INTERNATIONAL STANDARD

ISO 10303-21

Second edition
2002-01-15

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 10303-21:2002(E)

Contents

Tables page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardizations.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO 10303 may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

International Standard ISO 10303-21 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 4, *Industrial data*.

This second edition cancels and replaces the first edition (ISO 10303-21:1994), of which it constitutes a technical revision. It incorporates the corrections published in ISO 10303-21:1994/Cor.1:1996.

This International Standard is organized as a series of parts, each published separately. The structure of this International Standard is described in ISO 10303-1.

Each part of this International Standard is a member of one of the following series: description methods, implementation methods, conformance testing methodology and framework, integrated generic resources, integrated application resources, application protocols, abstract test suites, application interpreted constructs, and application modules. This part is a member of the implementation methods series.

A complete list of parts of ISO 10303 is available from the Internet:

`<http://www.nist.gov/sc4/editing/step/titles/>`

Annexes A, B, C, D, E and F form a normative part of this part of ISO 10303. Annexes G and H are for information only.

Introduction

ISO 10303 is an International Standard for the computer-interpretable representation of product information and for the exchange of product data. The objective is to provide a neutral mechanism capable of describing products throughout their life cycle. This mechanism is suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases, and as a basis for archiving.

This part of ISO 10303 specifies a mechanism that allows product data described in 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.

This edition incorporates the following technical modifications to ISO 10303-21:1994:

— the SCOPE structure (&SCOPE / ENDSCOPE) has been eliminated;

— the exchange structure may now contain multiple data sections;

— the exchange structure header section may now identify the default language for string attributes of entity instances encoded in a data section;

— the exchange structure header section may now identify information describing contexts within which the entity instances encoded in a data section are applicable;

— enumeration values may now be encoded using short names if such names are available.

All exchange structures that are encoded according to the previous edition of ISO 10303-21 and that do not use the SCOPE structure also conform to this edition.

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 exchange 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 normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO 10303. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO 10303 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO 639-2:1998, *Codes for the representation of names of languages — Part 2: Alpha-3 code.*

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 (3 200 ftpi) — 63 cpmm (1 600 cpi).*

ISO 8601:2000, *Data elements and interchange formats — Information interchange — Representation of dates and times.*

ISO/IEC 8824-1:1998, *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

ISO/IEC 8859-1:1998, *Information technology — 8 bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1.*

ISO/IEC 8859-2:1999, *Information technology — 8 bit single-byte coded graphic character sets — Part 2: Latin alphabet No. 2.*

ISO/IEC 8859-3:1999, *Information technology — 8 bit single-byte coded graphic character sets — Part 3: Latin alphabet No. 3.*

ISO/IEC 8859-4:1998, *Information technology — 8 bit single-byte coded graphic character sets — Part 4: Latin alphabet No. 4.*

ISO/IEC 8859-5:1999, *Information technology — 8 bit single-byte coded graphic character sets — Part 5: Latin/Cyrillic alphabet.*

ISO/IEC 8859-6:1999, *Information technology — 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/IEC 8859-8:1999, *Information technology — 8 bit single-byte coded graphic character sets — Part 8: Latin/Hebrew alphabet.*

ISO/IEC 8859-9:1999, *Information technology — 8 bit single-byte coded graphic character sets — Part 9: Latin alphabet No. 5.*

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

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

ISO/IEC 10646-1:2000, *Information Processing — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.*

3 Terms, definitions, and abbreviations

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 terms 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.

— complex entity instance;

— data type;

— entity;

— partial complex entity instance;

— simple entity instance;

— token.

3.5 Other definitions

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

3.5.1
basic alphabet
the set of characters G(02/00) through G(07/14) of ISO 8859-1.

3.5.2
clear text encoding
the encoding of information, using a sequence of codes for characters in the basic alphabet.

3.5.3
control directive
a sequence of characters in the basic alphabet.

3.5.4
keyword
a special sequence of characters identifying an entity or a defined type in the exchange structure.

3.5.5
section
a collection of data of the same functional category of information.

3.5.6
sequential file
a file that can only be accessed in a sequential manner.

3.5.7
token separator
a sequence of one or more 8-bit bytes that separate any two tokens.

3.6 Abbreviations

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

BMP	Basic multilingual plane
WSN	Wirth Syntax Notation

4 Exchange structure fundamental concepts and assumptions

4.1 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 to the exchange structure syntax.

4.2 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, an annotation is introduced by the sequence `---->` where clarification is required.

4.3 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 instances represented in the exchange structure conform to the schemas listed in the header section of the exchange structure if every requirement or constraint of these schemas is satisfied with respect to each instance or grouping of instances to which it shall apply whatsoever and the mapping requirements defined in clauses 9 and 10 of this part of ISO 10303 are satisfied.

NOTE Annex F presents methods for evaluating schema conformance when an exchange structure contains multiple data sections based on different EXPRESS schemas.

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 complex entity instances (see 10.2.5). 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.

5 Formal definitions

5.1 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.

5.2 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.

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
```

5.3 Exchange structure

The exchange structure shall be a sequential file using a clear text encoding. The exchange structure shall consist of at least two sections: the header section and one or more data sections. The header section provides data relating to the exchange structure itself. The structure of the header section is specified in clause 8. The data section provides the data to be transferred. The structure of the data section is specified in clause 9. 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.

5.4 Definition of tokens

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

5.5 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 10.

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 } """" .
```

5.6 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 11.

NOTE *Space* is the only whitespace character within the basic alphabet described in 5.2. Line-delimiters such as *line feed* or *carriage return* are permitted in the exchange structure by annex A, but do not belong to the basic alphabet and are required by annex A to be ignored when processing the exchange structure. Consequently, line breaks may appear anywhere within the structure, including within tokens.

A comment shall be encoded as a *solidus asterisk* "/*" followed by any number of characters from the basic alphabet, 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 3 - WSN of the exchange structure

```
EXCHANGE_FILE      = "ISO-10303-21;"
                        HEADER_SECTION DATA_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" [ "(" PARAMETER_LIST ")" ] ";"
                      ENTITY_INSTANCE_LIST "ENDSEC;"
ENTITY_INSTANCE_LIST = { ENTITY_INSTANCE } .
ENTITY_INSTANCE = SIMPLE_ENTITY_INSTANCE | COMPLEX_ENTITY_INSTANCE .
SIMPLE_ENTITY_INSTANCE  = ENTITY_INSTANCE_NAME "=" SIMPLE_RECORD ";" .
COMPLEX_ENTITY_INSTANCE = ENTITY_INSTANCE_NAME "=" SUBSUPER_RECORD ";" .
SIMPLE_RECORD     = KEYWORD "(" [ PARAMETER_LIST ] ")" .
SUBSUPER_RECORD   = "(" SIMPLE_RECORD_LIST ")" .
SIMPLE_RECORD_LIST = SIMPLE_RECORD { SIMPLE_RECORD } .
```

6 Tokens

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

6.1 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 sections of an exchange structure, and the special token "ENDSEC;" shall be used to close the data sections of an exchange structure.

The special token *dollar sign* ("$") is used to represent an object whose value is not provided in the exchange structure.

The special token *asterisk* ("*") is used to represent an object whose value is not provided in the exchange structure but can be derived from other values according to rules given in the EXPRESS schema (see 10.2.6).

The special tokens *semicolon* (";"), *parentheses* ("(", ")"), *comma* (",") and *solidus* ("/") are used to punctuate the exchange structure.

6.2 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.

6.3 Simple data type encodings

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

6.3.1 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

Valid integer expressions	Meaning
16	Positive 16
+12	Positive 12
-349	Negative 349
012	Positive 12
00	Zero

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

6.3.2 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

Valid real expressions	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 real expressions	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

6.3.3 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).

