INTERNATIONAL STANDARD

ISO 10303-21

First edition
1994-12-15

Industrial automation systems and integration — Product data representation and exchange —

Part 21:
Implementation methods: Clear text encoding of the exchange structure

Systèmes d'automatisation industrielle et intégration — Représentation et échange de données de produits —

Partie 21: Méthodes de mise en application: Encodage en texte clair des fichiers d'échange

Reference number
ISO 10303-21:1994(E)

Contents

Annexes

Figures

Tables

Foreword

The International Organization for Standardization (ISO) 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.

Draft International Standards adopted by 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.

International Standard ISO 10303-21 was prepared by Technical Committee ISO TC184, *Industrial automation systems and integration*, Subcommittee SC4, *Industrial data and global manufacturing languages*.

ISO 10303 consists of the following parts under the general title *Industrial automation systems and integration - Product data representation and exchange*:

— Part 1 Overview and fundamental principles;

— Part 11 Description methods: The EXPRESS language reference manual;

— Part 21 Implementation methods: Clear text encoding of the exchange structure;

— Part 22 Implementation methods: Standard data access interface;

— Part 31 Conformance testing methodology and framework: General concepts;

— Part 32 Conformance testing methodology and framework: requirements on testing laboratories and clients;

— Part 41 Integrated generic resources: Fundamentals of product description and support;

— Part 42 Integrated generic resources: Geometric and topological representation;

— Part 43 Integrated generic resources: Representation structures;

— Part 44 Integrated generic resources: Product structure configuration;

— Part 45 Integrated generic resources: Materials;

— Part 46 Integrated generic resources: Visual presentation;

— Part 47 Integrated generic resources: Shape variation tolerances;

— Part 49 Integrated generic resources: Process structure and properties;

— Part 101, Integrated application resources: Draughting;

— Part 104, Integrated application resources: Finite element analysis;

— Part 105, Integrated application resources: Kinematics;

— Part 201, Application protocol: Explicit draughting;

— Part 202, Application protocol: Associative draughting;

— Part 203, Application protocol: Configuration controlled design;

— Part 207, Application protocol: Sheet metal die planning and design;

— Part 210, Application protocol: Printed circuit assembly product design data;

— Part 213, Application protocol: Numerical control process plans for machined parts.

The structure of this International Standard is described in ISO 10303-1. The numbering of the parts of this International Standard reflects its structure:

— Part 11 specifies the description method;

— Parts 21 and 22 specify the implementation methods;

— Parts 31 and 32 specify the conformance testing methodology and framework;

— Parts 41 to 49 specify the integrated generic resources;

— Parts 101 to 105 specify the integrated application resources;

— Parts 201 to 213 specify the application protocols.

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

Annexes A, B and C form an integral part of this part of ISO 10303. Annexes D, E, and F are for information only.

Introduction

ISO 10303 is an International Standard for the computer-interpretable representation and exchange of product data. The objective is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product, independent from any particular system. The nature of this description makes it suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and archiving.

This International Standard is organized as a series of parts, each published separately. The parts of ISO 10303 fall into one of the following series: description methods, integrated resources, application protocols, abstract test suites, implementation methods, and conformance testing. The series are described in ISO 10303-1. This part of ISO 10303 is a member of the implementation methods series.

This part of ISO 10303 specifies a mechanism that allows product data represented using the EXPRESS language, specified in ISO 10303-11, to be transferred from one computer system to another.

Major subdivisions in this part of ISO 10303 are

— specification of the exchange structure syntax;

— mapping from an EXPRESS schema onto this syntax.

NOTE - The examples of EXPRESS usage in this part of ISO 10303 do not conform to any particular style rules. Indeed, the examples sometimes use poor style to conserve space or to concentrate on the important points. The examples are not intended to reflect the content of the information models defined in other parts of this International Standard. They are crafted to show particular features of EXPRESS or of the exchange structure. Many examples are annotated in a way that is not consistent with the syntax rules of this part of ISO 10303. These annotations are introduced by symbolic arrows, either horizontal '---->', or vertical. These annotations should be ignored when considering the parse rules. Any similarity between the examples and the normative models specified in other parts of this International Standard should be ignored. Several mapping examples have been provided throughout this document. Additional *spaces* and new lines have been inserted into some of these examples to aid readability. These *spaces* and new lines need not appear in an exchange structure.

Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure

1 Scope

This part of ISO 10303 specifies an exchange structure format using a clear text encoding of product data for which the conceptual model is specified in the EXPRESS language (ISO 10303-11). The file format is suitable for the transfer of product data among computer systems.

The mapping from the EXPRESS language to the syntax of the exchange structure is specified. Any EXPRESS schema can be mapped onto the exchange structure syntax.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO 10303. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO 10303 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO 3788:1990, *Information processing — 9-Track, 12,7 mm (0.5 in) wide magnetic tape for information interchange using phase encoding at 126 ftpmm (3200 ftpi) — 63 cpmm (1600 cpi).*

ISO 8601:1988, *Data elements and interchange formats — Information interchange — Representation of dates and times.*

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

ISO 8859-2:1987, *Information processing — 8 bit single-byte coded graphic character sets — Part 2: Latin alphabet No. 2.*

ISO 8859-3:1987, *Information processing — 8 bit single-byte coded graphic character sets — Part 3: Latin alphabet No. 3.*

ISO 8859-4:1987, *Information processing — 8 bit single-byte coded graphic character sets — Part 4: Latin alphabet No. 4.*

ISO 8859-5:1987, *Information processing — 8 bit single-byte coded graphic character sets — Part 5: Latin/Cyrillic alphabet.*

ISO 8859-6:1987, *Information processing — 8 bit single-byte coded graphic character sets — Part 6: Latin/Arabic alphabet.*

ISO 8859-7:1987, *Information processing — 8 bit single-byte coded graphic character sets — Part 7: Latin/Greek alphabet.*

ISO 8859-8:1987, *Information processing — 8 bit single-byte coded graphic character sets — Part 8: Latin/Hebrew alphabet.*

ISO 8859-9:1987, *Information processing — 8 bit single-byte coded graphic character sets — Part 9: Latin alphabet No. 5.*

ISO 10303-1:1994, *Industrial automation systems — Product data representation and exchange — Part 1: Overview and fundamental principles.*

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

ISO 10303-44:1994, *Industrial automation systems — Product data representation and exchange — Part 44: Product structure configuration.*

ISO/IEC 10646-1:1993-1, *Information Processing — Multiple octet coded character set — Part 1: Architecture and basic multilingual plane.*

3 Definitions

3.1 Terms defined in ISO 8859-1

This part of ISO 10303 makes use of the following terms defined in ISO 8859-1.

— byte;

— character;

— graphic character.

3.2 Terms defined in ISO 10646

This part of ISO 10303 makes use of the following term defined in ISO 10646.

— basic multilingual plane.

3.3 Terms defined in ISO 10303-1

This part of ISO 10303 makes use of the following term defined in ISO 10303-1.

— application protocol;

— exchange structure.

3.4 Terms defined in ISO 10303-11

This part of ISO 10303 makes use of the following terms defined in ISO 10303-11.

— data type;

— entity;

— token.

3.5 Terms defined in ISO 10303-44

This part of ISO 10303 makes use of the following terms defined in ISO 10303-44.

— directed acyclic graph;

— leaf node;

— node;

— tree.

3.6 Other definitions

For the purposes of this part of ISO 10303, the following definitions apply.

3.6.1 basic alphabet: the set of characters G(02/00) through G(07/14) of ISO 8859-1.

3.6.2 clear text encoding: the encoding of information that only uses 8-bit byte values corresponding to the basic alphabet.

3.6.3 control directive: a sequence of characters in the basic alphabet.

3.6.4 keyword: a special sequence of characters identifying an entity or a defined type in the exchange structure.

3.6.5 literal: an item in a language definition that stands for itself.

3.6.6 section: a collection of data of the same functional category of information.

3.6.7 sequential file: a file that can only be accessed in a sequential manner.

3.6.8 token separator: a sequence of one or more 8-bit bytes that separate any two tokens.

4 Abbreviations

For the purposes of this part of ISO 10303, the following abbreviations apply:

BMP	Basic multilingual plane;
WSN	Wirth Syntax Notation.

5 Exchange structure fundamental concepts and assumptions

5.1 Introduction

The exchange structure is described by an unambiguous, context-free grammar to facilitate parsing by software. The grammar is expressed in Wirth Syntax Notation that is described in annex B. The form of product data in the exchange structure is specified using a mapping from the EXPRESS language.

5.2 Notational and typographical conventions

Any *quotation marks* used in this part of ISO 10303 are not part of the text that appears in the exchange structure but serve to delimit that text. This statement applies to all places in the text where *quotation marks* are used. Table 2, table 3, and table 4 form an exception to this rule as the *quotation marks* used in those tables form part of the WSN rules.

In ISO 8859, each character is assigned an identifying name. When that name is used in this part of ISO 10303, it is typeset in *italics* to distinguish if from ordinary text. Thus *comma* is used to refer to ",", *low line* refers to "_", and *capital letter A* refers to "A".

Within examples in this part of ISO 10303, annotation is introduced by the sequence ----> where clarification is required.

5.3 Conformance

Two levels of conformance are specified:

— syntactical conformance of the exchange structure: an exchange structure conforms to ISO 10303-21 if the requirements of this part of ISO 10303 are satisfied;

— schema conformance of the exchange structure: the product data represented in the exchange structure conforms to the schema listed in the header section of the exchange structure if all requirements and constraints of this schema are satisfied and the mapping requirements defined in clauses 10 and 11 of this part of ISO 10303 are satisfied.

Syntactical conformance is a prerequisite for schema conformance.

Two classes of syntactical conformance are defined by this part of ISO 10303, depending on the method chosen for the encoding of entity instances whose entity types are subtypes/sypertypes (see 11.2.5). An implementation that claims syntactical conformance to this part of ISO 10303 shall read or write files or both that exhibit syntactical conformance in (at least) one of these two conformance classes.

An implementation that claims schema conformance to this part of ISO 10303 shall read or write files or both that exhibit schema as well as syntactical conformance.

6 Formal definitions

6.1 Formal notation

Wirth Syntax Notation (WSN) is used in this part of ISO 10303 to specify the syntax of the exchange structure in a formal notation. WSN is described in annex B. Literals appearing within this notation are delimited by *quotation marks*.

6.2 Basic alphabet definition

The alphabet of the exchange structure is defined as the characters from G(02/00) to G(07/14) of ISO 8859-1. This alphabet is represented in the exchange structure by the set of 8-bit bytes with decimal values 32 to 126. Table 1 - divides the basic alphabet into subsets. G(x/y) is a notation for the character in position (16 times x) + y in the code table in ISO 8859-1.

NOTE - Table D.1 gives the correspondence between the 8-bit bytes and their graphic representation in ISO 8859-1.

6.3 Exchange structure

The exchange structure shall be a sequential file using a clear text encoding. The exchange structure shall consist of two sections: the header section and the data section. The header section provides data relating to the exchange structure itself. The structure of the header section is specified in clause 9. The data section provides the data to be transferred. The structure of the data section is specified in clause 10. The exchange structure is defined by the WSN in table 3.

