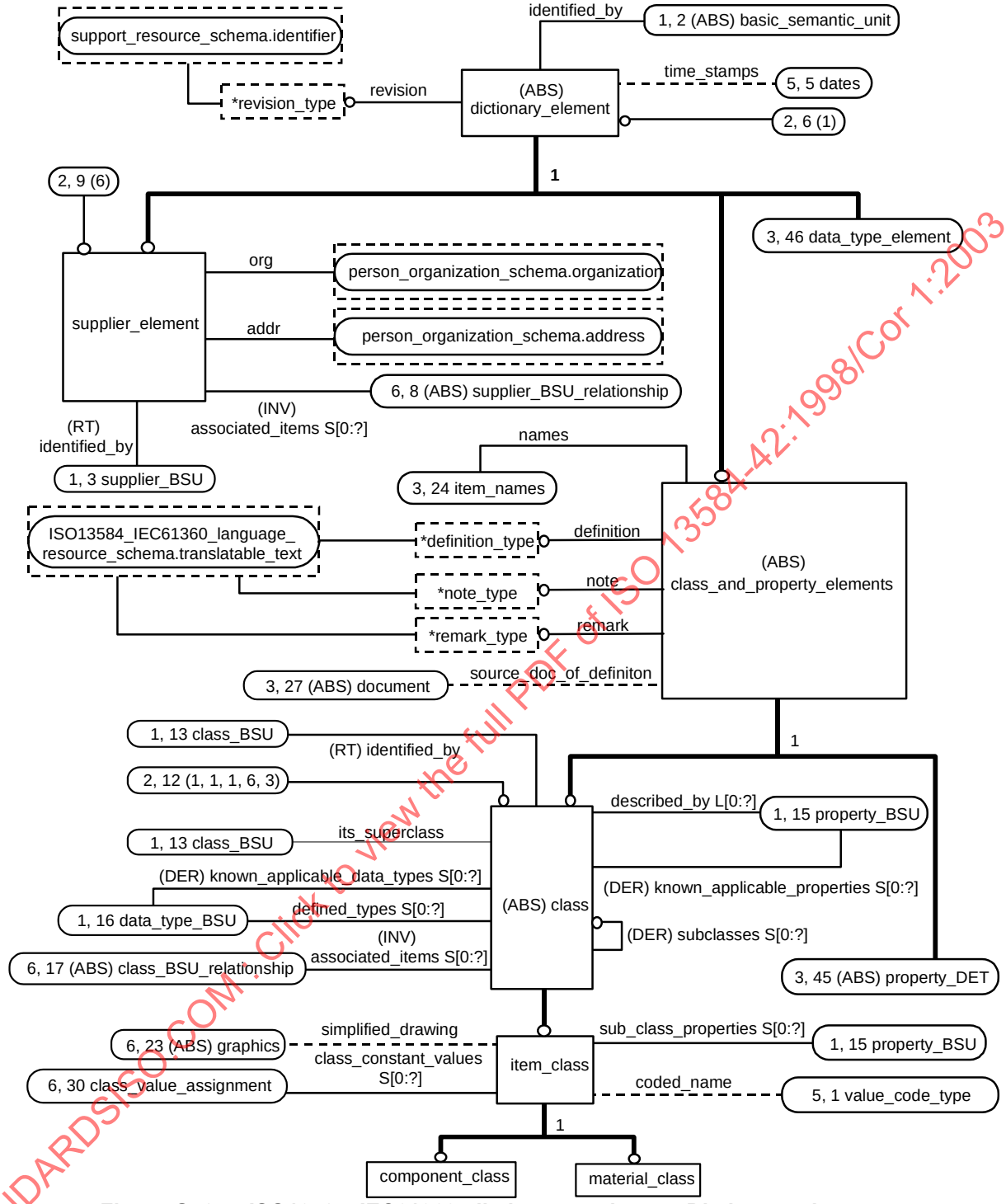

Figure G. 2 — ISO13584_IEC61360_dictionary_schema - Dictionary elements - EXPRESS-G diagram 2 of 6