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 .
```

Additional characters shall be encoded using the hexadecimal digits (see HEX in Table 2) as defined in 6.3.3.1 and 6.3.3.2. The WSN of control directives for encoding strings is given in Table 4.

6.3.3.1 Encoding the full alphabet of ISO 8859 within a string

In ISO 8859, G(x/y) is the notation for the character in "column" x "row" y, i.e., code value $(16 \cdot x) + y$, in the code table. Each part of ISO 8859 includes the basic alphabet (see 5.2) as positions G(02/00) through G(07/14). The various parts of ISO 8859 differ in the symbols of the extended character set — positions G(10/00) through G(15/14). To include characters from the extended character set in a string requires the use of control directives.

The PAGE control directive — *reverse solidus capital letter S reverse solidus* ("\S\") CHARACTER (see Table 4) — is used within a string to allow a character in the basic alphabet to represent the character in the corresponding position in the extended alphabet. The PAGE control directive shall be interpreted in the string as the single character G((x+8)/y), where G(x/y) is the basic alphabet character following the "\S\". That is, if the basic alphabet character has code value v, it shall be interpreted as the character with code value v + 128.

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.

EXAMPLE

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

6.3.3.2 Encoding the character sets of ISO 10646 within a string

ISO 10303-11:1994, clause 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 ideographic 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 1 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 2 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.

6.3.3.3 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 representing the cell octet of a character in row 0 of the BMP in ISO 10646.

NOTE The characters defined by ISO 10646 and ISO 8859-1 are identical within this range.

EXAMPLE

String as stored	Effective contents	Comments
`'see \X\A7 4.1'`	see § 4.1	
`'line one\X\0Aline two'`	line one line two	Contains embedded newline.

6.3.3.4 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 (see clause 11) or characters encoded according to 6.3.3.1, 6.3.3.2, or 6.3.3.3 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.

6.3.4 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 entity instance names is given in Table 2 in the ENTITY_INSTANCE_NAME production.

EXAMPLE

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.

6.3.5 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

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.

6.3.6 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 ≤ n ≤ 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

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

7 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 (possibly empty) sequence of PARAMETERs, each of which may be:

— a simple type encoding, as described in 6.3, or

— the special token *dollar sign* ("$"), or

— a TYPED_PARAMETER, representing an instance of a select type (see 10.1.8), or

— a LIST, representing an instance of a (nested) structured type.

A given LIST may contain more than one of the above forms. 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

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.

8 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 H presents an example of a header section within an exchange structure.

8.1 Header section entities

The header section of every exchange structure shall contain one instance of each of the following entities: **file_description**, **file_name**, and **file_schema**, and they shall appear in that order. Instances of **file_population**, **section_language** and **section_context** may appear after **file_schema**. If instances of user-defined header section entities are present, they shall appear after the header section entity instances defined in this section. 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 10 provides mapping of simple and aggregate data types to the PARAMETER_LIST for the attribute values of these entity instances.

8.2 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;

TYPE exchange_structure_identifier = STRING;
END_TYPE;
(*
```

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

NOTE The exchange_structure_identifier type serves the same purpose the identifier type in ISO 10303-41 but has been defined separately in order to keep this part of ISO 10303 independent from the data models defined in the ISO 10303 integrated resource series parts.

8.2.1 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 specification to which the encoding in this exchange structure conforms and any conformance options employed in that encoding. The value of this attribute shall indicate conformance to this version of this part of ISO 10303 by having either the value "3;1" or the value "3;2". The value for exchange structures adhering to conformance class 1 shall be "3;1". The value for exchange structures adhering to conformance class 2 shall be "3;2".

If the following restrictions on the encoding are met, the value "2;1" or the value "2;2" may be used to indicate conformance to this version of this part of ISO 10303:

— the exchange structure shall contain a single data section, and the "DATA" keyword shall not be followed by a parenthesized PARAMETER_LIST;

— the exchange structure header section shall not contain FILE_POPULATION entities;

— the exchange structure header section shall not contain SECTION_LANGUAGE entities;

— the exchange structure header section shall not contain SECTION_CONTEXT entities;

— the enumerated values of an EXPRESS ENUMERATION shall not be encoded using short names.

If used, the value "2;1" shall designate exchange structures adhering to conformance class 1, and the value "2;2" shall designate exchange structures adhering to conformance class 2.

NOTE 1 Conformance classes 1 and 2 are defined in 10.2.5.

NOTE 2 The general form for the value is "v;cc", where v is the version number of this part of ISO 10303, as specified in annex C, and cc is the encoding of conformance class. Future versions of this part of ISO 10303 may specify additional values for v and cc.

NOTE 3 The use of "2;1" and "2;2" is provided to support upward compatibility with implementations based on the previous version of this part of ISO 10303.

8.2.2 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.

EXPRESS Specification:

```
*)
ENTITY file_name;
  name                 : STRING (256) ;
  time_stamp           : time_stamp_text ;
  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;


TYPE time_stamp_text = STRING(256);
END_TYPE;
(*
```

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 4.2.1.1 of ISO 8601 concatenated to the extended format for the time of the day as specified either in 4.3.1.1 or in 4.3.3 of ISO 8601. The date and time shall be separated by the *capital letter T* as specified in 4.4.1 of ISO 8601. The alternate formats of 4.3.1.1 and 4.3.3 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

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

8.2.3 file_schema

The **file_schema** entity identifies the EXPRESS schemas that specify the entity instances in the data sections. 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 have the form 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.

NOTE The general form of an object identifier is a sequence of space-delimited integers. The sequence is enclosed within *braces* ("{", "}").

<u>EXPRESS Specification:</u>

```
*)
ENTITY file_schema;
  schema_identifiers : LIST [1:?] OF UNIQUE 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.

EXAMPLE The instance below identifies an EXPRESS schema called 'CONFIG_CONTROL_DESIGN':

```
FILE_SCHEMA (('CONFIG_CONTROL_DESIGN'));
```

The instance below uses an object identifier to indicate a specific version of an EXPRESS schema called 'AUTOMOTIVE_DESIGN':

```
FILE_SCHEMA (('AUTOMOTIVE_DESIGN { 1 0 10303 214 1 1 1 }'));
```

8.2.4 file_population

The **file_population** entity identifies a collection of entity instances within the exchange structure for the purpose of determining schema conformance to a particular EXPRESS schema. The collection of entity instances shall be determined by applying an algorithm identified by the attribute **determination_method** to the set of data sections identified by the attribute **governed_sections**. If no value is provided for **governed_sections**, the algorithm shall be applied to all data sections in the exchange structure.

An exchange structure may contain zero, one, or many **file_population** instances. A data section name may appear in the **governed_sections** attribute of zero, one, or many **file_population** instances.

NOTE 1 Clause F.2 defines three possible determination methods.

NOTE 2 If no value is provided for the governed_sections attribute, the attribute is encoded by a *dollar sign* ("$"), as specified in 10.2.2, and not as an empty list.

EXPRESS Specification:

```
*)
ENTITY file_population;
  governing_schema      : schema_name;
  determination_method  : exchange_structure_identifier;
  governed_sections     : OPTIONAL SET [ 1 : ? ] OF section_name;
END_ENTITY;

TYPE section_name = exchange_structure_identifier;
END_TYPE;
(*
```

Attribute Descriptions:

governing_schema: name of the EXPRESS schema that applies to population of entity instances identified by this header section **file_population** entry. The schema name must appear in the header section **file_schema** entry.

determination_method: string of graphic characters used to identify an algorithm to be used for selecting entity instances in the exchange.

governed_sections: the names of the data sections that are used as input to the determination method.

8.2.5 section_language

The **section_language** entity identifies the default language for string values in a data section. The attribute **default_language** shall contain the name of the language. The name of the language shall be encoded using the Alpha-3 bibliographic code specified in ISO 639-2.

The attribute **section** shall contain the name of a data section in the exchange structure for which the default language shall apply. If the exchange structure contains a single, unnamed, data section, no value shall be provided for the attribute **section** (see 10.2.2). The header section of an exchange structure shall contain at most one **section_language** instance where no value is provided for the attribute **section**. If present, the default language encoded by this instance shall apply to all data sections in the exchange structure for which no other **section_language** instance applies.

EXAMPLE Some possible values for default_language are ’eng’ for English, ’fre’ for French, ’rus’ for Russian, or ’ger’ for German.

EXPRESS Specification:

```
*)
ENTITY section_language;
  section          : OPTIONAL section_name;
  default_language : language_name;
UNIQUE
  UR1: section;
END_ENTITY;

TYPE language_name = exchange_structure_identifier;
END_TYPE;
(*
```

Attribute Descriptions:

section: name of the data section for which the default_language is to apply.

default_language: name of the language used for string values.

8.2.6 section_context

The **section_context** entity identifies information describing the contexts within which the instances encoded in the exchange structure are applicable.

The attribute **section** shall contain the name of a data section in the exchange structure for which the context identifiers shall apply. If the exchange structure contains a single, unnamed, data section, no value shall be provided for the attribute **section** (see 10.2.2). The header section of an exchange structure shall contain at most one **section_context** instance where no value is provided for the attribute **section**. If present, the context identifiers encoded by this instance shall apply to all data sections in the exchange structure for which no other **section_context** instance applies.

<u>EXPRESS Specification:</u>

```
*)
ENTITY section_context;
  section             : OPTIONAL section_name;
  context_identifiers : LIST [1:?] OF context_name;
UNIQUE
  UR1: section;
END_ENTITY;

TYPE context_name = STRING;
END_TYPE;
(*
```

<u>Attribute Descriptions:</u>

section: name of the data section for which the context_identifiers are to apply.

context_identifiers: identifiers which convey contextual information about instances encoded in the exchange structure.

NOTE An application protocol may define symbolic identifiers for each application protocol conformance class. The context identifiers for a section may indicate a partial list of application protocol conformance classes satisfied by the data in the section.

EXAMPLE 1 Language and context identifier assignments for an exchange structure with a single, unnamed, data section.

