9 Object Serialization Early Binding (OSEB)

This clause specifies a mapping from an EXPRESS schema to an early bound XML markup declaration set called the Object Serialization Early Binding or OSEB.

This clause is organized such that the markup declarations derived from an EXPRESS schema are specified in 9.2 through 9.7. The use of the markup to represent data is specified in 9.8 and 9.9.

An OSEB document is capable of being mapped to a late bound document as defined in Annex B via an Extensible Style Language Transformation (XSLT) stylesheet.

NOTE - “Object serialization” entails both the process of streaming objects into data structures and the process of creating objects from such data structures. Object serialization can be used whenever objects are written to or read from flat files, relational database tables, network transfer buffers, and so forth. An algorithm describing the generation of this stylesheet is defined in Annex H.

9.1 Fundamental Concepts and Assumptions

This subclause specifies the OSEB fundamental concepts and assumptions.

9.1.1 Preconditions

The creation of OSEB XML markup declarations from an EXPRESS schema assumes that the schema meets the following preconditions:

— syntactically correct EXPRESS schema.

9.1.2 Unmapped EXPRESS concepts

The OSEB does not map the following EXPRESS features:

— RULE declarations (see 9.2.2 for how RULE declarations affect the markup);

— FUNCTION declarations;

— PROCEDURE declarations;

— CONSTANT declarations;

— Remarks.

9.2 OSEB element naming

9.2.1 Mapping EXPRESS identifiers to XML

Unless otherwise specified, every EXPRESS identifier that appears in the OSEB markup declarations shall be mapped to an XML name, or a part of an XML name, as follows:

The XML name shall be the same as the EXPRESS identifier except that the first character of the XML name shall be in upper case and all other characters shall be in lower case, as specified in 4.4.2

Due to the XML naming restrictions specified in 4.4.1, any EXPRESS entity data type or defined data type beginning with the characters “xml” ,regardless of case, shall map to an XML name beginning with the characters “X-m-l”.

NOTE - This subclause applies to most EXPRESS defined data type names, entity names, attribute names, and enumerated item names. Exceptions for certain situations are specified in the subclauses dealing with particular kinds of EXPRESS objects.

9.2.2 Names resulting from EXPRESS named data types

Every EXPRESS schema shall have an associated XML namespace URI.

Each EXPRESS data type name shall be mapped to a qualified name as defined in the W3C Namespaces in XML Recommendation. The local part of the qualified name shall be the EXPRESS data type identifier mapped as specified in 9.2.1. The fully qualified name shall appear in the element type declaration and be used to represent each XML instance if such qualification is required to eliminate name clashes in an XML document. The non-local part of the qualified name need not be used if there is no name clash.

For every EXPRESS named data type declared in a schema, the associated XML namespace for the qualified name shall be that of the schema.

NOTE - For the non-local part of the URI, see Annex E for an approach to standardized URIs for use with EXPRESS schemas defined in other parts of ISO 10303 and other ISO standards.

For elements that represent types interfaced using the REFERENCE FROM specification, the associated XML namespace for the qualified name shall be the XML namespace of the interfaced schema.

For elements that represent types interfaced using the USE FROM specification:

— if the data type is restricted by an EXPRESS global rule in the interfacing schema, the associated XML namespace for the qualified name shall be the XML namespace of the interfacing schema, and the local part of the qualified name shall be derived from the identifier for the data type in the interfacing schema;

— otherwise the associated XML namespace for the qualified name shall be the XML namespace of the interfaced schema and the local part of the qualified name shall be derived from the identifier for the data type in the interfaced schema.

NOTE - In the latter case, renaming does not affect the local part of the qualified name.

EXAMPLE 1 - The following illustrates a mapping of the EXPRESS REFERENCE FROM schema interface specification.

In EXPRESS:

```
SCHEMA BB;
   REFERENCE FROM A (e as x);
   REFERENCE FROM A (f);
 USE FROM A (g);
   ENTITY e;
   END_ENTITY;
END_SCHEMA;
```

As markup declarations:

```
 <!ELEMENT xyz:E EMPTY> ...
 <!ELEMENT xyz:F EMPTY> ...
 <!ELEMENT xyz:G EMPTY> ...
 <!ELEMENT E EMPTY>  ...
```

In XML instance data:

```
<uos c=“id10 id20 id30 id40”
     xmlns = " urn:iso10303-28:oseb /BB"
     xmlns:xyz = " urn:iso10303-28:oseb /A">

     <E x-id="id10"/>
     <xyz:E x-id="id20"/>
     <xyz:F x-id="id30"/>
     <xyz:G x-id="id40"/>
</uos>
```

EXAMPLE 2 - The following illustrates a mapping of the EXPRESS USE FROM schema interface specification.

In EXPRESS:

```
SCHEMA b;
  USE FROM a  (e as x);      -- maps to ...schema-b:x
  USE FROM a (f);            -- maps to ...schema-b:f
  ENTITY e; END_ENTITY;      -- maps to ...schema-b:e
  RULE z FOR (x, f, e);
```



```
      WHERE wr1 : (* rule constraining entities in schema b *);
    END RULE;
  END_SCHEMA;
```

As markup declarations:

```
<!ELEMENT X EMPTY> ...
<!ELEMENT F EMPTY>...
<!ELEMENT E EMPTY>....
```

In XML instance data:

```
<uos c="id20 id30 id10"
     xmlns = " urn:iso10303-28:oseb /b">

  <X x-id="id20"/>
  <F x-id="id30"/>
  <E x-id="id10"/>

</uos>
```

9.3 EXPRESS schema-independent element types

This subclause specifies the XML markup declarations that shall appear in every OSEB markup declaration set. They are independent of the EXPRESS schema.

Each XML element described in this subclause shall be considered to belong to the XML namespace designated as: urn:iso10303-28:oseb.

9.3.1 Elements for simple types

The markup declaration set shall contain the following XML markup declaration for EXPRESS NUMBER:

```
<!ELEMENT number EMPTY>
<!ATTLIST number
  x-id  ID     #REQUIRED
  val   CDATA  #REQUIRED>
```

The markup declaration set shall contain the following declaration for EXPRESS BOOLEAN:

```
<!ELEMENT boolean EMPTY>
<!ATTLIST boolean
  x-id  ID  #REQUIRED
  val  (true | false | 0 | 1)  #REQUIRED>
```

The markup declaration set shall contain the following declaration for EXPRESS LOGICAL:

```
<!ELEMENT logical EMPTY>
<!ATTLIST logical
  x-id  ID   #REQUIRED
```



```
    val  (true | false | unknown)  #REQUIRED>
```

The markup declaration set shall contain the following declaration for EXPRESS STRING:

```
<!ELEMENT string (#PCDATA)>
<!ATTLIST string
  x-id ID  #REQUIRED>
```

The markup declaration set shall contain the following declaration for EXPRESS INTEGER:

```
<!ELEMENT long EMPTY>
<!ATTLIST long
  x-id  ID    #REQUIRED
  val   CDATA #REQUIRED>
```

The markup declaration set shall contain the following declaration for EXPRESS REAL:

```
<!ELEMENT double EMPTY>
<!ATTLIST double
  x-id ID    #REQUIRED
  val  CDATA #REQUIRED>
```

The markup declaration set shall contain the following declarations for simple type EXPRESS BINARY, where the content data of the binary XML elements shall be encoded as specified in 7.4.1.5:

```
<!ELEMENT hex-binary (#PCDATA)>
<!ATTLIST hex-binary
  x-id  ID  #REQUIRED
  notation   (hex)  #REQUIRED>

<!ELEMENT base64-binary (#PCDATA)>
<!ATTLIST base64-binary
  x-id  ID  #REQUIRED
  notation (base64)  #REQUIRED>
```

9.3.2 Elements for aggregate types

The markup declaration set shall contain the following declaration for an EXPRESS aggregate (SET, BAG, ARRAY, or LIST) of EXPRESS NUMBER data type or a defined data type with EXPRESS NUMBER as its underlying type.

```
<!ELEMENT nctn EMPTY>
<!ATTLIST nctn
    x-id  ID     #REQUIRED
    c     CDATA  #REQUIRED>
```

NOTE - The convention for the names of these elements is that the string “ctn” is short for the word “collection” and prepended letters signify the data type of the collection.