The exchange structure is a stream of 8-bit bytes that are encodings of the graphic characters of the basic alphabet. The graphic characters are collected into recognizable sequences called tokens. Tokens may be separated by token separators. The exchange structure can be considered as a sequence of tokens and token separators.

Table 1 - WSN defining subsets of the basic alphabet

```
SPACE     = " " .

DIGIT     = "0"  ¦ "1"  ¦ "2"  ¦ "3"  ¦ "4"  ¦ "5"  ¦ "6"  ¦ "7"
          ¦ "8"  ¦ "9"  .

LOWER     = "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"  .

UPPER     = "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"  ¦ "_"  .

SPECIAL   = "!"  ¦ """"  ¦ "*"  ¦ "$"  ¦ "%"  ¦ "&"  ¦ "."  ¦ "#"
          ¦ "+"  ¦ ","  ¦ "-"  ¦ "("  ¦ ")"  ¦ "?"  ¦ "/"  ¦ ":"
          ¦ ";"  ¦ "<"  ¦ "="  ¦ ">"  ¦ "@"  ¦ "["  ¦ "]"  ¦ "{"
          ¦ "¦"  ¦ "}"  ¦ "^"  ¦ "`"  .

REVERSE_SOLIDUS  = "\" .

APOSTROPHE = "'" .

CHARACTER = SPACE ¦ DIGIT ¦ LOWER ¦ UPPER ¦ SPECIAL
          ¦ REVERSE_SOLIDUS ¦ APOSTROPHE
```

6.4 Definition of tokens

The tokens used in the exchange structure are defined by the WSN in table 2.

6.5 WSN of the exchange structure

The syntax of the exchange structure is specified in table 3. Table 3 - references the tokens defined in table 2. The relationship between the syntax and the EXPRESS schema is specified in clause 11.

6.6 Token separators

A token separator is a element that separates two tokens. Token separators are *space*, the explicit print control directives, and comments. A token separator may appear between the terminals or non-terminals of the productions of table 3. Any number of token separators may appear wherever one token separator may appear. A token separator shall not appear within tokens except that explicit print control directives may also appear within binaries and within strings. Print control directives are defined in clause 12.

A comment shall be encoded as a *solidus asterisk* "/*" followed by any number of characters, and terminated by an *asterisk solidus* "*/". Any occurrence of *solidus asterisk* following the first occurrence shall not be significant, i.e. comments cannot be nested. All graphic characters appearing inside a comment shall not be significant to the exchange structure and are only intended to be read by humans.

Table 2 - WSN of token definitions

```
KEYWORD              = USER_DEFINED_KEYWORD ¦ STANDARD_KEYWORD .

USER_DEFINED_KEYWORD = "!" UPPER { UPPER ¦ DIGIT } .

STANDARD_KEYWORD     = UPPER { UPPER ¦ DIGIT } .

SIGN                 = "+" ¦ "-".

INTEGER              = [ SIGN ] DIGIT { DIGIT } .

REAL                 = [ SIGN ] DIGIT { DIGIT } "." { DIGIT }
                       [ "E" [ SIGN ] DIGIT { DIGIT } ].

NON_Q_CHAR           = SPECIAL ¦ DIGIT ¦ SPACE ¦ LOWER ¦ UPPER .

STRING               = "'" { NON_Q_CHAR ¦
                     APOSTROPHE APOSTROPHE ¦
                     REVERSE_SOLIDUS REVERSE_SOLIDUS ¦
                     CONTROL_DIRECTIVE } "'" .

ENTITY_INSTANCE_NAME = "#" DIGIT { DIGIT } .

ENUMERATION          = "." UPPER { UPPER ¦ DIGIT } "." .

HEX                  = "0" ¦ "1" ¦ "2" ¦ "3" ¦ "4" ¦ "5" ¦ "6" ¦ "7" ¦
                       "8" ¦ "9" ¦ "A" ¦ "B" ¦ "C" ¦ "D" ¦ "E" ¦ "F" .

BINARY               = """" ( "0" ¦ "1" ¦ "2" ¦ "3" ) { HEX } """" .
```

6.7 Parameter lists

The PARAMETER_LIST production specified in table 3 is a list of instances of a simple or structured data type.

7 Tokens

In the exchange structure, a token is a special token, a keyword, or a simple data type.

7.1 Special tokens

The special token "ISO-10303-21;" shall be used to open an exchange structure, and the special token "END-ISO-10303-21;" shall be used to close an exchange structure.

The special token "HEADER;" shall be used to open the Header section of an exchange structure, and the special token "ENDSEC;" shall be used to close the Header section of an exchange section.

The special token "DATA;" shall be used to open the Data section of an exchange structure, and the special token "ENDSEC;" shall be used to close the Data section of an exchange structure.

Table 3 - WSN of the exchange structure

```
 EXCHANGE_FILE        = "ISO-10303-21;"
                          HEADER_SECTION DATA_SECTION
                          "END-ISO-10303-21;" .
 HEADER_SECTION       = "HEADER;"
                          HEADER_ENTITY HEADER_ENTITY HEADER_ENTITY
                          [HEADER_ENTITY_LIST]
                          "ENDSEC;" .
 HEADER_ENTITY_LIST = HEADER_ENTITY { HEADER_ENTITY } .
 HEADER_ENTITY       = KEYWORD  "(" [ PARAMETER_LIST ] ")" ";" .
 PARAMETER_LIST      = PARAMETER { "," PARAMETER } .
 PARAMETER           = TYPED_PARAMETER  ¦
                       UNTYPED_PARAMETER ¦ OMITTED_PARAMETER  .
 TYPED_PARAMETER     = KEYWORD "(" PARAMETER ")" .
 UNTYPED_PARAMETER = "$" ¦ INTEGER ¦ REAL ¦ STRING ¦ ENTITY_INSTANCE_NAME
                     ¦ ENUMERATION ¦ BINARY ¦ LIST .
 OMITTED_PARAMETER = "*" .
 LIST                = "(" [ PARAMETER { "," PARAMETER } ] ")" .
 DATA_SECTION        = "DATA;" ENTITY_INSTANCE_LIST "ENDSEC;" .
 ENTITY_INSTANCE_LIST = ENTITY_INSTANCE { ENTITY_INSTANCE } .
 ENTITY_INSTANCE     = ENTITY_INSTANCE_NAME "=" [ SCOPE ] SIMPLE_RECORD ";" ¦
                       ENTITY_INSTANCE_NAME "=" [ SCOPE ] SUBSUPER_RECORD ";"
.
 SCOPE               = "&SCOPE"  ENTITY_INSTANCE_LIST
                       "ENDSCOPE" [ EXPORT_LIST ] .
 EXPORT_LIST         = "/" ENTITY_INSTANCE_NAME { "," ENTITY_INSTANCE_NAME }
"/" .
 SIMPLE_RECORD       = KEYWORD "(" [ PARAMETER_LIST ] ")" .
 SUBSUPER_RECORD     = "(" SIMPLE_RECORD_LIST ")" .
 SIMPLE_RECORD_LIST = SIMPLE_RECORD { SIMPLE_RECORD } .
```

The special token "&SCOPE" shall be used to open the Scope structure, and the special token "ENDSCOPE" shall be used to close the Scope structure.

7.2 Keywords

Keywords are sequences of graphic characters indicating an entity or a defined type in the exchange structure. Keywords shall consist of *capital letters*, *digits*, *low lines*, and possibly an *exclamation mark* "!". The *exclamation mark* shall occur at most once, and only as the first character in a keyword.

Keywords may be schema-defined keywords or user-defined keywords. Keywords that do not begin with the *exclamation mark* are schema-defined keywords. Keywords that begin with the *exclamation mark* are user-defined keywords. A user-defined keyword is the identifier for a named type (an entity data type or a defined type) in the EXPRESS schema governing the exchange structure. The meaning of a user-defined keyword is a matter of agreement between the partners using the exchange structure.

7.3 Simple data types

Six simple data types are used in exchange structures: integer, real, string, entity instance name, enumeration, and binary.

7.3.1 Integer

An integer shall be encoded as a sequence of one or more digits, as prescribed in table 2, optionally preceded by a *plus sign* "+" or a *minus sign* "-". Integers shall be expressed in base 10. If no sign is associated with the integer, the integer shall be assumed to be positive.

EXAMPLE 1 -

Valid file expressions of Integer	Meaning
16	Positive 16
+12	Positive 12
-349	Negative 349
012	Positive 12
00	Zero

Invalid file expressions of Integer	Problem
26 54	Contains *spaces*
32.0	Contains *full stop*
+ 12	Contains *space* between *plus sign* and digits

7.3.2 Real

A real shall be encoded as prescribed in table 2. The encoding shall consist of a decimal mantissa optionally followed by a decimal exponent. The decimal mantissa consists of an optional *plus sign* "+" or *minus sign* "-", followed by a sequence of one or more digits, followed by a *full stop* ".", followed by a sequence of zero or more digits. A decimal exponent consists of the *capital letter E* optionally followed by a *plus sign* "+" or *minus sign* "-", followed by one or more digits.

Note - No attempt is made to convey the concept of precision in this part of ISO 10303. Where a precise meaning is necessary, the sender and receiver of the exchange structure should agree on one. Where a precise meaning is required as part of the description of an entity data type, this meaning should be included in the entity data type definition in the EXPRESS schema.

EXAMPLE 2 -

Valid file expressions of Real	Meaning
+0.0E0	0.0
-0.0E-0	0.0, as above example
1.5	1.5
-32.178E+02	-3217.8
0.25E8	25 million
0.E25	0.
2.	2.
5.0	5.0

Invalid file expressions of Real	Problem
1.2E3.	Decimal point not allowed in exponent
1E05	Decimal point required in mantissa

1,000.00	*Comma* not allowed
3.E	Digit(s) required in exponent
.5	At least one digit must precede the decimal point
1	Decimal point required in mantissa

7.3.3 String

A string shall be encoded as an *apostrophe* "'", followed by zero or more 8-bit bytes, and ended by an *apostrophe* "'". The null string (string of length zero) shall be encoded by two consecutive *apostrophes* "''". Within a string, a single *apostrophe* shall be encoded as two consecutive *apostrophes*. Within a string, a single *reverse solidus* "\" shall be encoded as two *reverse solidi* "\\". The 8-bit bytes allowed within a string are the decimal equivalents 32 through 126 (inclusive) of ISO 8859-1 that define the graphic characters of the basic alphabet.

NOTE - Table D.1 gives the correspondence between the 8-bit bytes and their graphic representation in ISO 8859-1. The *quotation mark* does not need to be doubled when appearing in a string. It appears doubled in table 1 because it is a meta-character of the WSN (See annex B).

Additional characters shall be encoded using the hexadecimal digits (see HEX in table 2) as defined in 7.3.3.1 and 7.3.3.2. The WSN of control directives for encoding strings is given in table 4.

7.3.3.1 Encoding the full alphabet of ISO 8859 within a string

Each part of ISO 8859 includes the basic alphabet as part of its specification; the various parts differ in the extended character set. To include the extended portion of the alphabet in a string requires the use of control directives. One control directive is used within a string to allow bytes in the basic alphabet, G(02/00) through G(07/14), to encode the portions of the alphabet not included in the basic alphabet. The extension of the basic alphabet corresponds to ISO 8859, G(10/00) to G(15/14). G(x/y) is a notation for the character in position (16 times x) + y in the code table in ISO 8859.