```
HEADER;
   .
   .
FILE_SCHEMA(('GEOMETRY'));
SECTION_LANGUAGE ($,'eng'); -----------------> A
SECTION_CONTEXT ($,('tag_a')); --------------> B
ENDSEC;
DATA;
   .
ENDSEC;
```

A: Assign the language encoded 'eng' (English) to the data section.

B: Assign the context tag 'tag_a' to the data section.

EXAMPLE 2 Language and context identifier assignments for an exchange structure with several data sections.

```
HEADER;
   .
   .
FILE_SCHEMA(('GEOMETRY'));
SECTION_LANGUAGE ('DS1','ger'); -------------> A
SECTION_LANGUAGE ('DS2','epo'); -------------> B
SECTION_LANGUAGE ($,'haw'); -----------------> C
SECTION_CONTEXT ('DS1',('tag_a','tag_b')); --> D
SECTION_CONTEXT ('DS2',('tag_c')); ----------> E
SECTION_CONTEXT ($,('tag_d')); --------------> F
ENDSEC;
DATA ('DS1',('GEOMETRY'));
   .
ENDSEC;
DATA ('DS2',('GEOMETRY'));
   .
ENDSEC;
DATA ('DS3',('GEOMETRY'));
   .
ENDSEC;
DATA ('DS4',('GEOMETRY'));
   .
ENDSEC;
```

A: Assign the language encoded 'ger' (German) to the data section named 'DS1'.

B: Assign the language encoded 'epo' (Esperanto) to the data section named 'DS2'.

C: Assign the language encoded 'haw' (Hawaiian) to all data sections that do not otherwise have a language assignment. These are the sections named 'DS3' and 'DS4'.

D: Assign the context identifiers 'tag_a' and 'tag_b' to the data section named 'DS1'.

E: Assign the context identifier 'tag_c' to the data section named 'DS2'.

F: Assign the context identifier 'tag_d' to all data sections that do not otherwise have a context identifier assignment. These are the sections named 'DS3' and 'DS4'.

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

8.3 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) The attributes of user-defined header section entities shall have EXPRESS data types and shall be mapped to the header section as specified in clause 10.

EXAMPLE

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

9 Data sections

The data sections contain instances to be transferred by the exchange structure. At least one data section shall be present in every exchange structure. Each data section contains instances of entities that correspond to one EXPRESS schema specified in the header section.

The syntax of the data section is specified in Table 3. Each data section shall begin with the "DATA" keyword. If an exchange structure contains more than one data section, each "DATA" keyword shall be followed by a parenthesized PARAMETER_LIST containing a STRING and a LIST parameter.

The first parameter shall be a STRING containing a unique name for the section. The second parameter shall be a LIST containing exactly one STRING. The string shall be the name of the schema that shall govern the data section. The schema name must appear in the header section **file_schema** entry.

If an exchange structure contains only one data section, the parenthesized PARAMETER_LIST may be omitted. In this case, the header section **file_schema** entry shall specify only one schema, and that schema shall govern the data section.

Each data section shall be terminated with the special token "ENDSEC;".

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

9.1 Data section entity instances

Each entity instance shall be mapped to an ENTITY_INSTANCE (see Table 3) in the data section, as specified in 10.2. Each entity instance shall be represented at most once in the exchange structure and shall have an entity instance name that is unique within the exchange structure. The entity instances need not be ordered in the exchange structure. An entity instance name may be referenced before it is defined by an ENTITY_INSTANCE in the exchange structure.

9.2 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

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

NOTE Rather than use the user-defined syntax defined in this clause, it is recommended that an implementation define an EXPRESS schema for the user-defined information and encode this information in a separate data section.

10 Mapping from EXPRESS to the exchange structure

This clause describes how instances of data types defined in the EXPRESS language are mapped to the exchange structure.

The EXPRESS language includes TYPE and ENTITY declarations, CONSTANT declarations, constraint specifications and algorithm descriptions. Only instances of data types, as defined by EXPRESS data types and TYPE and ENTITY declarations, are mapped to the exchange structure. Other elements of the language are not mapped to the exchange structure.

10.1 Mapping of EXPRESS data types

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

10.1.1 Mapping of EXPRESS simple data types

10.1.1.1 Integer

Values of the EXPRESS data type INTEGER shall be mapped to the exchange structure as an integer data type. 6.3.1 describes the composition of an integer data type.

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 instance
ENTITY AS ATTRIBUTE	entity instance name
ENUMERATION	enumeration
FUNCTION	NO INSTANTIATION
INTEGER	integer
INVERSE	NO INSTANTIATION
LIST	list
LOGICAL	enumeration
NUMBER	real
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
UNIQUE rule	NO INSTANTIATION
WHERE RULES	NO INSTANTIATION

10.1.1.2 String

Values of the EXPRESS data type STRING shall be mapped to the exchange structure as a string data type. 6.3.3 describes the composition of a string data type.

10.1.1.3 Boolean

Values of the EXPRESS data type BOOLEAN shall be mapped to the exchange structure as an enumeration data type. 6.3.5 describes the composition of a enumeration data type. The EXPRESS data type 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.

10.1.1.4 Logical

Values of the EXPRESS data type LOGICAL shall be mapped to the exchange structure as an enumeration data type. 6.3.5 describes the composition of a enumeration data type. The EXPRESS data type 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.

10.1.1.5 Real

Values of the EXPRESS data type REAL shall be mapped to the exchange structure as a real data type. 6.3.2 describes the composition of a real data type.

EXAMPLE 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 instance in the data section:

```
#2 = WIDGET(99, 99999, 'ABC', 'ABCDEFG', .T., .F., 9., 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 the value of 9. in this entity instance. The precision specification does not affect the encoding.

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

10.1.1.6 Binary

Values of the EXPRESS data type BINARY shall be mapped to the exchange structure as a binary data type. 6.3.6 describes the composition of a binary data type.

EXAMPLE 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.

10.1.1.7 Number

Values of the EXPRESS data type NUMBER shall be mapped to the exchange structure as a real data type. 6.3.2 describes the composition of a real data type.

10.1.2 List

Values of the EXPRESS data type LIST shall be mapped to the exchange structure as a list data type. Clause 7 describes the composition of a list data type. If the LIST is empty, the list shall be encoded as a *left parenthesis* ("(") followed by a *right parenthesis* (")"). Within the list, each instance of the element type shall be encoded as specified in clause 10 for that EXPRESS data type.

NOTE If, in a particular entity instance, no value is provided for an OPTIONAL attribute whose data type is a LIST, the attribute is encoded by a *dollar sign* ("$"), as specified in 10.2.2, and not as an empty list.

EXAMPLE 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.

10.1.3 Array

Values of the EXPRESS data type ARRAY shall be mapped to the exchange structure as a list data type. Clause 7 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 instances of the element type, 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. Within the list, each instance of the element type shall be encoded as specified in clause 10 for that EXPRESS data type. If the array data type has OPTIONAL elements, any element for which no value is provided shall be encoded by a *dollar sign* ("$").

NOTE If, in a particular entity instance, no value is provided for an OPTIONAL attribute whose data type is an ARRAY, the attribute is encoded by a *dollar sign* ("$"), as specified in 10.2.2, and not as an empty list.

EXAMPLE 1 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] )  )
```

EXAMPLE 2 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] = not provided
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
```

10.1.4 Set

Values of the EXPRESS data type SET shall be mapped to the exchange structure as a list data type. Clause 7 describes the composition of a list data type. Within the list, each instance of the element type shall be encoded as specified in clause 10 for that EXPRESS data type. If the SET is empty, the list shall be encoded as a *left parenthesis* ("(") followed by a *right parenthesis* (")").

NOTE If, in a particular entity instance, no value is provided for an OPTIONAL attribute whose data type is a SET, the attribute is encoded by a *dollar sign* ("$"), as specified in 10.2.2, and not as an empty list.

EXAMPLE 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.

10.1.5 Bag

Values of the EXPRESS data type BAG shall be mapped to the exchange structure as a list data type. Clause 7 describes the composition of a list data type. Within the list, each instance of the element type shall be encoded as specified in clause 10 for that EXPRESS data type. If the BAG is empty, the list shall be encoded as a *left parenthesis* ("(") followed by a *right parenthesis* (")").

NOTE If, in a particular entity instance, no value is provided for an OPTIONAL attribute whose data type is a BAG, the attribute is encoded by a *dollar sign* ("$"), as specified in 10.2.2, and not as an empty list.

EXAMPLE 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.