The markup declaration set shall contain the following declaration for an EXPRESS aggregate (SET, BAG, ARRAY, or LIST) of EXPRESS INTEGER data type or a defined data type with EXPRESS INTEGER as its underlying type.

```
<!ELEMENT lctn EMPTY>
<!ATTLIST lctn
    x-id  ID     #REQUIRED
    c     CDATA  #REQUIRED>
```

The markup declaration set shall contain the following declaration for an EXPRESS aggregate (SET, BAG, ARRAY, or LIST) of EXPRESS REAL data type or a defined data type with EXPRESS REAL as its underlying type.

```
<!ELEMENT dctn EMPTY>
<!ATTLIST dctn
    x-id  ID     #REQUIRED
    c     CDATA  #REQUIRED>
```

The markup declaration set shall contain the following declaration for an EXPRESS aggregate (SET, BAG, ARRAY, or LIST) of EXPRESS BOOLEAN data type or a defined data type with EXPRESS BOOLEAN as its underlying type.

```
<!ELEMENT bctn EMPTY>
<!ATTLIST bctn
    x-id  ID     #REQUIRED
    c     CDATA  #REQUIRED>
```

The markup declaration set shall contain the following declaration for an EXPRESS aggregate (SET, BAG, ARRAY, or LIST) of EXPRESS LOGICAL data type or a defined data type with EXPRESS LOGICAL as its underlying type.

```
<!ELEMENT lbctn EMPTY>
<!ATTLIST lbctn
    x-id  ID      #REQUIRED
    c     CDATA  #REQUIRED>
```

The markup declaration set shall contain the following declaration for an EXPRESS aggregate (SET, BAG, ARRAY, or LIST) of EXPRESS ENUMERATION type.

```
<!ELEMENT ectn EMPTY>
<!ATTLIST ectn
    x-id  ID      #REQUIRED
    c     CDATA  #REQUIRED>
```

The markup declaration set shall contain the following declaration for an EXPRESS aggregate (SET, BAG, ARRAY, or LIST) of the following EXPRESS types: EXPRESS entity data types, STRING, BINARY, SELECT, SET, BAG, ARRAY, LIST and defined types whose underlying type is STRING, BINARY, SET, BAG, ARRAY or LIST. This element shall also represent an ARRAY of OPTIONALs for any aggregated EXPRESS type.

```
<!ELEMENT ctn EMPTY>
<!ATTLIST ctn
```



```
    x-id      ID      #REQUIRED
    ctype     CDATA   #REQUIRED
    c         IDREFS  #REQUIRED>
```

9.3.3 The unset element

The markup declaration set shall contain the following declaration for the `unset` element:

```
<!ELEMENT unset EMPTY>
<!ATTLIST unset
 x-id  ID  #REQUIRED >
```

The `unset` element represents an unset instance of any type in a sparse collection (see 9.9.10.7).

9.3.4 The uos element

The `uos` element represents a unit of serialization (see 9.8.1). It is the OSEB representation of a schema instance in the XML document. The unit of serialization element `uos` shall be represented in the markup declaration set as follows:

```
<!ELEMENT uos ANY>
<!ATTLIST uos
  c        IDREFS  #REQUIRED
  unset    IDREF   #IMPLIED
  schema   NMTOKEN #IMPLIED>
```

9.3.5 The external identification elements

The `edokey` element provides for the external identification of a data set within the `uos`. The usage of these elements is specified in 9.8.1.1. The `edokey`, `this`, `lockey`, and `go` element type declarations shall appear in the markup declaration set as follows:

```
<!ELEMENT edokey (this?, lockey*, go*)>
<!ATTLIST edokey
  x-id        ID     #REQUIRED
  ob          IDREF  #IMPLIED
  owner       IDREF  #IMPLIED
  xml:base    CDATA  #IMPLIED
  xsi:type    CDATA  #IMPLIED
  xlink:type  (simple| extended | locator) #IMPLIED
  xlink:href  CDATA  #IMPLIED
  xlink:title CDATA  #IMPLIED
  xlink:role  CDATA  #IMPLIED>

<!ELEMENT this (#PCDATA)>
<!ATTLIST this
  xlink:type  (resource) #REQUIRED
  xlink:role  CDATA #REQUIRED
  xlink:title CDATA #IMPLIED>

<!ELEMENT lockey (#PCDATA)>
```



```
<!ATTLIST lockey
  ob          IDREF     #IMPLIED
  owner       IDREF     #IMPLIED
  xml:base    CDATA     #IMPLIED
  xsi:type    CDATA     #IMPLIED
  xlink:type  (locator) #IMPLIED
  xlink:href  CDATA     #REQUIRED
  xlink:title CDATA     #REQUIRED
  xlink:role  CDATA     #REQUIRED>

<!ELEMENT go EMPTY>
<!ATTLIST go
  xlink:type    (arc) #REQUIRED
  xlink:from    CDATA #REQUIRED
  xlink:to      CDATA #REQUIRED
  xlink:role    CDATA #REQUIRED
  xlink:actuate (onLoad|onRequest|undefined)  #IMPLIED
  xlink:show    (new|replace|embed|undefined) #IMPLIED>
```

9.4 XML declarations for EXPRESS named data types

This subclause specifies the XML markup declarations that represent the definition of data types that are defined in the EXPRESS context schema. Each XML element in this subclause shall belong to an XML namespace as specified in 9.2.2.

9.4.1 EXPRESS ENUMERATION types

For each EXPRESS ENUMERATION data type in the context schema, the markup declaration set shall contain a corresponding XML element and attribute list declaration. The name of the XML element shall be derived from the EXPRESS data type identifier as specified in 9.2. The content model of the XML element shall be EMPTY. The ATTLIST declaration shall contain the following XML attributes:

— a #REQUIRED XML attribute named `x-id`, of type ID;

— a #REQUIRED XML attribute named `val`, with an XML type of "(enumeration_item<1> | … | enumeration_item<n>)", where "enumeration_item<n>" is the nth identifier in the ordered list of enumeration items in the EXPRESS type declaration. Each enumeration item name shall be represented in lowercase.

EXAMPLE - The following example illustrates the mapping of EXPRESS ENUMERATION into its corresponding XML markup declaration.

In EXPRESS:

```
TYPE flower_colour
   = ENUMERATION OF (red, yellow, white);
END_TYPE;
```

As markup declarations:


```
<!ELEMENT Flower_colour EMPTY >
<!ATTLIST Flower_colour
 x-id ID #REQUIRED
 val (red | yellow | white)  #REQUIRED>
```

9.4.2 EXPRESS SELECT types

For each EXPRESS SELECT data type in the context schema, the markup declaration set shall contain a corresponding XML element and attribute list declaration. The name of the XML element shall be derived from the EXPRESS data type identifier as specified in 9.2.2. The content model of the XML element shall be EMPTY. The ATTLIST declaration for the XML element shall contain the following XML attributes:

— a #REQUIRED XML attribute named `x-id`, of type ID;

— a #REQUIRED XML attribute named `utype`, of type NMTOKEN;

— a #REQUIRED XML attribute named `val`, of type IDREF.

EXAMPLE - The following example illustrates the mapping of EXPRESS SELECT into its corresponding XML markup.