The control directive *reverse solidus capital letter S reverse solidus* "\S\" shall be used to indicate that the most significant bit of the next 8-bit byte in a string is interpreted as a binary one. The control directive *reverse solidus capital letter P* UPPER *reverse solidus* shall indicate that, for this string only, the subsequent *reverse solidus capital letter S reverse solidus* control directives shall be interpreted as referring to the extended alphabet defined in that part of ISO 8859 indicated by the value of UPPER. The *capital letter* referred to shall be one of the following letters : "A", "B", "C", "D", "E", "F", "G", "H", "I". In this context, the *capital letter A* identifies ISO 8859-1; *capital letter B* identifies ISO 8859-2, etc. If this control directive does not appear within a string, the value "A" shall be assumed; i.e., the extended alphabet shall be that specified in ISO 8859-1.

Table 4 - String control directives

```
CONTROL_DIRECTIVE = PAGE ¦ ALPHABET ¦ EXTENDED2
                    ¦ EXTENDED4 ¦ ARBITRARY .

PAGE = REVERSE_SOLIDUS "S" REVERSE SOLIDUS CHARACTER .

ALPHABET = REVERSE_SOLIDUS "P" UPPER REVERSE_SOLIDUS .

EXTENDED2 = REVERSE_SOLIDUS "X2" REVERSE_SOLIDUS
            HEX_TWO { HEX_TWO } END_EXTENDED .

EXTENDED4 = REVERSE_SOLIDUS "X4" REVERSE_SOLIDUS
            HEX_FOUR { HEX_FOUR } END_EXTENDED .

END_EXTENDED = REVERSE_SOLIDUS "X0" REVERSE_SOLIDUS .

ARBITRARY = REVERSE_SOLIDUS "X" REVERSE_SOLIDUS HEX_ONE .

HEX_ONE = HEX HEX .

HEX_TWO = HEX_ONE HEX_ONE .

HEX_FOUR = HEX_TWO HEX_TWO .
```

EXAMPLE 3 -

String as stored	Effective contents	Comments
'CAT'	CAT	
'Don''t'	Don't	
''''	'	
''		string of length zero
'\S\Drger'	Ärger	
'h\S\vtel'	hôtel	
'\PE\\S*\S\U\S\b'	Нет	Cyrillic, 'Nyet'

7.3.3.2 Encoding the character sets of ISO 10646 within a string

ISO 10303-11, 8.1.6, specifies that any character from ISO 10646 may occur in a string. This part of ISO 10303 specifies three control directives that allow encoding of characters from ISO 10646.

ISO 10646 defines a canonical form that uses four octets to represent any character in the full coding space. These characters specify the group, plane, row, and cell respectively. In addition, ISO 10646 defines a Basic Multilingual Plane (BMP), representing plane 00 of group 00 of the full coding space. The characters in the BMP are represented by two octets, specifying row and cell.

NOTE - The Basic Multilingual Plane includes characters in general use in alphabetic, syllabic, and idiographic scripts together with various symbols and digits.

The control directive *reverse solidus capital letter X digit two reverse solidus* "\X2\"shall be used to indicate that the following sequence of multiples of four hexadecimal characters shall be interpreted as encoding the two-octet representation of characters from the BMP in ISO 10646. The encoding in a string in the exchange structure shall be as follows:

— Each character in the representation of 10646 to be encoded shall be converted to two 8-bit bytes as specified in ISO 10646;

— Each of the two resulting 8-bit bytes shall be encoded as two hexadecimal characters in the basic alphabet corresponding to the graphic representation of the hexadecimal digit.

EXAMPLE 4 - The Latin *capital letter B* is converted to the hexadecimal value '0042' hex by table 1 in ISO 10646. The hexadecimal digits corresponding to this value are 0, 0, 4, and 2. The encoding in the exchange structure using the basic alphabet consists of the four characters 0042.

The control directive *reverse solidus capital letter X digit four reverse solidus* "\X4\" shall be used to indicate that the following sequence of multiples of eight hexadecimal characters shall be interpreted as encoding the four-octet representation of characters from the full coding space of ISO 10646. The encoding in a string in the exchange structure shall be as follows:

— Each character in the representation of 10646 to be encoded shall be converted to four 8-bit bytes as specified in ISO 10646;

— Each of the four resulting 8-bit bytes shall be encoded as two hexadecimal characters in the basic alphabet corresponding to the graphic representation of the hexadecimal digit.

EXAMPLE 5 - The Latin *capital letter B* is converted to the hexadecimal characters 00000042 by table 1 in ISO 10646. The hexadecimal digits are 0, 0, 0, 0, 0, 0, 4, and 2. The encoding in the exchange structure using the basic alphabet consists of the eight characters 00000042.

The control directive *reverse solidus capital letter X digit zero reverse solidus* "\X0\" shall be used to indicate the end of encoding of ISO 10646 characters in a string and a return to direct encoding in the basic alphabet.

7.3.3.3 Encoding a single 8-bit byte in a string

An 8-bit byte with a value between 0 and 255 may be encoded in a string. The control directive *reverse solidus capital letter X reverse solidus* "\X\" shall be used in a string to indicate that the following two hexadecimal characters shall be interpreted as an 8-bit byte.

7.3.3.4 Maximum string length

The maximum length of a string as stored in an exchange structure is 32769 8-bit bytes, including the beginning and ending *apostrophes*. If embedded *quotation marks*, *reverse solidi*, *apostrophes*, print control directives (clause 12) or characters encoded according to 7.3.3.1, 7.3.3.2, or 7.3.3.3 are included in the string as stored, the maximum length of the effective contents of the string will be less than 32767 graphic characters. The effective contents is the sequence of graphic characters after these encoding conventions have been resolved.

7.3.4 Entity instance names

An entity instance name shall be encoded as a *number sign*, "#", followed by an unsigned integer. The integer shall consist of any combination of one or more digits. At least one digit shall not be "0". Leading zeros in entity instance names are not significant.

The WSN for the entity instance names is given in table 3 in the production ENTITY_INSTANCE_NAME.

EXAMPLE 6 -

Valid name expressions	Meaning
#12	Names or refers to entity instance with identifier 12
#023	Names or refers to entity instance with identifier 23

Invalid name expressions	Problem
#+23	Contains '+' sign
#00.1	Contains decimal point
74	Does not begin with a *number sign*
#439A6	Contains alphabetic character

Entity instance names are used as references to other entity instances if they appear inside the attribute list of an entity instance. Both forward and backward references are permitted.

7.3.5 Enumeration values

An enumeration value shall be encoded as a sequence of *capital letters* or *digits* beginning with an *capital letter* delimited by a *full stops*. The meaning of a given enumeration value is determined by the EXPRESS schema and its associated definitions from the enumeration type declarations.

EXAMPLE 7 -

Valid enumeration expression	Meaning
.STEEL.	Indicates a value of STEEL

Invalid enumeration expressions	Problem
.RED	Missing ending *full stop*
.123.	Does not start with an alphabetic character.

7.3.6 Binary

A binary is a sequence of bits (0 or 1). A binary shall be encoded as determined by the following procedure.

— Count the number of bits in the sequence. Call the result p;

— Determine a number n, $0 \leq n \leq 3$, such that $k=p+n$ is a multiple of four;

— Left fill the binary with n zero bits. Divide the sequence into groups of four bits.

— Precede the sequence with the 4-bit representation of n;

— If the decimal equivalent of a 4-bit group is 9 or less, add 48 to that decimal value to create an 8-bit byte; if the decimal equivalent of the 4-bit group is greater than 9, add 55 to that decimal value to create an 8-bit byte.

NOTE - This is a binary to hexadecimal conversion.

— The encoding of a binary consists of k/4+1 hexadecimal digits. The first digit is the value of n. This is followed by the hexadecimal digits representing the binary;

— Delimit the encoded binary with *quotation marks* """.

EXAMPLE 8 -

Binary value	Representation
'null' or 'empty'	"0"
0	"30"
1	"31"
111011	"23B"
100100101010	"092A"

8 Structured data types

The only structured data type that appears in the exchange structure is LIST as defined in table 3. A LIST is a collection of instances of simple or structured data types. In the exchange structure, a LIST begins with a *left parenthesis* "(" and ends with a matching *right parenthesis* ")". Instances are separated by *commas*. LISTs can be nested to any depth.

EXAMPLE 9 -

Structured data type	Representation
List of Integer	(0,1,2,3,7,2,4)
List of String	('CAT', 'HELLO')
List of List of Real	((0.0, 1.0, 2.0), (3.0, 4.0, 5.0))
List of List of Real	((0.0, 1.0, 2.0), ( ))

In the last List of List of Real, the second embedded list is empty.

9 Header section

The header section contains information that is applicable to the entire exchange structure. This section shall be present in every exchange structure. The section shall begin with the special token "HEADER;" and shall terminate with the special token "ENDSEC;".

NOTE - Annex F presents an example of a header section within an exchange structure.

9.1 Header section entities

Three header section entities are specified and one instance of each is required to occur in the header section of every exchange structure. The header section entities are **file_description**, **file_name**, and **file_schema**, and they shall appear in that order. If instances of user-defined header section entities are present, they shall appear after the required header section entity instances and may appear in arbitrary order. The syntax of the header section entity instances is given in table 3 in WSN. Each entity name shall map to the KEYWORD of the HEADER_ENTITY production. Clause 11 provides mapping of simple and aggregate data types to the PARAMETER_LIST for the attribute values of these entity instances.

9.2 Header section schema

This clause specifies header section entities and types that appear in the header section of the exchange structure. The header section entities are specified in EXPRESS.

EXPRESS Specification:

```
*)
SCHEMA header_section_schema;
(*
```

This schema specifies the header section entities that are specific to the process of transferring product data using the exchange structure.

9.2.1 file_description

The **file_description** specifies the version of this part of ISO 10303 used to create the exchange structure as well as its contents.

EXPRESS Specification:

```
*)
ENTITY file_description;
   description : LIST [1:?] OF STRING (256) ;
   implementation_level : STRING (256) ;
END_ENTITY;
(*
```

Attribute Descriptions:

description: an informal description of the contents of this exchange structure.

implementation_level: an identification of the required capability of the system to process the data in this exchange structure. The value for this string for exchange structures adhering to conformance class 1 of this part of ISO 10303 shall be the graphic character "1". The value for this string for exchange structures adhering to conformance class 2 of this part of ISO 10303 shall be the graphic character "2". Conformance classes 1 and 2 are defined in 11.2.5.

NOTE - This last attribute has been provided for upward compatibility. Future versions of this part of ISO 10303 may specify additional values for this attribute.

9.2.2 file_name

The **file_name** provides human readable information about the exchange structure. With the exception of the time_stamp attribute, the contents of the attributes of this entity are not defined by this part of ISO 10303.

NOTE - The attributes not defined in this part of ISO 10303 are defined in application protocols.

EXPRESS Specification:

```
*)
ENTITY file_name;
  name : STRING (256) ;
  time_stamp : STRING (256) ;
  author : LIST [ 1 : ? ] OF STRING (256) ;
  organization : LIST[ 1 : ? ] OF STRING (256) ;
  preprocessor_version : STRING (256) ;
  originating_system : STRING (256) ;
  authorization : STRING (256) ;
END_ENTITY;
(*
```

Attribute Descriptions:

name: the string of graphic characters used to name this particular instance of an exchange structure.