10.1.6 Simple defined types

A simple defined type is a type defined by an EXPRESS type declaration in which the underlying type is neither an ENUMERATION nor a SELECT. A simple defined type shall be mapped to the exchange structure as the data type used in its definition.

EXAMPLE 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.

10.1.7 Enumeration

Values of an EXPRESS ENUMERATION data type shall be mapped to the exchange structure as an enumeration data type. 6.3.5 describes the composition of a enumeration data type.

If the document that defines the schema whose instances are the subject of the data section also defines a set of short names for the enumerated values within that schema, the actual value in an instance of the ENUMERATION may be the short name corresponding to one of the enumerated values in the EXPRESS schema. Otherwise, the actual value shall be one of the enumerated values in the EXPRESS schema. In either case, any *small letters* shall be converted to the corresponding *capital letters*, and the value shall be delimited by *full stops* "." as defined in the ENUMERATION production of Table 2.

EXAMPLE 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.

10.1.8 Select data types

An EXPRESS select data type defines a list of data types, called the "select-list", whose values are valid instances of the select data type. An instance of a select data type shall be a value of at least one of the data types in the select-list. The value shall be encoded in the exchange structure as determined by the following procedure:

— if the value is an instance of an entity data type in the select-list, it shall be mapped to the exchange structure as an entity instance name (see 6.3.4);

— if the value is an instance of a simple defined type in the select-list, it shall be mapped to the exchange structure as a TYPED_PARAMETER (see Table 3) in which the KEYWORD shall identify the simple defined type, as specified below, and the PARAMETER shall be the encoding of the value of the simple defined type, as specified in 10.1.6;

— if the value is an instance of an enumeration data type in the select-list, it shall be mapped to the exchange structure as a TYPED_PARAMETER (see Table 3) in which the KEYWORD shall identify the enumeration data type, as specified below, and the PARAMETER shall be the encoding of the value of the enumeration data type, as specified in 10.1.7;

— if the value is an instance of a (nested) select data type in the select-list, it shall be mapped to the exchange structure as an instance of that select type as provided in this clause.

For instances of simple defined types and enumeration data types, the KEYWORD shall identify the data type of the instance. The data type identified shall be one of the types in the select-list. If the document that defines the schema whose instances are the subject of the data section also defines a set of short names for the simple defined types and enumeration types within that schema, the KEYWORD may be the short name corresponding to the data type of the instance. Otherwise, the KEYWORD shall be the name of the simple defined type or enumeration data type itself. In either case, any small letters shall be converted to the corresponding capital letters, i.e., the encoding shall not contain any small letters.

NOTE 1 If the data type (in the select-list) which the value instantiates is itself a select data type, then this clause will be used recursively to encode the value. Only instances of entity data types, simple defined types and enumeration data types can actually be encoded (see Example 2).

NOTE 2 According to ISO 10303-11:1994, an instance of a subtype of an entity data type is an instance of the entity data type. So "an instance of an entity data type in the select list" includes instances of subtypes of those entity data types.

NOTE 3 If the entity data types in the select-list are not mutually exclusive, then a value of the select data type may instantiate more than one entity data type in the select-list (see Example 1).

NOTE 4 If the value is not an entity instance, it is an instance of exactly one simple defined-type or enumeration data type. The value may, however, be a valid instance of several (nested) select data types and thereby instantiate more than one type in the original select-list (see Example 2).

EXAMPLE 1 Entity definition in EXPRESS:

```
ENTITY Leader SUBTYPE OF (Employee);
```

```
  project: STRING;
END_ENTITY;

ENTITY Manager SUBTYPE OF (Employee);
  unit: STRING;
END_ENTITY;

ENTITY Employee;
  name: STRING;
END_ENTITY;

TYPE Supervisor = SELECT (Manager, Leader);
END_TYPE;



ENTITY Meeting;
  date:         STRING;
  attendees:    SET [2:?] OF Supervisor;
END_ENTITY;
```

Sample data section instances:

```
#1 = LEADER('J. Brahms','Academic Festival');
#2 = MANAGER('S. Ozawa', 'Tokyo Symphony');
#3 = (EMPLOYEE('G. Verdi') LEADER('Aida') MANAGER('La Scala'));
#4 = MEETING('14921012', (#1, #2, #3));
```

The second attribute of instance #4 is the attendees: a SET OF Supervisor. Instance #1 is a Leader and thus a valid Supervisor. Instance #2 is a Manager and thus a valid Supervisor. Instance #3 is an instance of both (entity) types Leader and Manager from the select-list of Supervisor. All are mapped according to 6.3.4.

EXAMPLE 2 Entity definition in EXPRESS:

```
TYPE Mass = SELECT (Mass_Spec, Mass_Substitute);
END_TYPE;

TYPE Mass_Spec = SELECT (Measured_Mass, Computed_Mass, Estimated_Mass);
END_TYPE;

TYPE Measured_Mass = REAL;
END_TYPE;

TYPE Computed_Mass = Extended_Real;
END_TYPE;

TYPE Estimated_Mass = REAL;
END_TYPE;

TYPE Mass_Substitute = SELECT(Weight, Estimated_Mass);
END_TYPE;

TYPE Weight = REAL;
END_TYPE;

TYPE Extended_Real = SELECT (FloatingNumber, NotaNumber);
END_TYPE;

TYPE FloatingNumber = REAL(7);
END_TYPE;
```

```
TYPE NotaNumber = ENUMERATION OF (plus_infinity,
  minus_infinity, indeterminate, invalid);
END_TYPE;

ENTITY Steel_Bar;
  bar_length: Extended_Real;
  bar_mass:   Mass;
END_ENTITY;
```

Sample instantiation in data section:

```
#1 = STEEL_BAR(FLOATINGNUMBER(77.0), MEASURED_MASS(13.25));
#2 = STEEL_BAR(NOTANUMBER(.INDETERMINATE.),
     ESTIMATED_MASS(10.0));
#3 = STEEL_BAR(FLOATINGNUMBER(77.0),
     COMPUTED_MASS(FLOATINGNUMBER(14.77719)));
```

The first attribute of instance #1 represents an Extended_Real value that is a FloatingNumber. It is mapped (following 10.1.8 for the select data type Extended_Real) as a TYPED_PARAMETER with KEYWORD FLOATINGNUMBER (the simple defined type in the select-list). The PARAMETER value 77.0 is mapped to the exchange structure, following 10.1.6 for FloatingNumber, as a value of the simple type REAL.

The second attribute of instance #1 represents a Measured_Mass value, which is a valid Mass_Spec value and therefore a valid Mass value. It is mapped (by recursive application of 10.1.8, since Mass is a select data type and Mass_Spec is a select data type) as a TYPED_PARAMETER with KEYWORD MEASURED_MASS (the simple defined type in the select-list). The PARAMETER value 13.25 is mapped to the exchange structure, following 10.1.6 for Measured_Mass, as a value of the simple type REAL.

The first attribute of instance #2 represents an Extended_Real value that is a NotaNumber value. It is mapped (following 10.1.8 for Extended_Real) as a TYPED_PARAMETER with KEYWORD NOTANUMBER (the enumeration type in the select-list). The PARAMETER value "indeterminate" is mapped to the exchange structure, following 10.1.7 for the enumeration type NotaNumber.

The second attribute of instance #2 represents an Estimated_Mass value. This is a valid Mass_Spec value and also a valid Mass_Substitute value and therefore a valid value of Mass. This value actually instantiates both (select) data types in the select-list of Mass. It is mapped (by recursive application of 10.1.8, since Mass is a select data type and Mass_Spec is a select data type) as a TYPED_PARAMETER with KEYWORD ESTIMATED_MASS (the simple defined type in the select-list). The PARAMETER value 10.0 is mapped to the exchange structure, following clause 10.1.6 for Estimated_Mass, as a value of the simple type REAL.

The first attribute of instance #3 is the same as the first attribute of instance #1.

The second attribute of instance #3 represents a Computed_Mass value, which is a valid Mass_Spec value and therefore a valid Mass value. It is mapped (by recursive application of 10.1.8, since Mass is a select data type and Mass_Spec is a select data type) as a TYPED_PARAMETER with KEYWORD COMPUTED_MASS (the simple defined type in the select-list). The PARAMETER value encoding follows clause 10.1.6 for a value of Computed_Mass, which is an Extended_Real value. Extended_Real is a select data type. Per clause 10.1.8, the Extended_Real value is encoded as a TYPED_PARAMETER with KEYWORD FLOATINGNUMBER and PARAMETER value 14.77719, following clause 10.1.6 for FloatingNumber.

10.2 Mapping of EXPRESS entity data types

An instance of an EXPRESS entity data type shall be mapped to the exchange structure as an ENTITY_INSTANCE.