In EXPRESS:

```
TYPE efficiency = SELECT (efficiency_measure, efficiency_range);
END_TYPE;
```

As markup declarations:

```
<!ELEMENT Efficiency EMPTY>
<!ATTLIST Efficiency
 x-id  ID  #REQUIRED
 utype  NMTOKEN  #REQUIRED
 val   IDREF  #REQUIRED>
```

9.4.3 EXPRESS entity data types

9.4.3.1 Instantiable entity data types

For each EXPRESS entity data type not declared to be ABSTRACT, the markup declaration set shall contain an XML element type declaration corresponding to the EXPRESS entity data type. The name of the XML element shall be derived from the EXPRESS data type identifier as specified in 9.2.2. The content model of the XML element type declaration shall be EMPTY.

The markup declaration set shall contain an XML ATTLIST declaration that corresponds to the XML element type declaration.

The XML ATTLIST declaration shall contain a #REQUIRED XML attribute of type ID named `x-id`.

For each EXPRESS attribute defined in the EXPRESS entity declaration, the XML ATTLIST declaration shall contain one corresponding XML attribute.

For each EXPRESS local rule defined in the EXPRESS entity declaration, the XML ATTLIST declaration shall contain one corresponding XML attribute.

If the EXPRESS entity declaration contains a SUBTYPE clause with a list of "supertypes", the entity data type is said to *inherit* the EXPRESS attributes and rules of all its supertypes. An entity data type inherits not only the EXPRESS attributes and rules appearing in the EXPRESS entity declarations of all of its supertypes, but also all the EXPRESS attributes and rules *they* inherit. The XML ATTLIST declaration for an entity data type shall also include one XML attribute for each EXPRESS attribute or local rule that the entity data type inherits.

The mapping of the EXPRESS attributes into attributes in the XML ATTLIST declaration shall be as specified in 9.5. The mapping of the EXPRESS rules into attributes in the XML ATTLIST declaration shall be as specified in 9.7. The order of the XML attributes shall not be significant.

EXAMPLE 1 - The following example illustrates the mapping of an EXPRESS ENTITY into its corresponding XML markup.

In EXPRESS:

```
ENTITY action_method;
name : INTEGER;
num : REAL;
END_ENTITY;
```

As markup declarations:

```
<!ELEMENT Action_method     EMPTY>
<!ATTLIST Action_method
    x-id  ID  #REQUIRED
    NameCDATA #REQUIRED
    Num CDATA #REQUIRED>
```

EXAMPLE 2 - The following example illustrates the mapping of an EXPRESS ENTITY with supertype attributes into its corresponding XML markup.

In EXPRESS:

```
ENTITY point;
x : REAL;
y : REAL;
END_ENTITY;

ENTITY cartesian_point
SUBTYPE OF (point);
z : REAL;
END_ENTITY;
```


As markup declarations:

```
<!ELEMENT Point EMPTY>
<!ATTLIST Point
x-id  ID  #REQUIRED
X  CDATA  #REQUIRED
Y  CDATA  #REQUIRED>

<!ELEMENT Cartesian_point EMPTY>
<!ATTLIST Cartesian_point
x-id  ID  #REQUIRED
X  CDATA  #REQUIRED
Y  CDATA  #REQUIRED
Z  CDATA  #REQUIRED>
```

9.4.3.2 Complex entity data types

A set of EXPRESS entity data type definitions that are linked by subtype relationships 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 types that may be simultaneously represented in a single instance. When this list contains more than one entity data type, the evaluated set member is said to represent a "complex entity data type", and the entity data types in the list are said to be its "partial entity data types". Within this list, an entity data type that has no subtypes, or has no subtype that also appears in the list, is referred to as a "leaf type" for that complex entity data type.

NOTE 1 - Every evaluated set member has at least one leaf type.

When an EXPRESS complex entity data type has only one leaf type, it is said to be *characterized* by the leaf type. That is, every EXPRESS entity instance that instantiates exactly the partial entity data types of the complex entity data type is an instance of the leaf type, and has exactly the properties of the leaf type. When the EXPRESS complex entity data type has more than one leaf type, the complex entity data type, and all corresponding EXPRESS entity instances, are said to be *uncharacterized* in the EXPRESS schema.

For every uncharacterized EXPRESS complex entity data type that is to be instantiated in the uos, the markup declaration set shall contain a corresponding XML element type declaration. The markup declaration set may, but need not, contain XML element type declarations corresponding to uncharacterized EXPRESS complex entity data types that are not instantiated in the uos.

NOTE 2 - The use of ISO 10303-11 annex B in this clause is solely to define characterized and uncharacterized entity instances. It specifies one way of determining the set of potential entity instance types that are implied by a EXPRESS schema.

NOTE 3 - The determination of which complex entities are to be included in the markup declaration set may be the result of a processor evaluating the complete set of potentially instantiable complex entity data types and then applying a defined convention to limit the size of the set. One such convention is the use of an addendum to the EXPRESS file which is described in ISO 10303-23:1997:5.3.13.9.

The XML element for an uncharacterized EXPRESS complex entity data type shall be declared as follows.

— The XML element type name shall be the concatenation of the EXPRESS identifiers for the leaf types, with the first letter of each leaf type identifier in upper case and the remaining letters in lower case. The converted EXPRESS identifiers shall appear in order according to the following collating sequence: end-of-identifier, then digits (0-9) in ascending numerical order, then upper case letters (A-Z) in ascending alphabetical order, then the LOW LINE mark (underscore), then the lower case letters (a-z) in ascending alphabetical order.

— The content model of the XML element shall be EMPTY.

— The XML element type declaration shall have a corresponding XML attribute list declaration that contains a #REQUIRED XML attribute of type ID named `x-id`.

— For each attribute of every EXPRESS partial entity data type of the EXPRESS complex entity data type, there shall be one XML attribute declaration derived from the EXPRESS attribute as specified in 9.4.

— For each named rule appearing in the entity declaration of any EXPRESS partial entity data type of the EXPRESS complex entity data type, there shall be one XML attribute declaration derived from the EXPRESS rule as specified in 9.7.

— The order of the XML attributes shall not be significant.

EXAMPLE - The following example illustrates the mapping of complex entities into its corresponding XML markup.

In EXPRESS:

```
SCHEMA sample;
ENTITY application_context_element
SUPERTYPE OF (product_context ANDOR product_definition_context);
name : INTEGER;
END_ENTITY;

ENTITY product_context
SUBTYPE OF (application_context_element);
discipline_type : INTEGER;
END_ENTITY;

ENTITY product_definition_context
SUBTYPE OF (application_context_element);
life_cycle_stage : INTEGER;
END_ENTITY;
END_SCHEMA;
```

As markup declarations:

```
<!ELEMENT Product_contextProduct_definition_context EMPTY>
<!ATTLIST Product_contextProduct_definition_context
   x-id   ID    #REQUIRED
   Name   CDATA  #REQUIRED
   Discipline_type  CDATA  #REQUIRED
   Life_cycle_stage CDATA   #REQUIRED>
```


9.4.3.3 EXPRESS ABSTRACT entity data types

EXPRESS ABSTRACT entity data types shall have no corresponding XML markup declaration.

EXAMPLE - The following is an example of how EXPRESS ABSTRACT supertypes of EXPRESS instances are managed in this binding.

In EXPRESS:

```
SCHEMA sample;
ENTITY closed_planar_curve
ABSTRACT SUPERTYPE;
area : REAL;
END_ENTITY;

ENTITY circle
SUBTYPE OF (closed_planar_curve);
radius : REAL;
END_ENTITY;
END_SCHEMA;
```

As markup declarations:

```
<!ELEMENT Circle EMPTY>
<!ATTLIST Circle
  x-id   ID    #REQUIRED
  Area   CDATA #REQUIRED
  Radius CDATA #REQUIRED>
```

9.4.4 EXPRESS non-constructed defined data types

A *non-constructed defined data type* is an EXPRESS defined data type whose final underlying type is not a SELECT type or an ENUMERATION type.