NOTE - The name is intended to be used as human to human communication between sender and receiver.

time_stamp: the date and time specifying when the exchange structure was created. The contents of the string shall correspond to the extended format for the complete calendar date as specified in 5.2.1.1 of ISO 8601 concatenated to the extended format for the time of the day as specified either in 5.3.1.1 or in 5.3.3 of ISO 8601. The date and time shall be separated by the *capital letter T* as specified in 5.4.1 of ISO 8601. The alternate formats of 5.3.1.1 and 5.3.3 permit the optional inclusion of a time zone specifier.

author: the name and mailing address of the person responsible for creating the exchange structure

organization: the group or organization with whom the author is associated.

preprocessor_version: the system used to create the exchange structure, including the system product name and version.

originating_system: the system from which the data in this exchange structure originated.

authorization: the name and mailing address of the person who authorized the sending of the exchange structure.

EXAMPLE 10 -

Time stamp element	Complete extended format
Calendar Date 12 April 1993	1993-04-12
Time of the Day 27 minutes 46 seconds past 15 hours	15:27:46
Time Zone 5 hours west of Greenwich	Time Zone field is optional -05:00
Above date and time encoded within the Time_Stamp field	1993-04-12T15:27:46-05:00

9.2.3 file_schema

The **file_schema** entity identifies the EXPRESS schemas that specify the entity instances in the data section. The attribute **schema_identifiers** shall consist of a list of strings, each of which shall contain the name of the schema optionally followed by the object identifier assigned to that schema.

If the name of a schema contains *small letters*, such *small letters* shall be converted to the corresponding *capital letters*. Only *capital letters* shall occur in strings of the **schema_name**.

If an object identifier is provided, it shall be encoded as specified in ISO/IEC 8824-1. The use of object identifiers within this International Standard is described in clause 3 of ISO 10303-1. When available, the use of the object identifier is recommended as it provides unambiguous identification of the schema.

The **schema_identifiers** attribute shall contain exactly one **schema_name**.

EXPRESS Specification:

```
*)
ENTITY file_schema;
  schema_identifiers : LIST [1:1] OF schema_name;
END_ENTITY;

TYPE schema_name = STRING(1024);
END_TYPE;
(*
```

Attribute Descriptions:

schema_identifiers: the schemas that specify the entity instances in the data section.

```
*)
END_SCHEMA;
(*
```

9.3 User-defined header section entities

User-defined header section entity instances may be placed in the header section with specific restrictions as listed below:

a) User-defined header section entity instances shall conform to the same syntax of all header section entity instances with the additional requirement that the first character of the keyword shall be an *exclamation mark* "!".

b) User-defined header section entity instances shall not use any simple data types or structured data types beyond those specified in this part of ISO 10303.

EXAMPLE 11 -

```
HEADER;
    .
    .
FILE_SCHEMA(('GEOMETRY'));
!A_SPECIAL_ENTITY ('ABC',123); -------> USER DEFINED ENTITY
    .
    .
ENDSEC;
```

10 Data section

The data section contains the product data to be transferred by the exchange structure. This section shall be present in every exchange structure. The data section contains instances of entities that correspond to the EXPRESS schema governing the exchange structure as specified in the header section. The data section shall begin with the special token "DATA;" and shall be followed by entity instances. The data section shall be terminated with the special token "ENDSEC;".

NOTE - Annex F presents a complete example of a data section within an exchange structure.

10.1 Data section entities instances

The syntax for the data section entity instances is specified in table 3. Each entity instance shall be represented at most once in the data section; distinct entity instances shall have distinct entity instance names. The entity instances need not be ordered in the exchange structure. An entity instance may be referenced before its name occurs as an identifier in the exchange structure.

In the production ENTITY_INSTANCE in table 3, the first alternative is for entity instances without supertypes or for entity instances with supertypes where internal mapping is applied. The second alternative is for entity instances with supertypes where external mapping has been applied. Internal and external mappings are specified in 11.2.5.

10.2 Optional values

ISO 10303-11 allows attributes and elements of arrays to be specified as optional. When a value is not provided for an optional attribute or for an optional element of an array, the attribute value shall be encoded as the *dollar sign* "$".

EXAMPLE 12 -

```
#2 = NODE(1.0, $, 2.0,'ABC');
```

This entity has an optional second attribute value that is not provided in this instance.

10.3 Scope structure

The SCOPE structure within entity instances is a mechanism for providing a scope of reference and for defining existence relationships among entites.

10.3.1 Syntax

The syntax of the SCOPE structure is specified in table 3. The SCOPE structure is optional for all entity instances. If the SCOPE structure is used in a SIMPLE_RECORD, the SCOPE structure shall appear between the *equals sign* "=" and the keyword of the SIMPLE_RECORD. If the SCOPE structure is used in a SUBSUPER_RECORD, the SCOPE structure shall appear between the *equals sign* and the starting *left parenthesis*.

The SCOPE structure shall be started by the special token "&SCOPE". At least one entity instance shall follow. The structure shall be terminated by the special token "ENDSCOPE". Entity instances within a SCOPE structure may themselves have SCOPE structures.

10.3.2 References to entity instances within a SCOPE structure

An Entity instance defined within a SCOPE structure may be referenced by another entity instance within the same SCOPE structure or as an attribute of the entity instance in whose immediate SCOPE structure they both exist. The availability of an entity instance referenced in a SCOPE structure S may be extended to the SCOPE structure that properly contains S by including its entity instance name in the export list of S. The ability to be referenced may be further extended by progressively including the entity instance name into wider and wider SCOPE structures. An entity instance name exported from the outermost SCOPE structure may be referenced by all other entity instances in the exchange structure.

10.3.3 Behaviour

The EXPRESS schema describing a particular application area specifies entities and, by having as attributes references to other entities, also establishes the existence of relations between entities. In addition to these specific references, the exchange structure can convey information concerning the creation and deletion of entity instances. If an entity instance B can exist only when another entity instance A exists, B is considered to be in the scope of A.

NOTE - If A references an entity that references B, then A is considered to reference B indirectly. By extension, if A references an entity that references B indirectly, then A references B indirectly also. Existence dependence is independent of any reference between A and B.

This means that if a request for the deletion of A implies that for the data to remain meaningful, the deletion of B is either simultaneous or completed prior to the deletion of A and if this is true irrespective of:

— the type of entities A and B;

— whether A references B directly or indirectly, or A references B at all;

— whether B references A directly or indirectly, or B references A at all;

then:

— The existence of B depends on the existence of A,

or, equivalently,

— B is considered to be in the scope of A.

The creator of an exchange structure can convey the existence dependency of entity instances. The SCOPE structure is introduced to make it possible to collect all the entities that are existence-dependent on the same entity.

EXAMPLES

13 -
```
#1  = &SCOPE
#2  = POINT (0.0,0.0,0.0);
#3  = POINT (0.0,1.0,0.0);
#4  = POINT (3.0,1.0,0.0);
#5  = LINE (#2,#3);
#6  = LINE (#3,#4);
#7  = LINE (#4,#2);
  ENDSCOPE
  TRIANG(#5,#6,#7);
```

The entity instance #1 of type TRIANG is defined by three LINE entities. Because the LINE entities are within the scope of the TRIANG entity, the assertion is being made that they are existence-dependent upon the TRIANG entity. The POINT entity instances that define the lines are also dependent upon the TRIANG for their existence. The POINT instances are available to be referenced by the LINE entities that are at the same scope level.

14 -
```
#1  = &SCOPE
#12 = PART ('PART 74A',$,$,$);
#13 = PART ('PART 68B',$,$,$);
#14 = PART ('PART 12C',$,$,$);
    .
    .
    .
  ENDSCOPE /#13, #14/
  ASMBLY ((#12, #13),$,$,$);
#15 = APROVL ((#13), 'JOHN SMITH APPROVAL AUTHORITY');
```

The part entity instances #13 and #14 are exported from the SCOPE structure of the ASMBLY entity through the EXPORT_LIST. Following the attributes of the ASMBLY entity, an APROVL entity instance occurs that references the PART entity with entity instance #13. In this case, it is only that particular PART entity instance that is approved, while at the same time maintaining the existence dependency of that PART entity instance upon its parent ASMBLY.

10.4 Data section user-defined entity instances

A user-defined entity instance is an entity that is not part of the EXPRESS schema specified in the header section. User-defined entity instances shall conform to the same syntax of all data section entity instances except that the USER_DEFINED_KEYWORD choice shall be used in the SIMPLE_RECORD that is part of this definition. The meaning of a user-defined entity instance, and the number, data types and meanings of its attributes, is a matter of agreement between the partners using the exchange structure.

EXAMPLE 15 -

```
DATA;
    .
    .
    .
#1=PT(1.0,2.0,3.0); ------> STANDARD ENTITY
#2=PT(1.0,2.0,5.0);
    .
    .
    .
#12=!MYCURVE(0.0,0.0,0.0,1.0,$,$,$); ----> USER DEFINED ENTITY
    .
    .
    .
ENDSEC;
```

11 Mapping from EXPRESS to the exchange structure

This clause describes how instances of EXPRESS language elements are mapped to the exchange structure. The EXPRESS language is specified in ISO 10303-11.

The EXPRESS language includes declarative statements, executable statements, and algorithms. Only certain declarative statements are actually mapped to the exchange structure. The executable statements and algorithms of EXPRESS are not mapped to the exchange structure.

NOTE - Table 5 - provides a reference of the mappings between EXPRESS and the exchange structure. Specifically, these are the mappings from the elements of EXPRESS onto their counterparts in the data section of the exchange structure.

11.1 Mapping of EXPRESS data types

This clause specifies the mapping from the EXPRESS elements that are data types to the exchange structure.

Table 5 - Quick reference mapping table

EXPRESS element	mapped onto:
ARRAY	list
BAG	list
BOOLEAN	boolean
BINARY	binary
CONSTANT	NO INSTANTIATION
DERIVED ATTRIBUTE	NO INSTANTIATION
ENTITY	entity
ENTITY AS ATTRIBUTE	entity instance name
ENTITY AS SUPERTYPE	See 11.2.5
ENUMERATION	enumeration
FUNCTION	NO INSTANTIATION
INTEGER	integer
LIST	list
LOGICAL	enumeration
PROCEDURE	NO INSTANTIATION
REAL	real
REMARKS	NO INSTANTIATION
RULE	NO INSTANTIATION
SCHEMA	NO INSTANTIATION
SELECT	See 11.1.8
SET	list
STRING	string
TYPE	See 11.1.6
WHERE RULES	NO INSTANTIATION

11.1.1 Mapping of EXPRESS simple data types

11.1.1.1 Integer

The EXPRESS element of INTEGER shall be mapped to the exchange structure as an integer data type. 7.3.1 describes the composition of a integer data type.

11.1.1.2 String

The EXPRESS element of STRING shall be mapped to the exchange structure as a string data type. 7.3.3 describes the composition of a string data type.

11.1.1.3 Boolean

The EXPRESS element of BOOLEAN shall be mapped to the exchange structure as an enumeration data type. 7.3.5 describes the composition of a enumeration data type. The EXPRESS element of BOOLEAN shall be treated as a predefined enumerated data type with a value encoded by the graphic characters "T" or "F". These values shall correspond to true and false respectively.