As defined by ISO 10303-11:1994, a "simple entity instance" is an entity instance that is not an instance of a subtype of any entity data type. All other entity instances are called "complex entity instances". A simple entity instance shall be mapped as specified in 10.2.1. A complex entity instance shall be mapped as specified in 10.2.5.

NOTE A simple entity instance is an entity instance which is completely described by a *single* EXPRESS entity-declaration. A complex entity instance is an instance whose description involves more than one entity-declaration, even when only one of them contains explicit attributes. A simple entity instance can be an instance of a supertype, as long as it is not an instance of any subtype, but an instance of a subtype is always "complex".

Only the explicit attributes of an EXPRESS entity are mapped to the exchange structure. Special provisions, however, apply to OPTIONAL explicit attributes (see 10.2.2), explicit attributes whose values are entity instances (see 10.2.4), and all redeclarations of explicit attributes (see 10.2.6, 10.2.7, and 10.2.8)

NOTE More than one such provision may apply to the same attribute.

10.2.1 Mapping of a simple entity instance

A simple entity instance shall be mapped as a SIMPLE_ENTITY_INSTANCE in the exchange structure. The entity data type name shall be mapped to the KEYWORD of the SIMPLE_RECORD, as specified in 10.2.11.

Each explicit attribute shall be mapped directly to a PARAMETER of the SIMPLE_RECORD in the exchange structure. The order of the PARAMETERs in the exchange structure shall be the same as the order of the corresponding attributes in the EXPRESS entity declaration. The first PARAMETER shall be the value of the first explicit attribute; the second PARAMETER shall be the value of the second explicit attribute, and so on. If the EXPRESS entity data type has no explicit attributes, the PARAMETER_LIST shall be empty.

The form of each PARAMETER shall depend on the data type of the corresponding attribute, as specified in 10.1.

EXAMPLE 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.

10.2.2 Mapping of OPTIONAL explicit attributes

An explicit attribute that is declared to be OPTIONAL is not required to have a value in a given entity instance. When the optional value is supplied in an entity instance, it shall be encoded according to the data type of the attribute, as specified in 10.1. 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 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.

10.2.3 Mapping of derived attributes

The derived attributes of an entity shall not be mapped to the exchange structure. When a derived attribute in a subtype redeclares an attribute in a supertype, the mapping used shall be as described in 10.2.6.

EXAMPLE 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.

10.2.4 Mapping of attributes whose values are entity instances

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 6.3.4). The referenced entity instance shall be defined within the data sections, i.e. somewhere within the data sections 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. The definition need not occur within the same data section as the use of the entity instance as an attribute.

NOTE Annex F describes methods for evaluating schema conformance when an exchange structure contains multiple data sections based on different EXPRESS schemas, including the validity of references among entity instances defined in data sections based on different EXPRESS schemas.

EXAMPLE 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 = XXX (#1, #3);
```

```
    #3 = YYY (1., 2., 3.);
```

10.2.5 Entities defined as subtypes of other entities

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.

Complex entity instances 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.

NOTE 1 The selection of mapping depends on the entity instance rather than the entity type. It is possible for different instances of the same entity data type to use different mappings, depending on whether they are instances of subtypes and which subtypes they instantiate.

NOTE 2 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. Such instances are mapped as provided in 10.2.1.

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 10.2.5.1. For implementations of conformance class 2, the external mapping prescribed by 10.2.5.3 shall be used for all complex entity instances.

10.2.5.1 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. Each member of the evaluated set specifies a list of entity data type names.

Each particular instance of an entity data type corresponds to one member of the evaluated set. The mapping that may be applied to a particular instance depends on the member of the evaluated set to which it corresponds.

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 that corresponds 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 member) 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 b above.

10.2.5.2 Internal mapping

If the internal mapping is used, the entity instance shall be mapped to a SIMPLE_ENTITY_INSTANCE (see Table 3). The KEYWORD shall encode the name of the leaf entity data type, as specified in 10.2.11. The PARAMETER_LIST shall encode the values of the inherited explicit 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.

EXAMPLE 1 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 .

EXAMPLE 2 An example of the mapping of a supertype that is not an ABSTRACT supertype. Entity definition in EXPRESS:

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

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

ENTITY cc SUBTYPE OF (bb);  -------------->  C
  attrib_c :  INTEGER;
END_ENTITY;

ENTITY dd SUBTYPE OF (aa);  -------------->  D
  attrib_d : REAL;
END_ENTITY;

ENTITY ee ;  ----------------------------->  E
  attrib_e : aa;
END_ENTITY;
```

Sample entity instance in data section:

```
#1 = AA('SAMPLE STRING');  --------------->  A
#2 = BB('ABC');  ------------------------->  B
#3 = CC('DEF', 123);   ------------------->  C
#4 = DD('XYZ', 99.99); ------------------->  D
#5 = EE(#1); ----------------------------->  E
#6 = EE(#2); ----------------------------->  E
#7 = EE(#3); ----------------------------->  E
#8 = EE(#4); ----------------------------->  E
```

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, but since entity bb does not define any attributes the parameter list will only contain attrib_a.

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

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

E: The entity ee references entity aa as an attribute. Therefore, an instance of entity ee may reference any of #1, #2, #3 or #4.

EXAMPLE 3 An example of the mapping of an entity with multiple supertypes in the SUBTYPE OF expression. Entity definition in EXPRESS:

```
ENTITY base SUPERTYPE OF (branch_one,branch_two);  --->  A
  attrib_a :  STRING;
END_ENTITY;

ENTITY branch_one SUBTYPE OF (base); ----------------->  B
  attrib_b :  INTEGER;
END_ENTITY;

ENTITY branch_two SUBTYPE OF (base);  ---------------->  C
  attrib_c :  BOOLEAN;
END_ENTITY;

ENTITY leaf SUBTYPE OF (branch_one, branch_two);  ---->  D
  attrib_d : REAL;
END_ENTITY;
```

Sample entity instance in data section:

```
#1 = BASE('SAMPLE STRING');  ------------------------->  A
#2 = BRANCH_ONE('ABC', 123);  ------------------------>  B
#3 = BRANCH_TWO('DEF', .T.);   ----------------------->  C
#4 = LEAF('XYZ', 123, .T., 99.99); ------------------->  D
```

A: Entity base has no supertypes. When instantiated in an exchange structure, its parameter list will contain only a value for the attrib_a attribute.

B: The entity branch_one is a subtype of base. When instantiated in an exchange structure, its parameter list will contain the inherited attributes of base followed by the attributes of branch_one.

C: The entity branch_two is a subtype of base. When instantiated in an exchange structure, its parameter list will contain the inherited attributes of base followed by the attributes of branch_two.

D: The entity leaf is a subtype of branch_one and branch_two. When instantiated in an exchange structure, its parameter list will contain the inherited attributes of branch_one, which includes the attributes of base, followed by the inherited attributes of branch_two, followed by the attributes of leaf. The attributes of base are only written once, while writing the attributes of branch_one. They are ignored when they are encountered a second time when writing the attributes of branch_two.

10.2.5.3 External mapping

If the external mapping is used, the entity instance shall be mapped to a COMPLEX_ENTITY_INSTANCE (see Table 3).

ISO 10303-11:1994 defines a "partial (complex) entity value" to be the set of attribute values described by a single EXPRESS entity declaration. Every entity data type name in the evaluated set member identifies a partial complex entity value of the entity instance. Thus, the evaluated set member identifies the set of partial complex entity values that, together with the entity instance name, completely describe the entity instance.

Every partial complex entity value identified by an entity data type name in the evaluated set member shall be mapped to a SIMPLE_RECORD within the SUBSUPER_RECORD. The order of SIMPLE_RECORDs within the SUBSUPER_RECORD shall be in ascending sequence of their entity data type names, using the collating sequence defined in 5.2.

Each SIMPLE_RECORD shall encode one partial complex entity value. In each SIMPLE_RECORD, the KEYWORD shall encode the corresponding entity data type name, as specified in 10.2.11, and the PARAMETER_LIST shall encode the values of the explicit attributes, if any, appearing in the corresponding entity declaration. The order of the PARAMETERs in the exchange structure shall be the same as the order of the corresponding attributes in the EXPRESS entity declaration. If the EXPRESS entity declaration contains no explicit attributes, the PARAMETER_LIST shall be empty. The form of each PARAMETER shall depend on the data type of the corresponding attribute, as specified in 10.1.

NOTE 1 The sequence of partial entity values (SIMPLE_RECORDs) is determined by the entity data type name (the so-called "long name") and not by the "short name", if provided, which may be used for the encoding.

NOTE 2 Every partial entity value in the evaluated set is required to appear, including supertypes which have no explicit attributes.

EXAMPLE 1 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.

EXAMPLE 2 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.

EXAMPLE 3 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.

EXAMPLE 4 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.

EXAMPLE 5 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.