For each EXPRESS defined data type with a final underlying type of EXPRESS INTEGER, REAL, LOGICAL, NUMBER or BOOLEAN in the context schema, the markup declaration set shall contain a corresponding XML element and attribute list declaration. The name of the XML element shall be derived from the EXPRESS data type identifier as specified in 9.2.2. The content model of the XML element shall be EMPTY. The ATTLIST declaration for the XML element shall contain the following XML attributes:

— a #REQUIRED attribute named `x-id`, of type ID;

— a #REQUIRED attribute named `val`, of type CDATA.

EXAMPLE 1 - The following example illustrates the mapping of EXPRESS TYPE into its corresponding XML markup.

In EXPRESS:

```
TYPE area = REAL;
END_TYPE;
```

As XML markup declarations:

```
<!ELEMENT Area EMPTY>
<!ATTLIST Area
  x-id ID    #REQUIRED
  val  CDATA #REQUIRED>
```

For each EXPRESS defined data type with a final underlying type of EXPRESS STRING in the context schema, the markup declaration set shall contain a corresponding XML element and attribute list declaration. The name of the XML element shall be derived from the EXPRESS defined data type identifier as specified in 9.2.2. The content model of the XML element shall be #PCDATA. The ATTLIST declaration for the XML element shall contain the following XML:

— a #REQUIRED attribute named `x-id`, of type ID.

EXAMPLE 2 - The following example illustrates the mapping of EXPRESS TYPE into its corresponding XML markup.

In EXPRESS:

```
TYPE len = STRING;
END_TYPE;
```

As XML markup declarations:

```
<!ELEMENT Len (#PCDATA)>
<!ATTLIST Len
 x-id  ID  #REQUIRED>
```

For each EXPRESS defined data type with a final underlying type of EXPRESS BINARY in the context schema, the markup declaration set shall contain a corresponding XML element and attribute list declaration. The name of the XML element shall be derived from the EXPRESS defined data type identifier as specified in 9.2.2. The content model of the XML element shall be #PCDATA. The ATTLIST declaration for the XML element shall contain the following XML attributes:

— a #REQUIRED attribute named `x-id` of type ID;

— a #REQUIRED attribute named `notation` of type NOTATION.

EXAMPLE 3 - The following example illustrates the mapping of EXPRESS TYPE into its corresponding XML markup.

In EXPRESS:

```
TYPE bin = BINARY;
END_TYPE;
```


As XML markup declarations:

```
<!ELEMENT Bin (#PCDATA)>
<!ATTLIST Bin
 x-id  ID  #REQUIRED
 notation NOTATION #REQUIRED>
```

For each EXPRESS defined data type with a final underlying type of an EXPRESS aggregate data type (SET, BAG, ARRAY, or LIST) in the context schema, the markup declaration set shall contain a corresponding XML element and attribute list declaration. The name of the XML element shall be derived from the EXPRESS defined data type identifier as specified in 9.2.2. The content model of the XML element shall be EMPTY. The ATTLIST declaration for the XML element shall contain the following XML attributes:

— a #REQUIRED attribute named `x-id`, of type ID;

— a #REQUIRED attribute named `val-r` of type IDREF.

EXAMPLE 4 - The following example illustrates the mapping of EXPRESS TYPE into its corresponding XML markup.

In EXPRESS:

```
TYPE point_set = SET OF point;
END_TYPE;
```

As XML markup declarations:

```
<!ELEMENT Point_set>
<!ATTLIST Point_set
  x-id  ID    #REQUIRED
  val-r IDREF #REQUIRED>
```

9.5 XML declarations for EXPRESS attributes

9.5.1 Attribute naming

The XML attribute names specified in this subclause may be modified in other subclauses of clause 9.

For each XML attribute derived from an EXPRESS attribute, the XML attribute name shall be the same as the identifier for the EXPRESS attribute, except that the first character of the XML name shall be uppercase and all other characters shall be lowercase.

Exception: If an EXPRESS complex entity data type has two distinct EXPRESS attributes declared in different EXPRESS entity data type declarations, and both EXPRESS attributes have the same EXPRESS identifier, each corresponding XML attribute name shall be the concatenation of:

— the XML name derived as specified in 9.2 from the identifier of the EXPRESS entity data type in which the attribute is declared, followed by

— the character '.' (FULL STOP or PERIOD), followed by

— the XML name derived from the EXPRESS attribute identifier as specified in 9.2.

NOTE 1 - The exception applies to both characterized and uncharacterized EXPRESS complex entity data types (see 9.4.3.2)

NOTE 2 - Repeated inheritance: If an EXPRESS entity data type inherits the same attribute from different supertypes that in turn inherit it from a common "ancestral" supertype, the two "copies" of the attribute are not "distinct EXPRESS attributes". They are both declared in the same entity declaration and an instance of the entity data type has only one such attribute. The above exception does not apply. The XML ATTLIST declaration contains only one mapping of the attribute, and the attribute name is not qualified.

EXAMPLE - The following example illustrates the mapping of attribute inheritance into its corresponding XML markup.

In EXPRESS:

```
SCHEMA sample;
ENTITY a;
attr1 : REAL;
END_ENTITY;

ENTITY b
SUBTYPE OF (a);
attr2 : REAL;
attr3 : INTEGER;
END_ENTITY;

ENTITY c
SUBTYPE OF (a);
attr2 : REAL;
attr3 : REAL;
attr5 : REAL;
END_ENTITY;

ENTITY d
SUBTYPE OF (b, c);
attr4 : INTEGER;
END_ENTITY;
END_SCHEMA;
```

As markup declarations:

```
<!ELEMENT A EMPTY>
<!ATTLIST A
x-id ID #REQUIRED
Attr1 CDATA #REQUIRED>

<!ELEMENT B EMPTY>
<!ATTLIST B
x-id ID #REQUIRED
Attr1 CDATA #REQUIRED
Attr2 CDATA #REQUIRED
```



```
Attr3 CDATA #REQUIRED>

<!ELEMENT C EMPTY>
<!ATTLIST C
x-id ID #REQUIRED
Attr1 CDATA #REQUIRED
Attr2 CDATA #REQUIRED
Attr3 CDATA #REQUIRED
Attr5 CDATA #REQUIRED>

<!ELEMENT D EMPTY>
<!ATTLIST D
x-id ID #REQUIRED
Attr1 CDATA #REQUIRED
B.Attr2 CDATA #REQUIRED
C.Attr2 CDATA #REQUIRED
B.Attr3 CDATA #REQUIRED
C.Attr3 CDATA #REQUIRED
Attr4 CDATA #REQUIRED
Attr5 CDATA #REQUIRED>
```

9.5.2 Explicit attributes

For each EXPRESS explicit attribute of an EXPRESS entity data type declaration associated with the XML element type declaration, the corresponding ATTLIST declaration shall contain an XML attribute declaration corresponding to the EXPRESS attribute.

The name of the XML attribute shall be as specified in 9.5.1. If the XML attribute is declared to be of type IDREF, the XML attribute name shall be appended with a HYPHEN '-' followed by a lower case 'r'.

NOTE - The appending of the XML attribute name with '-r' is to facilitate XSLT mappings without DTDs. The '-r' serves as an indicator that the attribute contains an IDREF.

If the EXPRESS attribute is declared to be OPTIONAL, then the XML attribute shall be declared to be #IMPLIED. Otherwise, the XML attribute shall be declared to be #REQUIRED.

The type of the XML attribute shall be declared according to the data type of the EXPRESS attribute, as specified in 9.6.

9.5.3 INVERSE attributes

For each EXPRESS INVERSE attribute of an EXPRESS entity data type declaration associated with the XML element type declaration, the corresponding ATTLIST declaration shall contain an XML attribute declaration corresponding to the EXPRESS attribute.

The name of the XML attribute shall be as specified in 9.5.1, except that the name shall be prepended with the character pair 'I-' (uppercase 'I' followed by a HYPHEN '-').

The XML attribute shall be declared to be #IMPLIED.

The type of the XML attribute shall be IDREFS.

EXAMPLE - The following is an example of how INVERSE attributes of EXPRESS instances are managed in this binding.