11.1.1.4 Logical

The EXPRESS element of LOGICAL shall be mapped to the exchange structure as an enumeration data type. 7.3.5 describes the composition of a enumeration data type. The EXPRESS element of LOGICAL shall be treated as a predefined enumerated data type with a value encoded by the graphic characters "T", "F" or "U". These values shall correspond to true, false, and unknown respectively.

11.1.1.5 Real

The EXPRESS element of REAL shall be mapped to the exchange structure as a real data type. 7.3.2 describes the composition of a real data type.

EXAMPLE 16 - Entity definition in EXPRESS:

```
ENTITY widget;
  i1: INTEGER;     ----------->  A
  i2: INTEGER;     ----------->  B
  s1: STRING(3);   ----------->  C
  s2: STRING;      ----------->  D
  l : LOGICAL;     ----------->  E
  b : BOOLEAN;     ----------->  F
  r1: REAL(4);     ----------->  G
  r2: REAL;        ----------->  H
END_ENTITY;
```

Sample entity instance in Data section:

```
#2=WIDGET(99, 99999, 'ABC', 'ABCDEFG', .T., .F., 9.000, 1.2345);
          ^     ^      ^         ^      ^    ^     ^       ^
          |     |      |         |      |    |     |       |
          A     B      C         D      E    F     G       H
```

A: i1 has a value of 99 in this entity instance.

B: i2 has a value of 99999 in this entity instance.

C: s1 has a value of 'ABC' in this entity instance. This value falls within the range (3 characters) specified for this attribute.

D: s2 has a value of 'ABCDEFG' in this entity instance.

E: l has a value of TRUE in this entity instance.

F: b has a value of FALSE in this entity instance.

G: r1 has a value of 9.000 in this entity instance. This value falls within the precision (the number contains four significant digits) specified for this attribute.

H: r2 has a value of 1.2345 in this entity instance.

11.1.1.6 Binary

The EXPRESS element of BINARY shall be mapped to the exchange structure as a binary data type. 7.3.6 describes the composition of a binary data type.

EXAMPLE 17 - Entity definition in EXPRESS:

```
ENTITY picture;
  bn :  BINARY;
END_ENTITY;
```

Sample entity instance in data section:

```
          #4 = PICTURE("1556FB0");
                         ^
                         |
                         A
```

A: bn has an encoding of "1556FB0" in this instance, corresponding with the bit sequence 101 0101 0110 1111 1011 0000.

11.1.2 List

The EXPRESS element of LIST shall be mapped to the exchange structure as a list data type. Clause 8 describes the composition of a list data type. If an entire list is optional and does not exist, the list shall be encoded by a *dollar sign* "$". If the list is empty, the list shall be encoded as a *left parenthesis* "(" followed by a *right parenthesis* ")".

EXAMPLE 18 - Entity definition in EXPRESS:

```
ENTITY widget;
  attribute1:  LIST [0 : ?] OF INTEGER; ----------> A
  attribute2:  LIST [1 : ?] OF INTEGER; ----------> B
  attribute3:  OPTIONAL LIST [1 : ?] OF INTEGER; --> C
  attribute4:  REAL; -----------------------------> D
END_ENTITY;
```

Sample entity instance in data section:

```
          #4 = WIDGET((), (1,2,4), $, 2.56);
                      ^     ^    ^   ^
                      |     |    |   |
                      A     B    C   D
```

A: attribute1 is an empty list (list with zero elements).

B: attribute2 contains three elements in this instance.

C: attribute3 does not have a value in this instance.

D: attribute4 has a value of 2.56 in this instance.

11.1.3 Array

The EXPRESS element of ARRAY shall be mapped to the exchange structure as a list data type. Clause 8 describes the composition of a list data type. If an EXPRESS attribute is a multidimensional array the attribute shall be encoded as a list of lists, nested as deeply as there are dimensions. In constructing such lists, the inner-most list, the list containing only primitive data elements, shall correspond to the right-most ARRAY specifier in the EXPRESS statement defining the entity. The ordering of the elements within the encoding shall be that all the elements of the inner-most list are encoded for each element of the next outer list. This order means that the right-most index in each list shall vary first.

EXAMPLES

19 - Entity definition in EXPRESS:

```
X : ARRAY[1:5] OF ARRAY[100:102] OF INTEGER
```

This is encoded in the following order:

```
( (X [1,100], X [1,101], X [1,102] ),
  (X [2,100], X [2,101], X [2,102] ),
  (X [3,100], X [3,101], X [3,102] ),
  (X [4,100], X [4,101], X [4,102] ),
  (X [5,100], X [5,101], X [5,102] )  )
```

20 - Entity definition in EXPRESS:

```
ENTITY widget;
  attribute1:  ARRAY [-1 : 3] OF INTEGER;     ---------------------> A
  attribute2:  ARRAY [1 : 5] OF OPTIONAL INTEGER;      --------------> B
  attribute3:  ARRAY [1 : 2] OF ARRAY [1 : 3] OF INTEGER;     ------> C
END_ENTITY;
```

Sample entity instance in data section:

```
#30 = WIDGET((1,2,3,4,5) , (1,2,3,$,5) , ((1,2,3),(4,5,6)));
                  ^             ^                ^
                  |             |                |
                  A             B                C
```

A: attribute1 contains the following values:

```
attribute1 [-1]  = 1
attribute1 [0]   = 2
attribute1 [1]   = 3
attribute1 [2]   = 4
attribute1 [3]   = 5
```

B: attribute2 contains the following values:

```
attribute2 [1] = 1
attribute2 [2] = 2
attribute2 [3] = 3
attribute2 [4] = MISSING
attribute2 [5] = 5
```

The significance of a missing value is defined within the EXPRESS schema.

C: attribute3 contains the following values:

```
attribute3 [1,1] = 1
attribute3 [1,2] = 2
attribute3 [1,3] = 3
attribute3 [2,1] = 4
attribute3 [2,2] = 5
attribute3 [2,3] = 6
```

11.1.4 Set

The EXPRESS element of SET shall be mapped to the exchange structure as a list data type. Clause 8 describes the composition of a list data type.

EXAMPLE 21 - Entity definition in EXPRESS:

```
ENTITY widget;
  a_number: SET OF INTEGER;
END_ENTITY;
```

Sample entity instance in data section:

```
#2 = WIDGET((0,1,2)); --------> A
#3 = WIDGET((0,$,2)); --------> B
#4 = WIDGET((0,0,2)); --------> C
```

A: The attribute a_number was defined by the set numbers 0, 1, 2 in this instance.

B: Syntactically the instance is correct. However, the instance is incorrect with respect to the definition of a SET in EXPRESS because a SET is not allowed to have missing members.

C: Syntactically the instance is correct. However, the instance is incorrect with respect to the definition of a SET in EXPRESS because a SET is not allowed to have duplicate members.

11.1.5 Bag

The EXPRESS element of BAG shall be mapped to the exchange structure as a list data type. Clause 8 describes the composition of a list data type.

EXAMPLE 22 - Entity definition in EXPRESS:

```
ENTITY widget;
  a_numbers: BAG OF INTEGER;
END_ENTITY;
```

Sample entity instance in data section:

```
#2 = WIDGET((0, 1, 1, 2));  --------> A
#3 = WIDGET((0, $, 2));     --------> B
```

A: The attribute a_numbers was defined by the collection of numbers 0, 1, 1, 2 in this instance.

B: Syntactically, the instance is correct. However, the instance is incorrect with respect to the definition of BAG in EXPRESS because a BAG is not allowed to have missing members.

11.1.6 Simple defined types

An EXPRESS simple defined type is a TYPE declaration that is not an ENUMERATION or a SELECT. An EXPRESS simple defined type shall be mapped to the exchange structure as the data type used in its definition.

EXAMPLE 23 - Entity definition in EXPRESS:

```
TYPE
  type1 =  INTEGER;
END_TYPE;

TYPE
  type2 = LIST [1 : 2] of REAL;
END_TYPE;

ENTITY widget;
  attribute1:  LOGICAL;  ------------>  A
  attribute2:  TYPE1;    ------------>  B
  attribute3:  TYPE2;    ------------>  C
END_ENTITY;
```

Sample entity instance in data section:

```
#4 = WIDGET( .T., 256, (1.0,0.0));

             ^    ^      ^
             |    |      |
             A    B      C
```

A: The value of the attribute attribute1; in this instance, TRUE.

B: Type1 is an integer type and, therefore, the value 256 is valid.

C: Type2 is a list and, therefore, a list with 2 REAL elements is valid.

11.1.7 Enumeration

The EXPRESS element of ENUMERATION shall be mapped to the exchange structure as an enumeration data type. 7.3.5 describes the composition of a enumeration data type. The actual value in an instance of the ENUMERATION shall be one of the enumerated values in the EXPRESS schema in *capital letters* and delimited by *full stops* "." as defined in the ENUMERATION production of table 2.

EXAMPLE 24 - Entity definition in EXPRESS:

```
TYPE
  primary_colour = ENUMERATION OF (red, green, blue);
END_TYPE;

ENTITY widget;
  p_colour: primary_colour;      ---------------> A
END_ENTITY;
```

Sample entity Instance in data section:

```
#2 = WIDGET(.RED.);
              ^
              |
              A
```

A: The value of the attribute p_colour in this entity instance is RED.

11.1.8 Select data types

The EXPRESS element of SELECT shall be mapped to the data section as provided in this clause. The actual value in an instance of the SELECT shall be a value corresponding to one of the data types in the select list of the SELECT type.

The SELECT data type can be represented as a directed acyclic graph with a root node that represents the SELECT data type itself. The following algorithm is applied to develop the graph:

— Each named-type that is a SELECT data type is connected by outward arcs to nodes representing each of the named-types in the select list in its type-declaration.

— Each named-type that is an entity-type is considered a terminal node.

— Each named-type that is not an entity-type or a SELECT data type is connected by a single outward arc to a node representing the underlying type in its type declaration.

— Each node that does not represent a named-type is considered a terminal node.

— Every type is represented only once in the graph. When the type-declarations of two or more named-types refer to the same underlying type, the corresponding arcs are both connected (inward) to the same node in the graph.

— All unnamed aggregate data types are represented by a single terminal node called AGGREGATE (that appears only once in the graph).

— All unnamed ENUMERATION data types are represented by a single terminal node called ENUMERATION (that appears only once in the graph).

— Each unnamed primitive type is represented by a single terminal node (that appears only once in the graph) named with the corresponding Express keyword.

This process is continued until every non-terminal named-type in the graph is connected by one or more arcs to its underlying types.

NOTE - Entity types are also only represented once in the graph. When an entity type is referred to by two or more select types in the graph, the entity type is represented by a single node with multiple arcs connecting it to the nodes representing the different select types.

EXAMPLE 25 - Sample graphical representation (mapping) of a SELECT data type with simple types

```
Refer to Figure 1.

TYPE
xxx = STRING;
END_TYPE;

TYPE
yyy = STRING;
END_TYPE;

ENTITY widget;
   attr1: REAL;
END_ENTITY;

TYPE
aaa = SELECT (widget, xxx, yyy);
END_TYPE;
```

As shown, each named type is connected by an arc to a node representing the type in its definition.