10.2.6 Explicit attributes redeclared as DERIVEd

If a subtype entity redeclares an attribute of its supertype using the DERIVE clause and if the original attribute is an explicit attribute, the value of the original attribute in the supertype shall be encoded as an *asterisk* ("*").

EXAMPLE Entity definition in EXPRESS:

```
ENTITY point;
  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.

10.2.7 Attributes redeclared as INVERSE

If a subtype entity redeclares an attribute of its supertype using the INVERSE clause, there is no effect on the encoding. The redeclared attribute is not encoded in any way.

10.2.8 Attributes redeclared as explicit attributes

If a subtype entity redeclares an attribute of one of its supertypes as an explicit attribute, i.e. not in a DERIVE clause nor an INVERSE clause, there is no effect on the encoding.

The attribute value shall be encoded as an attribute of the supertype, as specified in 10.2.5, applying the mapping specified in clause 10 for the data type of the attribute in the supertype. The redeclared attribute shall be ignored, that is, it shall not be considered an attribute of the subtype for encoding purposes.

EXAMPLE Entity definition in EXPRESS:

```
ENTITY aaa SUPERTYPE OF (ONEOF (bbb, ccc));
  a1    : NUMBER;
  a2    : curve;
INVERSE
  a3    : SET OF mmm FOR m1;
END_ENTITY;

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

ENTITY ccc SUBTYPE OF (aaa);
  SELF\aaa.a2   : line;
INVERSE
  SELF\aaa.a3   : SET [1:2] OF mmm FOR m1;
END_ENTITY;

ENTITY curve;
  ...
END_ENTITY;

ENTITY line SUBTYPE OF (curve);
  ...
END_ENTITY;

ENTITY mmm;
  m1    : aaa;
END_ENTITY;
```

Sample instantiation in data section:

```
#1 = LINE(...);
```

```
      #2 = CURVE(...);
      #3 = BBB(1.0, #2, 0.5);
      #4 = CCC(1.5, #1);
```

For the instances #3 and #4 the encoding is the same as if there were no redeclared attributes in the entities bbb and ccc.

10.2.9 Entity local rules

Entity local rules, WHERE rules and UNIQUE rules, shall not be mapped to the exchange structure.

EXAMPLE 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.

10.2.10 Mapping of INVERSE attributes

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

10.2.11 Encoding of entity type names

If the document that defines the schema whose instances are the subject of a data section also defines a set of short names for each of the entity data types within that schema, these short names may 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.

10.3 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 8.2.3.

10.4 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.

10.5 Mapping of the EXPRESS element of RULE

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

10.6 Remarks

Remarks shall not be mapped to the exchange structure.

11 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 6.3.3. 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 G 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 define the preferred format for interchange:

— 12,7 mm (0.5 in) inch tape width;

- unlabelled;
- 9 track tape;
- 63 cpmm (1600 bpi) tape density;
- 4000 octet (8-bit byte) physical block size;
- blocks separated by inter-record gaps;
- last block followed by tape mark.

Other standard magnetic tape media, such as 246 cpmm (6250 bpi), may also be used.

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 media 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.
- the last line shall be padded, if necessary, with spaces after the special token "END-ISO-10303-21;" that terminates the exchange structure.

The means by which lines are delimited and end-of-file is represented are dependent on the operating system and are a matter of agreement between sender and receiver, subject to the restriction that the delimiters shall not make use of any character in the basic alphabet. Whatever their representation, line- and file-delimiters shall be ignored when processing 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 organized 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.

Annex B
(normative)

WSN notational conventions

The syntax of the exchange structure is defined in Wirth Syntax Notation (WSN) as published by Niklaus Wirth in Communications of the ACM, 20:11 (Nov 77), 822-823. The WSN consists of a set of productions or substitution rules. The element given on the left side of a production (i.e. before the equals sign) can be used to represent an occurrence of the pattern given on the right side. Elementary terms that appear only on the right side of such productions are called terminals. Elements that occur on the left side of a production are called non-terminals.

The notational conventions are given below. Table B.1 presents WSN defined in itself.

— a string in *capital letters* is an element of the language; the string is the name of the element (for convenience, *small letters* are used for undefined identifiers and commentary);

— any string enclosed in *quotation marks* is literally what is contained within the *quotation marks*. The one exception to this rule is a *quotation mark* within a literal. To accomplish this, the *quotation mark* is immediately repeated once. The sequence """" is interpreted as ", and the sequence "AB""C" is interpreted as AB"C.

— the *equals sign* "=" indicates a production. The element on the left is defined to be the combination of the elements on the right. Any *spaces* appearing between the elements of a production are meaningless unless they appear within a literal. A production is terminated by a *full stop* ".";

— *curly brackets* "{ }" indicates zero or more repetitions;

— *square brackets* "[ ]" indicate optional parameters;

— *vertical line* "|" indicates the logical OR;

— *parentheses* "(" ")" indicate priority operations. In particular, where they enclose elements separated by *vertical lines*, one of the elements is to be chosen in conjunction with any other operation.

EXAMPLE The sequence "A(B|C|D)" is equivalent to "AB|AC|AD"

Table B.1 - Wirth Syntax Notation (WSN) defined in itself

```
SYNTAX          = { PRODUCTION }.

PRODUCTION      = IDENTIFIER "=" EXPRESSION ".".

EXPRESSION      = TERM { "|" TERM }.

TERM            = FACTOR { FACTOR }.

FACTOR          = IDENTIFIER
                    | LITERAL
                    | "[" EXPRESSION "]"
                    | "(" EXPRESSION ")"
                    | "{" EXPRESSION "}" .

IDENTIFIER      = letter { letter }.

LITERAL         = """" character { character } """".
```

Annex C
(normative)

Information object registration

C.1 Document identification

In order to provide for unambiguous identification of an information object in an open system, the object identifier

{ iso standard 10303 part(21) version(3) }

is assigned to this part of ISO 10303. The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

C.2 Schema identification

In order to provide for unambiguous identification of the header_section_schema in an open information system, the object identifier

{ iso standard 10303 part(21) version(3) object(1) header-section-schema(1) }

is assigned to the header_section_schema schema (see clause 8). The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

Annex D
(normative)

Basic alphabet and graphic character set

The graphic characters used in the exchange structure are the graphic characters of columns 02-07 of the Latin-1 alphabet of ISO 8859-1.

NOTE The graphic characters are displayed in Table D.1. The top row contains the most significant four bits of this byte, the left-most column the least significant four bits.

Table D.1 - Character set used in the exchange structure

	0010 32	0011 48	0100 64	0101 80	0110 96	0111 112
0000 0		0	@	P	`	p
0001 1	!	1	A	Q	a	q
0010 2	"	2	B	R	b	r
0011 3	#	3	C	S	c	s
0100 4	$	4	D	T	d	t
0101 5	%	5	E	U	e	u
0110 6	&	6	F	V	f	v
0111 7	'	7	G	W	g	w
1000 8	(	8	H	X	h	x
1001 9	)	9	I	Y	i	y
1010 10	*	:	J	Z	j	z
1011 11	+	;	K	[	k	{
1100 12	,	<	L	\	l	|
1101 13	-	=	M	]	m	}
1110 14	.	>	N	^	n	~
1111 15	/	?	O	_	o	

Annex E
(normative)

Protocol Implementation Conformance Statement (PICS) proforma

The Protocol Information and Conformance Statement (PICS) Proforma supports the conformance assessment of implementations requesting evaluation to this part of ISO 10303. This annex is in the form of a questionnaire. This questionnaire is intended to be filled out by the implementor and may be used in preparation for conformance testing by a testing laboratory.

All implementors shall provide answers to the questions provided in E.1 and E.2.

E.1 Conformance to specified function

Check as many as are appropriate.

E.1.1 Entity instance encoding

Does the implementation support Conformance Class One?

___ for reading ___ for writing

Does the implementation support Conformance Class Two?

___ for reading ___ for writing

E.1.2 Short name encoding

Does the implementation support short names for entities

___ for reading ___ for writing

Does the implementation support short names for select type values

___ for reading ___ for writing

Does the implementation support short names for enumeration values

___ for reading ___ for writing

E.1.3 String encoding

Does the implementation support the \X\ encoding for 8-bit bytes?

___ for reading, and if so, what is the binary representation used by the implementation?

___ for writing

Does the implementation support the \S\ and \P\ encoding for ISO 8859 characters?

___ for reading, and if so, what is the binary representation used by the implementation?

___ for writing

Does the implementation support the \X2\ encoding for ISO 10646 characters?

___ for reading, and if so, what is the binary representation used by the implementation?

___ for writing

Does the implementation support the \X4\ encoding for ISO 10646 characters?

___ for reading, and if so, what is the binary representation used by the implementation?

___ for writing

E.2 Implementation limits

What is the maximum number of schemas that be referenced by an exchange structure?

What is the maximum number of data sections that may exist within an exchange structure?

What is the maximum number of entity instances that may exist within a data section?

What is the maximum number of entity instances that may exist within an exchange structure?

What is the maximum value (or binary representation used by the implementation) for entity instance identifiers that is supported?

What are the maximum and minimum values (or binary representation used by the implementation) for EXPRESS INTEGER that is supported?

What is the limit of EXPRESS REAL precision (or binary representation used by the implementation) that is supported?

What is the maximum length of EXPRESS STRING that is supported?

What is the maximum length of EXPRESS BINARY that is supported?

What is the maximum number of elements that may appear in the encoding of an aggregate?

What is the maximum nesting depth that may appear in the encoding of nested aggregates?

Annex F
(normative)

Multiple EXPRESS schemas in an exchange structure

An exchange structure with multiple data sections may have entity instance references among instances defined in data sections based on different EXPRESS schemas. These references across schemas raise two questions, 1) what references are to be considered valid and 2) which instances shall be considered as being based on a particular schema when determining the schema conformance of the exchange structure.

F.1 Reference validity

This subclause describes two methods of determining what references between entity instances are to be considered valid for determining schema conformance of the exchange structure. Other methods are possible but are not specified in this part of ISO 10303.

NOTE This section describes ways in which the authors of documents that define EXPRESS schemas may also define reference validity between two or more schemas.

When determining schema conformance, an implementation must refer to the document that defines the schema for the EXPRESS definition, short names, and other requirements or constraints. If the schema is to be used in combination with others, the document must also define what references are to be considered valid.

It is expected that this determination will be done once and will govern all uses of the multiple schemas, so no attempt is made to convey within a particular exchange structure the means by which reference validity was originally determined.

F.1.1 EXPRESS interface specification method

If this method is used, references between entity instances defined by different schemas shall be governed by the EXPRESS interface specification set forth in clause 11 of ISO 10303-11.

An instance of a type defined by a schema may be referenced by an instance of a type defined by a second schema if the first type is interfaced by the second schema using USE or REFERENCE constructs.

EXAMPLE Consider the following two schemas and the exchange structure based on them:

```
SCHEMA base;
ENTITY a;
  range : REAL;
END_ENTITY;
ENTITY b;
  name : STRING;
END_ENTITY;
```

```
END_SCHEMA;

SCHEMA extension;
USE FROM base (a, b);

ENTITY c;
  addressed_item : b;
  address : STRING;
END_ENTITY;

RULE a_range_positive FOR (a);
WHERE
  WR1: SIZEOF (QUERY (inst <* a | inst.range < 0)) = 0;
END_RULE;
END_SCHEMA;


ISO-10303-21;
HEADER;
.
.
FILE_SCHEMA(('BASE', 'EXTENSION'));
ENDSEC;
DATA ('ONE', ('BASE'));
#1=A(-3.5);
#2=B('Sam Smith');
#3=B('John Doe');
ENDSEC;
DATA ('TWO', ('EXTENSION'));
#4=C(#2, '100 Main Street');
#5=C(#3, '1300 Elmwood Avenue);
ENDSEC;
END-ISO-10303-21;
```

Using the EXPRESS interface specification method, when determining the validity of references, an implementation must consider that entity type B is explicitly interfaced by schema 'EXTENSION' so the references from #4 to #2 and #5 to #3 are permitted.

F.1.2 SDAI domain equivalence method

If this method is used, references between entity instances defined by different schemas shall be governed by the domain equivalence methods specified in ISO 10303-22.

Given an instance of a type defined by a schema, instances of any entity type in the schema that may reference instances of the first type may also reference instances of any type that is declared domain equivalent to the first type.

EXAMPLE Consider the following two schemas and the exchange structure based on them:

```
SCHEMA LONGA;
ENTITY a;
  range : REAL;
END_ENTITY;
ENTITY b;
  name : STRING;
END_ENTITY;
END_SCHEMA;
```

```
SCHEMA LONGB;
ENTITY a;
  range : REAL;
END_ENTITY;
ENTITY b;
  name : STRING;
END_ENTITY;

ENTITY c;
  addressed_item : b;
  address : STRING;
END_ENTITY;

RULE a_range_positive FOR (a);
WHERE
  WR1: SIZEOF (QUERY (inst <* a | inst.range < 0)) = 0;
END_RULE;
END_SCHEMA;


ISO-10303-21;
HEADER;
.
.
FILE_SCHEMA(('LONGA', 'LONGB'));
ENDSEC;
DATA ('ONE', ('LONGA'));
#1=A(-3.5);
#2=B('Sam Smith');
#3=B('John Doe');
ENDSEC;
DATA ('TWO', ('LONGB'));
#4=C(#2, '100 Main Street');
#5=C(#3, '1300 Elmwood Avenue);
ENDSEC;
END-ISO-10303-21;
```

For the purpose of this example, assume that we have domain equivalence information, determined using one of the techniques described in ISO 10303-22, that states types A and B are domain equivalent in both schemas. That is:

```
LONGA.A is DEQ to LONGB.A
LONGB.A is DEQ to LONGA.A

LONGA.B is DEQ to LONGB.B
LONGB.B is DEQ to LONGA.B
```

Using the domain equivalence, when determining the validity of references, an implementation must consider that entity type LONGA.B is domain equivalent to LONGB.B so the references from #4 to #2 and #5 to #3 are permitted.

F.2 Determining population of a schema

The **file_population** header section entity associates an EXPRESS schema and a collection of entity instances within the exchange structure. The attribute **determination_method** identifies an algorithm for selecting a collection of entity instances when given a set of data sections. This subclause describes three determination methods. Other methods are possible but are not specified in this part of ISO 10303.

When determining schema conformance of the exchange structure, the collection of entity instances identified by each **file_population** shall be checked against the associated EXPRESS schema. If a data section is not referenced by any **file_population**, that data section shall be checked against its governing schema according to the section boundary method described below.

When determining the satisfaction of requirements and constraints of a schema, references to entity instances outside of the collection of entity instances shall behave as unset values.

NOTE The following procedure describes an approach for checking the exchange structure that is consistent with contents of clause 8.2.4 and the paragraph above:

For each file_population "F" in the exchange structure:
— find a set of instances by applying the determination method identified by F.determination_method to the data sections called out by F.governed_sections. If F.governed_sections is unset, apply to all data sections in the exchange structure;
— check this set of instances against the rules and constraints defined by F.governing_schema;
— mark the data sections that were given as input to F.determination_method.

For each data section "D" that has not been marked:
— check the set of instances in D against the rules and constraints defined by its schema.

F.2.1 Section boundary method

The **determination_method** attribute shall have the value "SECTION_BOUNDARY" when the section boundary method is to be used. Given an input of one or more data sections, the collection of entity instances shall consist of the following:

— all instances in the given data sections.

EXAMPLE 1- Consider the schemas and exchange structure described by the example in clause F.1.1. The header section does not contain any file_population instances. When determining the schema conformance of the exchange structure, the following must be considered:

— data section ONE is not referenced by any file_population, so we must check all instances in the data section against its governing schema. All instances in data section ONE must satisfy the requirements and constraints of schema BASE. In this example, the population satisfies all constraints of schema BASE.

— data section TWO is not referenced by any file_population, so we must check all instances in the data section against its governing schema. All instances in data section TWO must satisfy the requirements and constraints of schema EXTENSION. In this example, the data section does not contain any instances of entity A, so rule a_range_positive is satisfied. The addressed_item attribute of instances #4 and #5 refer to instances outside of the population, so the references must be treated as if they were unset values. The addressed_item attribute is not optional, so this population does not satisfy the constraints of schema EXTENSION.

EXAMPLE 2 - Consider the schemas and exchange structure described by the example in clause F.1.1, but with a header section as shown below:

```
HEADER;
.
.
FILE_SCHEMA(('BASE', 'EXTENSION'));
FILE_POPULATION('BASE', 'SECTION_BOUNDARY', ('ONE'));
FILE_POPULATION('EXTENSION', 'SECTION_BOUNDARY', ('ONE','TWO'));
ENDSEC;
```

When determining the schema conformance of the exchange structure, the following must be considered:

— the first file_population defines a collection of instances that is governed by schema BASE. All instances in data section ONE must satisfy the requirements and constraints of schema BASE. In this example, the population satisfies all constraints of schema BASE.

— the second file_population defines a collection of instances that is governed by schema EXTENSION. All instances in data section ONE and data section TWO must satisfy the requirements and constraints of schema EXTENSION. In this example, rule a_range_positive is violated by instance #1, so this population does not satisfy the constraints of schema EXTENSION.

F.2.2 Include all compatible method

The **determination_method** attribute shall have the value "INCLUDE_ALL_COMPATIBLE" when the include all compatible method is to be used. Given an input of one or more data sections, the collection of entity instances shall consist of the following:

— all instances in the given data sections;

— all instances in other data sections, where the entity type of the instance is permitted to be referenced by instances defined by the governing schema of the population.

EXAMPLE - Consider the schemas and exchange structure described by the example in clause F.1.1, but with a header section as shown below:

```
HEADER;
.
.
FILE_SCHEMA(('BASE', 'EXTENSION'));
FILE_POPULATION('BASE', 'INCLUDE_ALL_COMPATIBLE', ('ONE'));
FILE_POPULATION('EXTENSION', 'INCLUDE_ALL_COMPATIBLE', ('TWO'));
ENDSEC;
```

When determining the schema conformance of the exchange structure, the following must be considered:

— the first file_population defines a collection of instances that is governed by schema BASE. This collection contains all instances in data section ONE. It must also contain instances in data section TWO that are permitted to be referenced by instances defined by schema BASE. In this example, there are none. However, if data section TWO contained instances of type A or B, they would be considered for constraint validation. In this example, the population satisfies all constraints of schema BASE.

— the second file_population defines a collection of instances that is governed by schema EXTENSION. This collection contains all instances in data section TWO. It also contains all instances of A and B from data section ONE because those types are permitted to be referenced by instances defined by schema EXTENSION. In this example, rule a_range_positive is violated by instance #1, so this population does not satisfy the constraints of schema EXTENSION.

F.2.3 Include referenced instance method

The **determination_method** attribute shall have the value "INCLUDE_REFERENCED" when the include referenced instance method is to be used. Given an input of one or more data sections, the collection of entity instances shall consist of the following:

— all instances in the given data sections;

— instances in other data sections, where the instance is referenced by an instance in the given data sections.

EXAMPLE - Consider the schemas and exchange structure described by the example in clause F.1.1, but with a header section as shown below:

```
HEADER;
.
.
FILE_SCHEMA(('BASE', 'EXTENSION'));
FILE_POPULATION('BASE', 'INCLUDE_REFERENCED', ('ONE'));
FILE_POPULATION('EXTENSION', 'INCLUDE_REFERENCED', ('TWO'));
ENDSEC;
```

When determining the schema conformance of the exchange structure, the following must be considered:

— the first file_population defines a collection of instances that is governed by schema BASE. This collection contains all instances in data section ONE. It must also contain instances in data section TWO that are referenced by instances in data section ONE. In this example, there are none. In this example, the population satisfies all constraints of schema BASE.

— the second file_population defines a collection of instances that is governed by schema EXTENSION. This collection contains all instances in data section TWO. It also contains instances #2 and #3 from data section ONE because those types are referenced by instances in data section TWO. In this example, instance #1 is not a member of the population, so rule a_range_positive is satisfied. This population satisfies all constraints of schema EXTENSION.

Annex G
(informative)

Guidelines for printing the exchange structure

The exchange structure allows the optional inclusion of directives that explicitly control a printed representation of the structure. Where such directives are not included but a printed representation of the exchange structure is to be created, the set of implicit print control guidelines specified in G.2 should be used. The explicit print control directives override the implicit print control guidelines.

G.1 Explicit print control directives

Explicit print control directives may be used in those cases where the sender needs accurate control of the printed appearance of the exchange structure.

The directive *reverse solidus capital letter N reverse solidus* "\N\" indicates that the first character immediately following the directive appears at the start of a new printed line. The directive *reverse solidus capital letter F reverse solidus* "\F\" indicates that the printing continues at the start of a new page. In either case, the print control directive itself should not be printed.

NOTE The printing of the exchange structure will normally not be controlled by the explicit or implicit print control directives. It is assumed that the print control directives will only be used in those circumstances where the sender requires control over the printed appearance of the exchange structure. This situation may occur if the exchange structure is part of a legal contract.

G.2 Implicit print control directives

When printing the exchange structure, the following directives may be used in the absence of explicit print control directives:

a) printed lines should be left justified;

b) each section should begin at the start of a new printed line;

c) header section entities should begin at the start of a new printed line;

d) an entity instance name preceding the *equals sign* "=" should begin on a new printed line;

e) tokens other than strings and binaries should not be broken across printed lines. Strings and binaries should be broken only if they do not fit on a single printed line. Directive h) determines where strings and binaries should be broken;

f) comments should begin on a new printed line;

g) a token separator other than a comment should not be broken across printed lines. A comment will be broken if it does not fit on a printed line. Directive h) determines where a comment will be broken;

h) printed lines should be at most 72 characters long. If a character would be printed in the 73rd position of a line, this character should be printed on the first position of a new printed line instead.

Annex H
(informative)

Example of a complete exchange structure

H.1 Introduction

An example EXPRESS schema definitions, a table of short names and an exchange structure are presented below. This EXPRESS schema does not reflect the contents of any part of ISO 10303.

H.2 Example schema

The EXPRESS schema definitions used for the exchange structure example.

```
SCHEMA example_geometry;

TYPE length_measure = NUMBER;
END_TYPE;

ENTITY geometry
SUPERTYPE OF (ONEOF(point));
END_ENTITY;

ENTITY point
SUPERTYPE OF (ONEOF(cartesian_point))
SUBTYPE OF (geometry);
END_ENTITY;

ENTITY cartesian_point
SUBTYPE OF (point);
  x_coordinate  : length_measure;
  y_coordinate  : length_measure;
  z_coordinate  : OPTIONAL length_measure;
END_ENTITY;

TYPE edge_or_logical = SELECT (edge, edge_logical_structure);
END_TYPE;

ENTITY topology
SUPERTYPE OF (ONEOF(vertex, edge, loop));
END_ENTITY;

ENTITY vertex
SUBTYPE OF (topology);
  vertex_point : OPTIONAL point;
END_ENTITY;

ENTITY edge
```

```
SUBTYPE OF (topology);
  edge_start   : vertex;
  edge_end     : vertex;
END_ENTITY;

ENTITY edge_logical_structure;


  edge_element : edge;
  flag : BOOLEAN;
END_ENTITY;

ENTITY loop
SUPERTYPE OF (ONEOF(edge_loop))
SUBTYPE OF (topology);
END_ENTITY;

ENTITY edge_loop
SUBTYPE OF (loop);
  loop_edges : LIST [1:?] OF edge_or_logical;
END_ENTITY;

END_SCHEMA;
```

H.3 Example short names

The following gives the short names for the schema above.

Entity name	Short name
cartesian_point	cpt
vertex	vx
edge	ed
edge_logical_structure	ed_strc
edge_loop	ed_loop

H.4 Example exchange structure

The following is an example of a complete exchange structure. Note that since the schema is only an example, there is no schema registration id present in the schema_name attribute of the **file_schema** entity instance in the header section.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('THIS FILE CONTAINS A SMALL SAMPLE STEP MODEL'),'3;1');
FILE_NAME('EXAMPLE STEP FILE #1',
'1992-02-11T15:30:00',
('JOHN DOE',
'ACME INC.',
'METROPOLIS USA'),
('ACME INC. A SUBSIDIARY OF GIANT INDUSTRIES','METROPOLIS USA'),
'CIM/STEP VERSION2',
'SUPER CIM SYSTEM RELEASE 4.0',
'APPROVED BY JOE BLOGGS');
FILE_SCHEMA(('EXAMPLE_GEOMETRY'));
ENDSEC;
DATA;
/*
    THE FOLLOWING 13 ENTITIES REPRESENT A TRIANGULAR EDGE LOOP
*/
#1=CPT(0.0,0.0,0.0);    /* THIS IS A CARTESIAN POINT ENTITY */
#2=CPT(0.0,1.0,0.0);
#3=CPT(1.0,0.0,0.0);
#11=VX(#1);             /* THIS IS A VERTEX ENTITY */
#12=VX(#2);
#13=VX(#3);
#16=ED(#11,#12);        /* THIS IS AN EDGE ENTITY */
#17=ED(#11,#13);
#18=ED(#13,#12);
#21=ED_STRC(#17,.F.);   /* THIS IS AN EDGE LOGICAL STRUCTURE ENTITY */
#22=ED_STRC(#18,.F.);
#23=ED_STRC(#16,.T.);
#24=ED_LOOP((#21,#22,#23));   /* THIS IS AN EDGE LOOP ENTITY */
/*
    OTHER SYNTACTICAL REPRESENTATIONS WERE POSSIBLE. THE PREVIOUS
    EXAMPLE IS REPRESENTATIVE OF ONE POSSIBLE APPROACH.
*/
ENDSEC;
END-ISO-10303-21;
```

NOTE This example exchange structure has been edited to aid readability. Unnecessary *spaces* have been added to aid readability.

Index

ICS 25.040.40

Price based on 72 pages