In EXPRESS:

```
SCHEMA sample;
ENTITY first;
   ref : second;
END_ENTITY;

ENTITY second;
INVERSE
   inv : first for ref;
END_ENTITY;
END_SCHEMA;
```

As markup declarations:

```
<!ELEMENT First EMPTY>
<!ATTLIST First
x-id  ID #REQUIRED
Ref-r IDREF #REQUIRED>

<!ELEMENT Second EMPTY>
<!ATTLIST Second
x-id   ID #REQUIRED
I-Inv-r IDREFS #IMPLIED>
```

9.5.4 DERIVEd attributes

For each DERIVEd attribute declared in an EXPRESS entity data type declaration associated with the XML element type declaration, the corresponding ATTLIST declaration shall contain an XML attribute declaration corresponding to the EXPRESS attribute.

The name of the XML attribute shall be as specified in 9.5.1, except that the name shall be prepended with the character pair 'D-' (uppercase 'D' followed by a HYPHEN '-').

The XML attribute shall be declared to be #IMPLIED.

The type of the XML attribute shall be declared according to the data type of the EXPRESS attribute, as specified in 9.6.

EXAMPLE - The following is an example of how derived attributes of EXPRESS instances are managed in this binding.

In EXPRESS:

```
SCHEMA sample;
ENTITY point;
```



```
x, y : REAL;
END_ENTITY;

ENTITY circle;
x : REAL;
y : REAL;
radius : REAL;
DERIVE
center : point := get_center(SELF);
area : REAL := 3.1416 * radius;
END_ENTITY;
END_SCHEMA;
```

As markup declarations:

```
<!ELEMENT Point EMPTY>
<!ATTLIST Point
x-id  ID #REQUIRED
XCDATA #REQUIRED
YCDATA #REQUIRED>

<!ELEMENT Circle EMPTY>
<!ATTLIST Circle
x-id  ID #REQUIRED
X  CDATA #REQUIRED
Y  CDATA #REQUIRED
Radius CDATA #REQUIRED
D-Center-r IDREF #IMPLIED
D-Area CDATA #IMPLIED>
```

9.5.5 Redeclared attributes

The redeclaration of EXPRESS attributes shall have no effect on the XML markup declaration.

9.6 XML attribute types for EXPRESS attributes

The type of the XML attributes corresponding to EXPRESS explicit and derived attributes shall be specified according to the data type of the EXPRESS attribute.

9.6.1 NUMBER, REAL, INTEGER attributes

The XML attribute corresponding to an EXPRESS attribute of type NUMBER or REAL or INTEGER shall be declared to have type CDATA in the XML markup declaration.

EXAMPLE - The following example illustrates the mapping of EXPRESS attributes of type NUMBER, REAL, and INTEGER into its corresponding XML markup.

In EXPRESS:

```
ENTITY car;
vin     : NUMBER;
```



```
mileage : INTEGER;
weight  : OPTIONAL REAL;
END_ENTITY;
```

As markup declarations:

```
<!ELEMENT Car EMPTY>
<!ATTLIST Car
x-id  ID  #REQUIRED
Vin CDATA  #REQUIRED
Mileage CDATA  #REQUIRED
Weight CDATA  #IMPLIED>
```

9.6.2 BOOLEAN attributes

The XML attribute corresponding to an EXPRESS attribute of type BOOLEAN shall be declared to have type "(true | false | 1 | 0)" in the XML markup declaration.

EXAMPLE - The following example illustrates the mapping of EXPRESS attributes of type BOOLEAN into its corresponding XML markup.

In EXPRESS:

```
ENTITY car;
airbag     : BOOLEAN;
sunroof    : OPTIONAL BOOLEAN;
END_ENTITY;
```

As markup declarations:

```
<!ELEMENT Car EMPTY>
<!ATTLIST Car
x-id ID  #REQUIRED
Airbag (true|false|1|0)#REQUIRED
Sunroof (true|false|1|0)#IMPLIED>
```

9.6.3 LOGICAL attributes

The XML attribute corresponding to an EXPRESS attribute of type LOGICAL shall be declared to have type "(true | false | unknown)" in the XML markup declaration.

EXAMPLE - The following example illustrates the mapping of EXPRESS attributes of type LOGICAL into its corresponding XML markup.

In EXPRESS:

```
ENTITY car;
front_airbag     : LOGICAL;
side_airbag      : OPTIONAL LOGICAL;
END_ENTITY;
```


As markup declarations:

```
<!ELEMENT Car EMPTY>
<!ATTLIST Car
x-id  ID #REQUIRED
Front_airbag (true|false|unknown) #REQUIRED
Side_airbag  (true|false|unknown) #IMPLIED>
```

9.6.4 STRING attributes

The XML attribute corresponding to an EXPRESS attribute of type STRING shall be declared to have type IDREF in the XML markup declaration.

EXAMPLE - The following example illustrates the mapping of EXPRESS attributes of type STRING into its corresponding XML markup.

In EXPRESS:

```
ENTITY car;
make    : STRING;
car_model   : OPTIONAL STRING;
END_ENTITY;
```

As markup declarations:

```
<!ELEMENT Car EMPTY>
<!ATTLIST Car
x-id   ID   #REQUIRED
Make-r IDREF  #REQUIRED
Car_model-r  IDREF  #IMPLIED>
```

9.6.5 BINARY attributes

The XML attribute corresponding to an EXPRESS attribute of type BINARY shall be declared to have type IDREF in the XML markup declaration.

EXAMPLE - The following example illustrates the mapping of EXPRESS attributes of type BINARY into its corresponding XML markup.

In EXPRESS:

```
ENTITY car;
mfg_code       : BINARY;
welding_code   : OPTIONAL BINARY;
END_ENTITY;
```

As markup declarations:

```
<!ELEMENT Car EMPTY>
<!ATTLIST Car
```



```
x-id   ID    #REQUIRED
Mfg_code-r   IDREF    #REQUIRED
Welding_code-r  IDREF    #IMPLIED>
```

9.6.6 ENUMERATION-typed attributes

The XML attribute corresponding to an EXPRESS attribute whose data type is an ENUMERATION data type shall be declared in the XML markup declaration to have type "(enumeration_item<1> | … | enumeration_item<n>)", where "enumeration_item<n>" is derived as specified in 9.2 from the nth identifier in the ordered list of enumeration items in the EXPRESS type declaration, and each enumeration item is represented in lowercase.

EXAMPLE - The following example illustrates the mapping of EXPRESS attributes that are ENUMERATION types into the corresponding XML markup.

In EXPRESS:

```
TYPE flower_colour = ENUMERATION OF (red, yellow, white);
END_TYPE;

ENTITY plant;
colour : flower_colour;
ext_colour : OPTIONAL flower_colour;
END_ENTITY;
```

As markup declarations:

```
<!ELEMENT Plant EMPTY>
<!ATTLIST Plant
x-id   ID    #REQUIRED
Colour (red|yellow|white) #REQUIRED
Ext_colour (red|yellow|white) #IMPLIED>
```

9.6.7 SELECT-typed attributes

The XML attribute corresponding to an EXPRESS attribute whose data type is a SELECT data type shall be declared to have type IDREF in the XML markup declaration.

EXAMPLE - The following example illustrates the mapping of EXPRESS attributes of defined types of EXPRESS SELECT into its corresponding XML markup.

In EXPRESS:

```
TYPE efficiency = SELECT (efficiency_measure, efficiency_range);
END_TYPE;

ENTITY filtration_system;
filtration_efficiency: efficiency;
pump_efficiency: OPTIONAL efficiency;
END_ENTITY;
```


As markup declarations:

```
<!ELEMENT Filtration_system EMPTY>
<!ATTLIST Filtration_system
x-id    ID  #REQUIRED
Filtration_efficiency-r  IDREF  #REQUIRED
Pump_efficiency-r IDREF  #IMPLIED>
```

9.6.8 EXPRESS entity instance-valued attributes

The XML attribute corresponding to an EXPRESS attribute whose data type is an EXPRESS entity data type shall be declared to have type IDREF in the XML markup declaration.