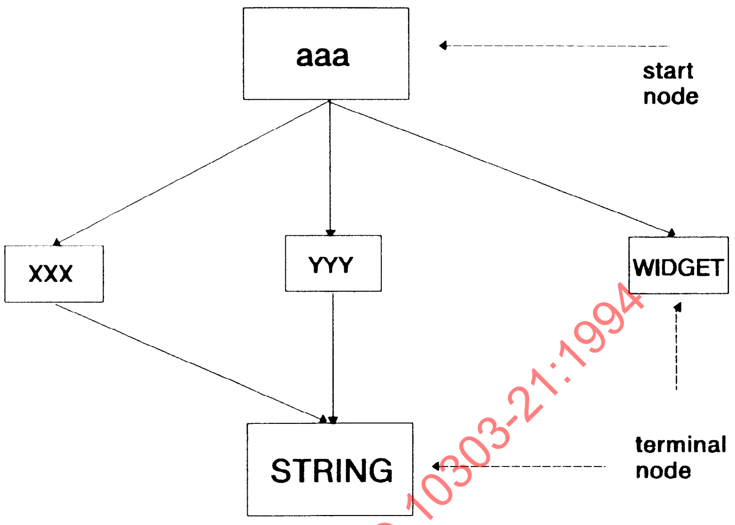

Figure 1 Example 25

The EXPRESS element of SELECT shall be mapped to the data section as follows:

— if the path from the SELECT (root) to the terminal (leaf) node contains no nodes that have more than one incoming arc (i.e., the graph is a tree), the PARAMETER shall be encoded as a value of the terminal type;

— otherwise, the PARAMETER shall be encoded as the composition of TYPED_PARAMETERs, one for each node in the path after the root node. The outermost (leftmost) TYPED_PARAMETER corresponds to the first such node encounterd on the path, the innermost TYPE_PARAMETER corresponds to the last such node. The PARAMETER of the innermost node shall be encoded as a value of the terminal type;

EXAMPLE 26 - Entity definition in EXPRESS:

```
TYPE
 size= SELECT (area, radius);
END_TYPE;

TYPE
 area = REAL;
END_TYPE;

TYPE
 radius = REAL;
END_TYPE;

ENTITY circle;
 howbig : size;
END_ENTITY;
```

Sample entity instance in data section:

```
           #15 = CIRCLE(AREA(3.3));
                        ^
                        |
                        A
```

A: The attribute howbig in this entity instance has a type of AREA and a value of 3.3.

EXAMPLE 27 - Entity definition in EXPRESS:

```
TYPE
  p = SELECT(r,s);
END_TYPE;

TYPE
  q = SELECT(s,t);
END_TYPE;

TYPE
  r = real;
END_TYPE;

TYPE
  s = SELECT(u,v);
END_TYPE;

TYPE
  t = SELECT(v,w);
END_TYPE;

TYPE
  u = REAL;
END_TYPE;

TYPE
  v = REAL;
END_TYPE;

TYPE
  w = REAL;
END_TYPE;

ENTITY a;
  attr1 : p;
  attr2 : q;
END_ENTITY;
```

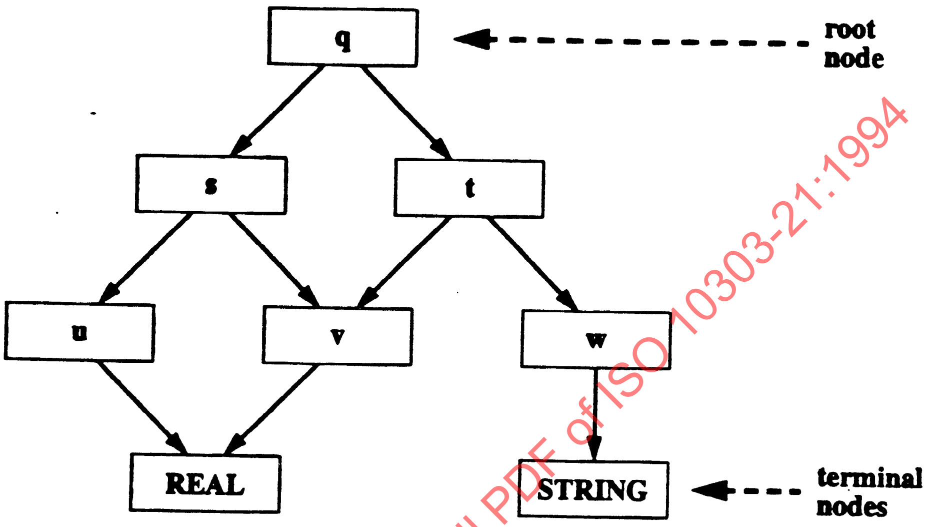

Figure 2 Example 27
Graph corresponding to SELECT type q

Sample entity instance in data section:

```
#22 = A( S(U(8.8)), S(U(2.345)) );
         ^          ^
         |          |
         A          B
```

A: The attribute attr1 in this entity instance has a type of p-s-u and a value of 8.8. The graph for p is not a tree.

B: The attribute attr2 in this entity has a type of q-s-u and a value of 2.345. Note that there is a difference between the type q-s-u and p-s-u as this represents another value with another meaning for the attribute.

11.2 Mapping of EXPRESS entity data types

The EXPRESS element of ENTITY shall be mapped to the data section as a KEYWORD. If the document that defines the schema whose instances are the subject of the data section also defines a set of short names for each of the entity data types within that schema, these short names shall be used as the encoding of the entity data type names. Otherwise, the encoding of the entity data type names shall be the entity data type names themselves. In either case, any *small letters* shall be converted to their corresponding *capital letters*, i.e., the encoding shall not contain any *small letters*.

The complexity of entity definitions in EXPRESS requires a number of mapping rules for encoding instances of entities. The mapping rules address the following entity definitions:

— Entities with explicit attributes;

— Entities with explicit optional attributes;

— Entities with derived attributes;

— Entities with other entities as attributes;

— Entities defined as subtypes of other entities;

— Entities with attributes redefined as an effect of a DERIVE;

— Entities with attributes redefined as an effect of a subtype declaration;

— Entities with WHERE rules;

— Entities with INVERSE attributes.

NOTE - A given entity may require several of these rules to be applied.

11.2.1 Entity with explicit attributes

Entities with explicit attributes shall be mapped directly to the entity instances in the exchange structure. The attribute order in the exchange structure shall be the same order as in the EXPRESS entity definition. The first entity attribute following the EXPRESS entity data type name shall be the first entity attribute; the second entity attribute following the EXPRESS entity data type name shall be the second attribute, and so on.

EXAMPLE 28 - Definition in EXPRESS:

```
TYPE
  primary_colour_abbreviation = ENUMERATION OF (r, g, b);
END_TYPE;

ENTITY widget; ----------------------------->  A
  attribute1: INTEGER; --------------------->  B
  attribute2: STRING;  --------------------->  C
  attribute3: LOGICAL; --------------------->  D
  attribute4: BOOLEAN; --------------------->  E
  attribute5: REAL; ------------------------>  F
  attribute6: LIST [1 : 2] of LOGICAL;  ---->  G
  attribute7: ARRAY [-1:3]  of INTEGER;  --->  H
  attribute8: PRIMARY_COLOUR_ABBREVIATION; ->  I
END_ENTITY;
```

Sample entity instance in data section:

```
#1 = WIDGET( 1, 'A', .T., .F., 1.0, (.T.,.F.), (1,0,1,2,3), .R.);
          ^     ^   ^    ^    ^    ^       ^            ^          ^
          |     |   |    |    |    |       |            |          |
          A     B   C    D    E    F       G            H          I
```

A: The EXPRESS entity name widget is mapped to the WIDGET standard keyword of the data section entity.

B: attribute1 has a value of 1 in this entity instance.

C: attribute2 has a value of A in this entity instance.

D: attribute3 has a value of T in this entity instance.

E: attribute4 has a value of F in this entity instance.

F: attribute5 has a value of 1.0 in this entity instance.

G: attribute6 is a list of logicals in this entity instance. The list values are:

```
ATTRIBUTE6(1) = T
ATTRIBUTE6(2) = F
```

H: attribute 7 is an array of integers in this entity instance. The array values are:

```
ATTRIBUTE7(-1) = 1
ATTRIBUTE7( 0) = 0
ATTRIBUTE7( 1) = 1
ATTRIBUTE7( 2) = 2
ATTRIBUTE7( 3) = 3
```

I: Attribute 8 is an enumeration. The attribute contains a value of R.

11.2.2 Entity with explicit optional attributes

An entity with optional explicit attributes that are supplied shall follow the same mapping algorithm as an entity with explicit attributes. (See 11.2.1.) When the optional value is not supplied in an entity instance, the missing attribute value shall be encoded in the exchange structure as the *dollar sign* "$".

EXAMPLE 29 - Entity definition in EXPRESS:

```
ENTITY xxx;
  attribute1: REAL;
  attribute2: REAL;
END_ENTITY;

ENTITY yyy; ----------------------->  A
  attribute1: OPTIONAL LOGICAL; ----->  B
  attribute2: xxx; ----------------->  C
  attribute3: xxx; ----------------->  D
  attribute4: OPTIONAL INTEGER; ----->  E
  attribute5: OPTIONAL REAL; ------->  F
END_ENTITY;
```

Sample entity instances in data section:

```
#1=XXX(1.0,2.0);
#2=XXX(3.0,4.0);
#3=YYY($,#2,#1,$,$);
    ^   ^  ^  ^ ^ ^
    |   |  |  | | |
    A   B  C  D E F
```

A: The EXPRESS entity name yyy is mapped to the YYY standard keyword of the data section entity.

B: attribute1 does not have a value in this entity instance.

C: attribute2 is a reference to the xxx entity with entity instance #2.

D: attribute3 is a reference to the xxx entity with entity instance #1.

E: attribute4 does not have a value in this entity instance.

F: attribute5 does not have a value in this entity instance.

11.2.3 Entity with derived attributes

The derived attributes of an entity shall not be mapped to the exchange structure. When a derived attribute in a subtype redefines an attribute in a supertype, the mapping used shall be as described in 11.2.6.

EXAMPLE 30 - Entity definition in EXPRESS:

```
ENTITY yyy;
  q0: Real;
  q1: Real;
  q2: Real;
END_ENTITY;
```

```
ENTITY xxx;   --------------------------> A
  p0: yyy;    --------------------------> B
  p1: yyy;    --------------------------> C
  p2: yyy;    --------------------------> D
DERIVE
  attrib5 : vector := func_normal (p0,p1,p2);  ----> E
  attrib4 : real   := func_diameter (p0,p1,p2);  --> F
END_ENTITY;
```

Sample entity instances in data section:

```
#9  = YYY( 0.0, 0.0, 0.0);
#10 = YYY( 1.0, 2.0, 3.0);
#11 = YYY( 4.0, 5.0, 6.0);
#12 = XXX( #9, #10, #11);
      ^    ^   ^    ^
      |    |   |    |
      A    B   C    D
```

A: The EXPRESS entity name xxx is mapped to the standard keyword of the data section entity.

B: p0 is a reference to the yyy entity with an entity instance #9.

C: p1 is a reference to the yyy entity with an entity instance #10.

D: p3 is a reference to the yyy entity with an entity instance #11.