EXAMPLE - The following example illustrates the mapping of EXPRESS attributes of type ENTITY into its corresponding XML markup.

In EXPRESS:

```
ENTITY garden;
has_pond : fish_pond;
has_greenhouse : OPTIONAL greenhouse;
END_ENTITY;

ENTITY fish_pond;
END_ENTITY;

ENTITY greenhouse;
END_ENTITY;
```

As markup declarations:

```
<!ELEMENT Garden EMPTY>
<!ATTLIST Garden
x-id   ID #REQUIRED
Has_pond-r   IDREF #REQUIRED
Has_greenhouse-r IDREF #IMPLIED>
```

9.6.9 Attributes with non-constructed defined types

The XML attribute corresponding to an EXPRESS attribute whose data type is a non-constructed defined type shall be declared to have the XML type corresponding to the underlying type of the defined data type in the EXPRESS type declaration. The corresponding XML type shall be as specified in clause 9.6.

EXAMPLE - The following example illustrates the mapping of EXPRESS attributes of defined types of underlying primitives into its corresponding XML markup.

In EXPRESS:

```
TYPE model_name = STRING;
END_TYPE;
```



```
TYPE vin_type = NUMBER;
END_TYPE;

TYPE mileage_type = INTEGER;
END_TYPE;

TYPE weight_type = REAL;
END_TYPE;

TYPE safety_feature_present = BOOLEAN;
END_TYPE;

TYPE traction_control = LOGICAL;
END_TYPE;

ENTITY car;
car_model   : model_name;
vin     : vin_type;
mileage : mileage_type;
weight  : OPTIONAL weight_type;
has_airbags : safety_feature_present;
has_AWD : traction_control;
END_ENTITY;
```

As markup declarations:

```
<!ELEMENT Car EMPTY>
<!ATTLIST Car
x-id ID  #REQUIRED
Car_model-r    IDREF       #REQUIRED
Vin CDATA  #REQUIRED
Mileage CDATA  #REQUIRED
Weight CDATA  #IMPLIED
Has_airbags (true|false|1|0) #REQUIRED
Has_awd   (true|false|unknown) #REQUIRED>
```

9.6.10 Attributes with aggregate data types

The XML attribute corresponding to an EXPRESS attribute whose data type is an aggregate data type shall be declared to have type IDREF in the XML markup declaration.

EXAMPLE - The following example illustrates the mapping of EXPRESS attributes of type aggregate into its corresponding XML markup.

In EXPRESS:

```
TYPE flower_color = ENUMERATION OF (red, yellow, white);
END_TYPE;

TYPE vehicle = SELECT(car, plane);
END_TYPE;

ENTITY car;
```



```
make : INTEGER;
END_ENTITY;

TYPE plane = INTEGER;
END_TYPE;

ENTITY product;
END_ENTITY;

ENTITY product_definition;
attr1 : SET OF INTEGER;
attr2 : BAG [0:?] OF REAL;
attr3 : ARRAY [1:3] OF NUMBER;
attr4 : LIST [1:?] OF BOOLEAN;
attr5 : SET OF LOGICAL;
attr6 : SET OF STRING;
attr7 : LIST OF flower_color;
attr8 : SET OF vehicle;
attr9 : LIST OF product;
attr10 : ARRAY [1:2] OF ARRAY [1:2] OF REAL;
END_ENTITY;
```

As markup declarations:

```
<!ELEMENT Product_definition EMPTY>
<!ATTLIST Product_definition
x-id ID #REQUIRED
Attr1-r IDREF  #REQUIRED
Attr2-r IDREF  #REQUIRED
Attr3-r IDREF  #REQUIRED
Attr4-r IDREF  #REQUIRED
Attr5-r IDREF  #REQUIRED
Attr6-r IDREF  #REQUIRED
Attr7-r IDREF  #REQUIRED
Attr8-r IDREF  #REQUIRED
Attr9-r IDREF  #REQUIRED
Attr10-rIDREF  #REQUIRED>
```

9.7 Constraints

This subclause specifies the declarations that shall be present in the markup declaration set representing constraints specified in EXPRESS entity data types.

9.7.1 Local domain rules

For each named WHERE rule in an EXPRESS entity data type declaration associated with the XML element type declaration, the corresponding ATTLIST declaration shall contain an XML attribute declaration corresponding to the WHERE rule.

The name of the XML attribute shall be derived from the EXPRESS identifier for the WHERE rule as specified in 9.2, except that the name shall be prepended with the character pair 'W-' (an uppercase ’W’ followed by a HYPHEN '-').

If the name of the local WHERE rule in the EXPRESS entity data type conflicts with the name of an inherited WHERE rule, the corresponding XML attribute name for each of the conflicting rules shall be the concatenation of:

— The character pair 'W-', followed by

— the XML name derived as specified in 9.2 from the identifier of the EXPRESS entity data type in which the WHERE rule is declared, followed by

— the character PERIOD '.', followed by

— the XML name derived from the EXPRESS attribute identifier as specified in 9.2.

The XML attribute shall be declared to be #IMPLIED.

The type of the XML attribute shall be “(true | false | unknown)”.

EXAMPLE - The following is an example of how named WHERE rules in EXPRESS entity data types are managed in this binding.

In EXPRESS:

```
SCHEMA sample;
ENTITY action;
num : INTEGER;
WHERE
wr1 : num > 0;
END_ENTITY;
END_SCHEMA;
```

As markup declarations:

```
<!ELEMENT Action EMPTY>
<!ATTLIST  Action
    x-id   ID    #REQUIRED
    Name   CDATA  #REQUIRED
    W-Wr1    ( false | true | unknown) #IMPLIED>
```

9.7.2 Uniqueness rules

For each named UNIQUE rule in an EXPRESS entity data type declaration associated with the XML element type declaration, the corresponding ATTLIST declaration shall contain an XML attribute declaration corresponding to the UNIQUE rule.

The name of the XML attribute shall be derived from the EXPRESS identifier for the UNIQUE rule as specified in 9.2, except that the name shall be prepended with the character pair 'U-' (uppercase ’U’ followed by a HYPHEN '-').

The XML attribute shall be declared to be #IMPLIED.

The type of the XML attribute shall be “(true | false | unknown)”.

EXAMPLE - The following is an example of how UNIQUE attributes of EXPRESS instances are managed in this binding.

In EXPRESS:

```
SCHEMA sample;

   ENTITY key;
   END_ENTITY;

   ENTITY product;
   id : INTEGER;
   name : key;
   UNIQUE
   UR1 : id;
   UR2 : name;
   END_ENTITY;
   END_SCHEMA;
```

As markup declarations:

```
   <!ELEMENT Key  EMPTY>
   <!ATTLIST Key
       x-id   ID   #REQUIRED>

   <!ELEMENT Product  EMPTY)>
   <!ATTLIST Product
       x-id  ID   #REQUIRED
       Id    CDATA   #REQUIRED
       Name-r  IDREF   #REQUIRED
       U-Ur1   (false | true | unknown)  #IMPLIED
       U-Ur2   (false | true | unknown)  #IMPLIED>
```

9.8 Creation of express_data content

This subclause specifies the rules governing the creation of the content of the `express_data` element that contain data represented using the OSEB.

9.8.1 Unit of serialization

A unit of serialization is a set of XML elements that:

— represent a graph of EXPRESS data instances that are described by a single EXPRESS schema, designated the "context schema" for the unit of serialization, and

— have been constructed from the context schema according to the specifications in this clause, and

— have no local XML references (IDREFs) to elements that are not in the set.

For each unit of serialization:

— One `uos` element shall be constructed. The content of the `uos` element shall consist of a collection of XML elements representing the set of EXPRESS data instances according to the specifications in this clause.