E: attrib5 does not map to the entity instance because it is a derived attribute.

F: attrib4 does not map to the entity instance because it is a derived attribute.

11.2.4 Entity with other entities as attributes

If an entity instance is specified as an attribute of a second (referencing) entity instance, the first (referenced) entity instance shall be mapped to the exchange structure as an entity instance name (see 7.3.4). The reference to this entity instance may occur inside or outside the SCOPE section of the referencing entity instance. The referenced entity instance shall be defined within the data section, i.e. somewhere within the data section the referenced entity instance shall occur on the left of the *equals sign* "=". This definition may precede or follow the use of the entity instance as an attribute.

EXAMPLE 31 - Entity definition in EXPRESS:

```
ENTITY yyy;
  x : REAL;
  y : REAL;
  z : REAL;
END_ENTITY;

ENTITY xxx;
  p0 :  yyy;  -------------->  A
  p1 :  yyy;  -------------->  B
END_ENTITY;
```

Sample entity instance in data section:

```
#1 = YYY(3., 4., 5.);
#2 = &SCOPE #3=YYY(1., 2., 3.);ENDSCOPE XXX (#1,#3);
                                             ^  ^
                                             |  |
                                             A  B
```

A: The attribute p0 is a reference to an entity that is of type yyy outside the SCOPE structure of the xxx entity.

B: The attribute p1 is a reference to an entity that is of type yyy in the SCOPE section of the xxx entity.

11.2.5 Entities defined as subtypes of other entities

Entities defined as subtypes or supertypes of other entities shall be mapped to the exchange structure using one of two mapping rules, internal mapping or external mapping. One mapping rule applies to each instance of a subtype entity. Entity instances that are not instances of ANY subtype entity, even when they are instances of an entity that is defined as a supertype, shall be mapped to the exchange structure as provided in 11.2.1.

ISO 10303-11 defines instances of an entity having a SUBTYPE clause to be "complex entity instances", in that they may involve attributes from more than one entity-type declaration. This subclause specifies how complex entity instances are mapped to the exchange structure.

This subclause applies ONLY to complex entity instances. It does not necessarily apply to every instance of a supertype entity. In particular, it does not apply to an instance of a supertype that is not an instance of any subtype. Such instances may exist if the supertype is not an abstract supertype and is not itself a subtype of some other entity.

The selection of which mapping rule to use for each entity instance depends on the conformance class chosen for the implementation. For implementations of conformance class 1, the choice of mapping is prescribed by 11.2.5.1. For implementations of conformance class 2, the external mapping prescribed by 11.2.5.3 shall be used for all complex entity instances.

11.2.5.1 Default selection of mapping

A set of entity data type definitions that are linked by subtype and implicit or explicit supertype expressions define a set of complex entity instance structures, referred to as the evaluated set in annex B of ISO 10303-11. The mapping that may be applied for a particular instance depends on the member of the evaluated set to which it corresponds.

Each particular instance of an entity data type that is declared to be a subtype corresponds to one member of the evaluated set for each of its supertypes. Each member of the evaluated set specifies a list of entity data type names.

To determine the mapping rules to apply to a given entity instance

a) Determine the list of entity data type names that are included in the evaluated set member(s) that correspond to the entity instance.

b) Identify from the list all entity-types that have no subtypes and all entity-types that may have subtypes but for which none of the subtypes appears in the list (evaluated set) for this instance.

c) If only one entity data type has been identified, this entity-type is referred to as the "leaf entity data type" and the internal mapping shall be used. Otherwise the external mapping shall be used.

NOTE - At least one entity data type will be identified in step 2 above.

11.2.5.2 Internal mapping

If the internal mapping is to be used, the syntax shall be that of the production ENTITY_INSTANCE using the production SIMPLE_RECORD (see table 3). The KEYWORD shall be the name of the leaf entity data type, and the PARAMETER_LIST shall contain the inherited attributes of all supertype entities and the explicit attributes of the leaf entity data type. The order in which the inherited and explicit attributes appear in the exchange structure shall be determined as follows:

— All inherited attributes shall appear sequentially prior to the explicit attributes of any entity.

— The attributes of a supertype entity shall be inherited in the order they appear in the supertype entity itself.

— If the supertype entity is itself a subtype of another entity, then the attributes of the higher supertype entity shall be inherited first.

— When multiple supertype entities are specified, the attributes of supertype entities shall be processed in the order specified in the SUBTYPE OF expression.

This procedure may result in a supertype entity being referenced more than once. In this case all references after the first one shall be ignored.

EXAMPLES

32 - An example of a simple subtype/supertype relationship. Entity definition in EXPRESS:

```
ENTITY aa ABSTRACT SUPERTYPE OF (ONEOF(bb,cc));  ------>  A
  attrib_a: zz;  ------------------------------------->  B
END_ENTITY;

ENTITY bb SUBTYPE OF (aa)
        ABSTRACT SUPERTYPE OF (ONEOF(xx));  ----------->  C
  attrib_b1: yy;  ------------------------------------>  D
  attrib_b2: yy;  ------------------------------------>  E
END_ENTITY;

ENTITY cc SUBTYPE OF (aa);
  attrib_c : REAL;
END_ENTITY;

ENTITY xx SUBTYPE OF (bb);
  attrib_x:  REAL;  ---------------------------------->  F
END_ENTITY;

ENTITY zz;
  attrib_z : STRING;
END_ENTITY;

ENTITY yy;
  attrib_1 : REAL;
  attrib_2 : REAL;
  attrib_3 : REAL;
END_ENTITY;
```

Sample entity instance of entity data type xx in data section:

```
#1 = ZZ('ZATTR');
#2 = YY(1.0, 2.0, 0.0);
#3 = YY(2.0, 2.0, 0.0);

#4 = XX(#1, #2, #3, 4.0);
         ^   ^   ^    ^
         |   |   |    |
         B   D   E    F
```

A: Because entity aa is an abstract supertype it does not map to the exchange structure.

B: The attribute attrib_a will map to the data section as an inherited attribute in an entity that is directly or indirectly subtyped to the aa entity. In this case, attrib_a is represented by the first attribute of the instance of xx, and refers to zz, entity instance #1.

C: Because entity bb is an abstract supertype it will not map to the exchange structure.

D: The attribute attrib_b1 will map to the data section as an inherited attribute in an entity that is directly or indirectly subtyped to the bb entity. In this case, attrib_b1 is represented by the second attribute of the instance of entity xx, and refers to yy, entity instance #2.

E: The attribute attrib_b2 will map to the data section as an inherited attribute in an entity that is directly or indirectly subtyped to the bb entity. In this case, attrib_b2 is represented by the third attribute of the instance of entity xx, and refers to yy, entity instance #3.

F: Attribute attrib_x is represented by its value 4.0 .

33 - An example of the mapping of a supertype that is not an ABSTRACT supertype. Entity definition in EXPRESS:

```
ENTITY aa SUPERTYPE OF (ONEOF(bb,cc));  -->  A
  attrib_a :  STRING;
END_ENTITY;

ENTITY bb SUBTYPE OF (aa); --------------->  B
  attrib_b :  INTEGER;
END_ENTITY;

ENTITY cc SUBTYPE OF (aa);  -------------->  C
  attrib_c : REAL;
END_ENTITY;

ENTITY dd ;  ----------------------------->  D
  attrib_d : aa;
END_ENTITY;
```

Sample entity instance in data section:

```
#1 = AA('SAMPLE STRING');  --------------->  A
#2 = BB('ABC', 666);  -------------------->  B
#3 = CC('XYZ', 99.99); ------------------->  C
#4 = DD(#1); ----------------------------->  D
#5 = DD(#2); ----------------------------->  D
#6 = DD(#3); ----------------------------->  D
```

A: Since it was not an abstract supertype, the supertype entity aa can be instantiated in an exchange structure. Note that it contains only its attrib_a attribute when it is instantiated.

B: The entity bb is a subtype of aa and therefore its instances will contain the attributes of both aa and bb.

C: The entity cc is a subtype of aa and therefore its instances will contain the attributes of both aa and cc.

D: The entity dd references entity aa as an attribute. Therefore, an instance of entity dd may reference any of #1, #2 or #3.

11.2.5.3 External mapping

If the external mapping is to be used, the syntax shall be that of the production ENTITY_INSTANCE using the production SUBSUPER_RECORD (see table 3). Each entity data type in the evaluated set member(s) corresponding to the entity instance, including any abstract supertypes, shall appear as a SIMPLE_RECORD within the *parentheses* of the SUBSUPER_RECORD. The entity data types shall be ordered within the SUBSUPER_RECORD in sequence of the given keywords in the order given by the collating sequence defined in 6.2. The collating sequence shall be determined by the entity data type name (the so-called long name) and not by the short name used for the encoding.

EXAMPLES

34 - An example of the mapping of subtypes related by ANDOR.

Entity definition in EXPRESS:

```
ENTITY aa SUPERTYPE OF (bb ANDOR cc);  -->  A
  attrib_a :  STRING;
```

```
END_ENTITY;

ENTITY bb SUBTYPE OF (aa); -------------->  B
  attrib_b :  INTEGER;
END_ENTITY;

ENTITY cc SUBTYPE OF (aa);  ------------->  C
  attrib_c : REAL;
END_ENTITY;

ENTITY dd ;  ---------------------------->  D
  attrib_d : aa;
END_ENTITY;
```

Sample entity instance in data section:

```
#1 = BB('sample string', 15);   ------------> A
#2 = CC('S', 3.0);  ------------------------> B
#3 = (AA('ASTRID')BB(17)CC(4.0));  ---------> C
#4 = DD(#1);  ------------------------------> D
#5 = DD(#2);  ------------------------------> D
#6 = DD(#3);  ------------------------------> D
#7 = AA('ABC');  ---------------------------> E
```

A: #1 is an instance of aa and bb combined.

B: #2 is an instance of aa and cc combined.

C: #3 is an instance of aa, bb and cc combined.

D: The entity dd references entity aa as an attribute. Therefore, an instance of entity dd may legally reference any of #1, #2 or #3.

E: The non-abstract supertype aa can be instantiated, and the internal mapping applies because the evaluated set contains only one member.

35 - An example of the mappings of a more complicated subtype/supertype graph. Entity definition in EXPRESS:

```
ENTITY x;
  attrib_x : INTEGER;
END_ENTITY;

ENTITY a ABSTRACT SUPERTYPE OF(ONEOF(b,c));  -->  A
  attrib_a : x  ----------------------------->  B
END_ENTITY;

ENTITY b SUPERTYPE OF(d ANDOR e)
    SUBTYPE OF (a);
  attrib_b : REAL;  ------------------------->  B
END_ENTITY;

ENTITY c SUBTYPE OF (a);  ------------------->  C
  attrib_c : REAL;
END_ENTITY;

ENTITY d SUBTYPE OF (b);  ------------------->  D
  attrib_d : x;
END_ENTITY;

ENTITY e ABSTRACT SUPERTYPE
      SUBTYPE OF (b);  ---------------------->  A
  attrib_e : x;  ---------------------------->  B
END_ENTITY;
```

```
ENTITY f SUPERTYPE OF (h);
  attrib_f : x;  ------------------------------>  B
END_ENTITY;

ENTITY g SUBTYPE OF (e);  --------------------->  E
  attrib_g : INTEGER;
END_ENTITY;

ENTITY h SUBTYPE OF (e,f);  ------------------->  E
  attrib_h : INTEGER;
END_ENTITY;
```

A: Since entity a and e are abstract supertypes they cannot occur on the exchange structure as independent instances.