— The `schema` XML attribute of the `uos` element shall identify the 'context schema' to which the contained data corresponds. The EXPRESS schema name shall be represented in the value of the `schema` attribute as in EXPRESS, except that the first character shall be in uppercase and all other characters shall be in lower case.

— Representing EXPRESS entity instances as OSEB elements may result in an unconnected graph of elements that can be considered as a set of directed graphs. The graph is defined as a set of nodes that are elements with ID XML attributes and a set of directed edges that are IDREF links. These elements are the content of the `uos` element. Each directed graph in the set has zero or more “root nodes”, i.e. elements that cannot be reached by traversal of the directed edges within the graph. The `x-id` value of each of these root nodes shall appear in the `c` XML attribute of the `uos` element. For any directed graph that does not have a root node, that graph has a cycle and the `x-id` value of one element from the cycle shall appear in the `c` XML attribute of the `uos` element.

— Every EXPRESS entity instance in the unit of serialization shall be represented by one or more XML elements in the content of the `uos` element, as specified in 9.8.3.

— No more than one `unset` element shall be contained in the `uos` element, as specified in 9.8.4. If the `unset` element is contained within the `uos` element, the `unset` attribute of the `uos` element shall be present and shall match the `x-id` value of the `unset` element. If the `unset` element is not contained within the `uos` element, the `unset` attribute of the `uos` element shall not be present.

EXAMPLE - This example illustrates the unit of serialization.

In EXPRESS:

```
SCHEMA car_ownership;
ENTITY person;
  name : STRING;
  owns : car;
END_ENTITY;
ENTITY car;
  make : STRING;
  car_model : STRING;
END_ENTITY;
```



```
END_SCHEMA;
```

In XML instance data:

```
<?xml version="1.0"?>
<!DOCTYPE iso_10303_28 SYSTEM "car_ownership.dtd">
<iso_10303_28 representation_category=”OSEB”>
  <express_data id=”expdata01”>
      <osb:uos xmlns= “urn:iso10303-28:oseb/Car_ownership”
            xmlns:osb = "urn:iso10303-28:oseb"
             schema="Car_ownership"
             c="id10">
     <Person  x-id=”id10” Name-r=”s3” Owns-r=”id20”/>
     <Car  x-id=”id20” Make-r=”s1” Car_model-r=”s2”/>
     <osb:string x-id=”s1”>BMW</osb:string>
     <osb:string x-id=”s2”>Z3</osb:string>
     <osb:string x-id=”s3”>John Q. Public</osb:string>
   </osb:uos>
</express_data>
</iso_10303_28>
```

No local XML identifiers for any elements within a `uos` element shall be referenced from outside that `uos` element. Every reference from an element in one `uos` to an element in another `uos` shall use an external reference element (see 9.3.5), even when both `uos` elements are in the same XML document.

The `uos` element (i.e. the contained data set) may contain a fixed "external" identifier in some larger name space, which permits it to be referenced by other data sets, as an "external reference". This "external" identifier is represented by the `edokey` element and is specified in 9.3.5.

9.8.1.1 The edokey element, enterprise data objects and external identifiers

When an XML data instance within a unit of serialization is associated with a persistent external identifier, the XML data instance is said to represent an ‘enterprise data object’ or ‘edo’ and the persistent identifier is said to be its ‘edo key’. The `edokey` element specified in 9.3.5 shall represent the persistent identifier for an XML element within a unit of serialization. For each `edokey` element contained in the unit of serialization the following shall apply.

— The `edokey` elements shall be contained in the `uos` element representing the unit of serialization.

— The `ob` attribute of the `edokey` element, if present, shall reference the `x-id` of the XML element for which it is the persistent identifier.

— The `xlink:type` attribute of the `edokey` element, if present, shall have the value “extended”.

— External identification of a data set within the `uos` element shall be provided by the `edokey`, `lockey`, `this` and `go` elements.

— The `xlink:href`, `xlink:title`, `xlink:show`, and `xlink:actuate` attributes for the `edokey` element and any of its children, if present, shall contain values as specified in XML Linking Language.

NOTE - An edokey (enterprise data object key) is the persistent identifier for an enterprise data object (edo). It may be represented as a string and shared and persisted across middleware technologies. It is represented as an XML information item.

EXAMPLE - This example illustrates a unit of serialization that contains an edo and an edo key.

In EXPRESS:

```
SCHEMA edo_schema;
ENTITY edo;
  rels : edo_ext;
END_ENTITY;
ENTITY edo_ext;
  role : STRING;
  val  : edokey;
END_ENTITY;
ENTITY edokey;
END_ENTITY;
END_SCHEMA;

SCHEMA car_ownership;
USE FROM edo_schema;
ENTITY person
SUBTYPE OF (edo);
  name : STRING;
  owns : car;
END_ENTITY;
ENTITY car;
  make : STRING;
  car_model : STRING;
END_ENTITY;
END_SCHEMA;
```

And associated instance data in a file named “www.acme.com/cardata.xml”:

```
#10 = Person(‘John Q. Public’, #20);
#20 = Car(‘BMW’,’Z3’);
```

In XML instance data:

```
<?XML version=”1.0”?>
<!DOCTYPE iso_10303_28 SYSTEM "car_ownership.dtd">
<iso_10303_28 representation_category=”OSEB”>
  <express_data id=”expdata02”>
    <osb:uos
      xmlns= “urn:iso10303-28:oseb/car_ownership”
      xmlns:osb = "urn:iso10303-28:oseb"
      xmlns:xlink=”http://www.w3.org/1999/xlink”
      schema="Car_ownership" c=”id0”>
```



```
        <osb:edokey x-id="id0" ob="id10"
           xlink:type=”extended”
           xlink:role=”osb:edokey”
           xml:base = "http://www.acme.com/1999/edos/person.home">

            <osb:this xlink:type = “resource”
              xlink:role = “osb:this”
             xlink:title=”Optional user help”>John Q. Public</osb:this>

        <osb:lockey
             xlink:type= “locator”
             xlink:href = “#xpointer(Person[@Name = ‘John Q. Public’
             and namespace() = ‘urn:iso10303-23:oseb/Car_ownership’])”
             xlink:role= “osb:edo”
             xlink:title=“The data object John Q. Public” />

         <osb:go
             xlink:type = “arc”
             xlink:from = “osb:this”
             xlink:to = “osb:edo”
             xlink:actuate = “onRequest”
             xlink:role = “osb:this” />

       </osb:edokey>

       <Person  x-id=”id10” Name-r=”s3” Owns-r=”id20”/>
       <Car  x-id=”id20” Make-r=”s1” Car_model-r=”s2”/>
       <osb:string x-id=”s1”>BMW</osb:string>
       <osb:string x-id=”s2”>Z3</osb:string>
       <osb:string x-id=”s3”>John Q. Public</osb:string>
 </osb:uos>
 </express_data>
</iso_10303_28>
```

9.8.2 Representation of EXPRESS entity instances

For each EXPRESS entity instance in the unit of serialization, the `uos` element shall contain an XML element of the type corresponding to the EXPRESS entity data type as specified in 9.4.3. The XML element shall have a local XML identifier, and the value of the `x-id` attribute shall be that identifier.

For each EXPRESS mandatory explicit attribute of the entity data type, the XML element shall have a corresponding XML attribute, as specified in 9.4.3. The value of the XML attribute shall represent the value of that EXPRESS attribute in the instance according to the data type of that attribute, as specified in 9.9.

For each EXPRESS OPTIONAL explicit attribute of the entity data type, the XML element shall have a corresponding XML attribute, as specified in 9.4.3, if the entity instance has a value for that attribute. The value of the XML attribute shall represent the value of the EXPRESS attribute in the instance according to the data type of that attribute as specified in 9.9. If the instance has no value for that attribute, there shall be no corresponding XML attribute in the XML element.