B: Since attrib_a, attrib_b, attrib_e and attrib_f are attributes of supertype entities, they will be mapped as inherited attributes if a subtype is mapped using the internal mapping. They will be mapped as attributes of the entities in which they are declared if a subtype is mapped using the external mapping.

C: Since entity c participates in an ONEOF operation and its supertype participates in no supertype operation, it will use the internal mapping.

D: The mapping of d will depend on the structure of the evaluated set in which it appears.

E: Since entities g and h both have a supertype (entity e) that participates in an ANDOR operation. their mapping will depend on the structure of the evaluated set in which they appear.

36 - An entity instance showing internal mapping.

```
#1=X(1);
#2=C(#1, 2.0);
   ^  ^   ^
   |  |   |
   A  B   C
```

A: The evaluated set of '#2' is [c & a] and therefore uses the internal mapping.

B: attrib_a is inherited by entity data type c. The evaluated set is a reference to an instance of entity data type x.

C: attrib_c appears after all inherited attributes.

37 - Entity instance showing internal mapping:

```
#4=X(3);
#1=X(1);
#2=D(#1, 2.0, #4)
   ^  ^   ^    ^
   |  |   |    |
   A  B   C    D
```

A: Since entity instance #2 belongs to the evaluated set [a & b & d] that has exactly one leaf (d), it is internally mapped.

B: The attribute of a with name attrib_a is inherited by entity data type d.

C: attrib_b is inherited by entity data type d.

D: attrib_d is the last attribute in the d instance because inherited attributes from the supertype entities a and b come first.

38 - Entity instance showing external mapping:

```
#1=X(1);
#2=(A(#1) B(9.0) D(#1) E(#1) F(#1) H(4) );  -------------> A
```

A: Since entity instance #2 is a member of [a & b & d & e & f & h] and this evaluated set has more than one leaf (d and h), external mapping is used. There is no single entity data type name that can be associated with the entity; rather it can be considered to have the composite name a-b-d-e-f-h. The *spaces* between the entity records are optional and have been added to this example for ease of reading.

11.2.6 Entities with attributes redefined as an effect of a DERIVE

If a subtype entity redefines an attribute of its supertype using the DERIVE clause, that attribute in the supertype shall be encoded with an *asterisk* "*".

EXAMPLE 39 - Entity definition in EXPRESS:

```
ENTITY point SUPERTYPE OF ONEOF(point_on_curve) ;
  x : REAL;
  y : REAL;
  z : REAL;
END_ENTITY;

ENTITY point_on_curve SUBTYPE OF (point);
  u : REAL;
  c : curve;
DERIVE
  SELF\point.x : real := fx(u, c);
  SELF\point.y : real := fy(u, c);
  SELF\point.z : real := fz(u, c);
END_ENTITY;

ENTITY curve;
  attr : STRING;
END_ENTITY;
```

Sample entity instance in data section

```
#1 = CURVE('curve_attribute');
#2 = POINT_ON_CURVE( *, *, *, 0.55, #1);  -----------> A
#3 = POINT(2.0, 3.0, 4.0);  -------------------------> B
```

A: Because a subtype with derived attributes is used here, the attributes x, y and z are replaced by *asterisks*.

B: Because POINT is not an ABSTRACT SUPERTYPE, it is possible to have an instance of POINT in the exchange structure. The attributes x, y, and z appear as normal.

11.2.7 Attributes redefined as an effect of a subtype declaration

A supertype attribute may be redefined by a subtype entity if the attribute name in the subtype is the same as in the supertype. In this situation, the inheritance mapping shall be encoded by the same procedure as for entities defined as subtypes or supertypes of other entities ignoring the redefinition of the attribute. The value of the redefined attribute in the supertype shall be encoded according to the normal mapping. The value of the redefined attribute in the subtype shall be encoded as an *asterisk* "*".

EXAMPLE 40 - Entity definition in EXPRESS:

```
ENTITY aaa SUPERTYPE OF (ONEOF( bbb, ccc));
  a1      : NUMBER;
  a2      : curve;
END_ENTITY;

ENTITY bbb SUBTYPE OF (aaa);
  SELF\aaa.a1      : INTEGER;
  b       : REAL;
END_ENTITY;

ENTITY ccc SUBTYPE OF (aaa);
  SELF\aaa.a2      : line;
END_ENTITY;
```

Sample instances in data section:

```
#1 = LINE(...);
#2 = CURVE(...);
#3 = BBB(1, #2, *, 0.5);  ---------> A
#4 = CCC(1.5, #1, *);  ------------> B
```

A: The third attribute is the redefined a1 attribute of aaa. This attribute value is replaced with a "*"', because the value of this attribute has already been reported in the first attribute.

B: The third attribute is the a2 attribute of aaa. This attribute value is replaced with a "*", because the value of this attribute has already been reported in the first attribute.

11.2.8 Mapping of the EXPRESS element of NUMBER

The EXPRESS element of NUMBER shall be mapped to the exchange structure as a REAL.

11.2.9 Entity WHERE rules

WHERE rules shall not be mapped to the exchange structure.

EXAMPLE 41 - Entity definition in EXPRESS:

```
ENTITY widget; ---------------------->  A
  a : REAL; ------------------------>  B
  b : REAL; ------------------------>  C
  c : REAL; ------------------------>  D
WHERE
  a ** 2 + b ** 2 + c ** 2 = 3.0; ---->  E
END_ENTITY;
```

Sample entity instance in data section.:

The WHERE rules are not instantiated in the entity instance.

```
#2 = WIDGET( 1.0, 1.0, 2.0);
         ^     ^    ^    ^
         |     |    |    |
         A     B    C    D
```

A: The EXPRESS entity name widget is mapped to the entity data type keyword of the data section entity.

B: Attribute a has a value of 1.0 in the entity instance.

C: Attribute b has a value of 1.0 in the entity instance.

D: Attribute c has a value of 2.0 in the entity instance.

E: The WHERE rule did not map to the exchange structure. The entity is syntactically correct. However, the WHERE rule is not satisfied by the values of the three attributes.

11.2.10 Entity with INVERSE attributes

Attributes within the INVERSE clause shall not be mapped to the exchange structure.

11.3 Mapping of the EXPRESS element of SCHEMA

The EXPRESS element of SCHEMA shall not be mapped to the exchange structure. The name of the schema that specifies entities that appear in an exchange structure shall be mapped to the header section of the exchange structure by use of an instance of the **file-schema** entity data type as specified in 9.2.3.

11.4 Mapping of the EXPRESS element of CONSTANT

The EXPRESS element of CONSTANT shall not be mapped to the exchange structure.

Note - The existence of multiple references to the same constant is not preserved when that data is mapped to the exchange structure.

11.5 Mapping of the EXPRESS element of RULE

The EXPRESS element of RULE shall not be mapped to the exchange structure.

11.6 Remarks

Remarks shall not be mapped to the exchange structure.

12 Printed representation of exchange structures

The graphic character combinations as specified in table 6 may be used to control the printed appearance of an exchange structure. These directives may appear at any position where a token separator may appear, and within strings and binaries. Details on token separators are given in table 3. The graphic character combinations shall appear together without any intervening graphic characters.

Explicit print control directives are also allowed within strings. The directives "\N\" and "\F\" do not contribute to the effective contents of the string. For the encoding of strings see 7.3.3. The print control directives "\N\" and "\F\" are relevant only for the printed presentation of the exchange structure and are to be ignored otherwise. Annex E provides guidance on the printing of the exchange structure.

Table 6 - Print control directives

GRAPHIC CHARACTER SEQUENCE	MEANING
\N\ REVERSE_SOLIDUS N REVERSE_SOLIDUS	NEWLINE
\F\ REVERSE_SOLIDUS F REVERSE_SOLIDUS	FORMFEED

Annex A
(normative)

File representation on storage media

The following media containing exchange structures are distinguished:

— Record-Oriented Transport - usually characterized by magnetic tape but which could be any storage media with record-oriented storage;

— Line-Oriented Transport - usually characterized by diskette but which could be any storage media with line-oriented storage.

In both cases the intent is to transport a sequence of graphic characters from the basic alphabet as specified in this part of ISO 10303. The following conventions apply for media containing the exchange structure.

A.1 Record-oriented transport content

A magnetic tape may contain several data sets. Each data set may be a sequential file whose content can be interpreted as an exchange structure conforming to this part of ISO 10303.

It is the responsibility of the sender of such a tape to communicate to any receiver, either on the tape or otherwise, information on which data sets are exchange structures conforming to this part of ISO 10303.

The transport format for each exchange structure is as follows:

— The file shall consist of a sequence of records;

— The first graphic characters in the first record shall be the special token "ISO-10303-21;" that initiates the exchange structure. The formatting of the storage medium is dependent on the operating system and is a matter of agreement between the sender and the receiver;

— The last record shall be padded (if necessary) with *spaces* after the *semicolon* of the special token "END-ISO-10303-21;".

A.1.1 Transport format for magnetic tape media

For tape transport of exchange files, the following characteristics defined in ISO 3788 should be used:

— 12,7 mm (0.5 in) inch tape width;

— Unlabelled;

— 9 track tape;

— 63 cpmm (1600 bpi) tape density;

NOTE - 246 cpmm (6250 bpi) tape density may also be used.

— 4000 octet (8-bit byte) physical block size;

— Blocks separated by inter-record gaps;

— Last block followed by tape mark.

A.1.2 Other storage media with record-oriented storage

Other media on which exchange structures are stored as sequences of records may use the same transport format as specified for tapes.

A.2 Line-oriented transport content

Some magnetic meda contain data sets stored as a sequence of lines. Each of these data sets may be a sequential file whose content can be interpreted as a file conforming to this part of ISO 10303.

The file shall consist of a sequence of lines:

— The first graphic characters in the first line shall be the special token "ISO-10303-21;" that initiates the exchange structure.

— Each line shall be terminated by a sequence of two 8-bit bytes, the first with decimal value of 13, the second with decimal value 10. These two 8-bit bytes shall be ignored when reading the exchange structure;

— The last line shall be padded (if necessary) with *spaces* after the *semicolon* of the special token "END-ISO-10303-21;" that terminates the exchange structure. The file shall be terminated by the 8-bit byte with decimal value 26. This byte shall be ignored when reading the exchange structure.

A.2.1 Transport format for diskette media

Any of the popular media for diskettes (3 1/2", 5 1/4", low or high density) may be used to transmit the exchange structure.

It is the responsibility of the sender of such a diskette to communicate to any receiver either on the diskette or outside the diskette which data sets are exchange structures adhering to this part of ISO 10303.

A.2.2 Other media

Other media on which files are stored as sequences of lines may use the same transport format as specified for diskettes. In particular, this format may be appropriate for transfer over communication networks (E-mail).

A.3 Treatment of multi-volume files

It may be necessary to spread an exchange structure over more than one physical volume. The way multi-volume files are organised is outside the scope of this part of ISO 10303.

NOTE - Depending on circumstances, special administrative software or the operating system of the receiving system may concatenate the physically separate parts of a multi-volume file into one exchange structure.