For each DERIVEd attribute of the entity data type, the XML element may have a corresponding XML attribute, as specified in 9.5.4. If provided, the value of the XML attribute shall represent the value of the EXPRESS attribute in the instance according to the data type of that attribute as specified in 9.9.

For each INVERSE attribute of the entity data type, the XML element may have a corresponding XML attribute, as specified in 9.5.3. If provided, the value of the XML attribute shall represent the value of that EXPRESS attribute in the instance according to the data type of that attribute, as specified in 9.9.

For each named WHERE rule and each named UNIQUE rule of the EXPRESS entity data type, the XML element may have a corresponding XML attribute, as specified in 9.7. If provided, the value of the XML attribute shall be "true" if the expression in the rule evaluates to EXPRESS TRUE and "false" if the constraint is not met. In those cases in which the producer of the XML document does not evaluate a constraint, the corresponding XML attribute may be provided with a value of "unknown", or not provided.

This part of ISO 10303 does not specify the circumstances under which values for XML attributes corresponding to DERIVEd and INVERSE attributes, or to WHERE and UNIQUE rules, are to be provided.

9.8.3 Representation of the unset element

The `unset` element represents a non-existent value. The `uos` element shall contain no more than one `unset` element. The `unset` element shall have a local XML identifier and that identifier shall be the value of the `x-id` attribute of the element.

9.9 Representation of EXPRESS attribute values

This subclause describes the representations of EXPRESS attribute values.

9.9.1 Representation of INTEGER values

A value of EXPRESS data type INTEGER shall be represented in the form specified as Numeric Representation 1 (NR1) in ISO 6093, i.e. an optional HYPHEN mark followed by a sequence of one or more digits, where the presence of the HYPHEN mark (minus sign) signifies that the value is negative.

The minimum allowable value shall be -9223372036854775808, and the maximum allowable value shall be 9223372036854775808.

EXAMPLE - The following are valid literals that may be assigned to an XML attribute that represents an EXPRESS attribute of data type INTEGER:

```
-1
0
12678967543233
```


9.9.2 Representation of REAL and NUMBER values

A value of EXPRESS data type REAL or NUMBER shall be represented in one of the Numeric Representation forms (NR1, NR2, NR3) specified in ISO 6093.

NOTE 1 - The general form of content data conforming to NR1, NR2 or NR3 is an optional minus-sign (HYPHEN mark), followed by a sequence of one or more digits designating the integral part, optionally followed by a fractional part consisting of a decimal-point (either PERIOD or COMMA) and a sequence of one or more digits, optionally followed by an exponent part consisting of the letter “e” or “E”, a sign (PLUS or HYPHEN) and a sequence of one or more digits.

The instance data value of an XML attribute representing an EXPRESS REAL or EXPRESS NUMBER shall correspond to the IEEE 754 double-precision 64-bit floating point type [IEC 559:1987 = IEEE 754-1985]. The basic value space of such a data value consists of the values *m x 2^e*, where *m* is an integer whose absolute value is less than 2^53, and *e* is an integer between –1075 and 970, inclusive.

In addition, where data type REAL is considered to be extended by infinite values and exceptional values, the extended values shall be represented as follows:

— the character sequence "INF" shall designate a positive infinite value;

— the character sequence "-INF" shall designate a negative infinite value;

— the character sequence "NaN" shall designate any exceptional value.

NOTE 2 - Infinite and exceptional values are specified for computational arithmetic conforming to IEC 559:1994 "Floating-point arithmetic for microprocessors" and other standards.

EXAMPLE - The following are valid literals that may be assigned to an XML attribute that represents an EXPRESS REAL or EXPRESS NUMBER:

```
-1E+4
1267.43233E+12
12.78e-2
12
INF
```

9.9.3 Representation of BOOLEAN values

A value of EXPRESS BOOLEAN data type shall be represented by one of the following character sequences:

— "0" or "false" shall designate the value FALSE;

— "1" or "true" shall designate the value TRUE.

9.9.4 Representation of LOGICAL values

A value of EXPRESS LOGICAL data type shall be represented by one of the following character sequences:

— "false" shall designate the value FALSE;

— "true" shall designate the value TRUE;

— "unknown" shall designate the value UNKNOWN.

9.9.5 Representation of STRING values

For a value of EXPRESS STRING data type, the `uos` element shall contain an XML `string` element (see 9.3.1) and the content of that element shall be the encoding of the EXPRESS STRING using the same encoding as specified in 7.4.1.4. The `string` element shall have an XML identifier and that identifier shall be the value of the `x-id` attribute of the `string` element.

The local XML identifier for the `string` element shall represent the STRING value as the value of an EXPRESS attribute, or as a member of an aggregate value or as an instance of a select type.

9.9.6 Representation of BINARY values

For a value of EXPRESS BINARY data type, the `uos` element shall contain an XML `hex_binary` element or `base64_binary` element (see 9.3.1), and the content of that element shall be the encoding of the EXPRESS BINARY value using the corresponding encoding as specified in 7.4.1.5. The XML element shall have an XML identifier and that identifier shall be the value of the `x-id` attribute of the XML element.

The local XML identifier shall represent the BINARY value as the value of an EXPRESS attribute, or as a member of an aggregate value or as an instance of a select type.

9.9.7 Representation of ENUMERATION items

A value of an EXPRESS ENUMERATION type (an enumeration item) shall be represented by the XML name of that value (enumeration item), derived as specified in 9.2.1.

9.9.8 Representation of EXPRESS entity instances as values of attributes

For any EXPRESS entity instance used as the value of an attribute, the `uos` element shall contain an XML element corresponding to the EXPRESS entity instance as specified in 9.8.2. The local XML identifier for the EXPRESS entity instance element shall represent the reference to the EXPRESS entity instance as the value of an EXPRESS attribute, or as a member of an aggregate value, or as an instance of a select type.

9.9.9 Representation of values of SELECT types

For any value used as a value of an EXPRESS SELECT data type, the `uos` element shall contain an XML element of the type corresponding to the SELECT data type as specified in 9.6.7, hereafter called the *select element*. The select element shall have a local XML identifier and that identifier shall be the value of the `x-id` attribute of the element.

The value of the SELECT data type shall instantiate one of the EXPRESS data types in the select list of the EXPRESS declaration of the SELECT data type. The value of the `utype` attribute of the select element shall indicate which EXPRESS data type in the select list is being instantiated. The value of the `utype` attribute shall be the XML name token of the element corresponding to that EXPRESS data type, as specified in 9.2.

In addition, the `uos` element shall contain an XML element of the type corresponding to the structure of the value itself, herein called the *value element*, as follows:

— if the value is a NUMBER value, the value element shall be a `number` element (see 9.2.1), and the `val` attribute of that element shall be the NUMBER value represented as specified in 9.9.2;

— if the value is a REAL value, the value element shall be a `double` element (see 9.2.1), and the `val` attribute of that element shall be the REAL value represented as specified in 9.9.2;

— if the value is an INTEGER value, the value element shall be a `long` element (see 9.2.1), and the `val` attribute of that element shall be the INTEGER value represented as specified in 9.9.1;

— if the value is a BOOLEAN value, the value element shall be a `boolean` element (see 9.2.1), and the `val` attribute of that element shall be the BOOLEAN value represented as specified in 9.9.3;

— if the value is a LOGICAL value, the value element shall be a `logical` element (see 9.2.1), and the `val` attribute of that element shall be the LOGICAL value represented as specified in 9.9.4;

— if the value is a STRING value, the value element shall be a `string` element as specified in 9.9.5;

— if the value is a BINARY value, the value element shall be a `hex_binary` element or a `base64_binary` element as specified in 9.9.6;

— if the value is an EXPRESS entity instance identifier value, the value element shall be an element representing the entity instance, as specified in 9.9.8;

— if the value is a value of an EXPRESS non-constructed defined data type, the value element shall be an XML element of the type corresponding to the EXPRESS data type, as specified in 9.9.11.

— if the value is an enumeration item, the value element shall be an XML element of the type corresponding to the enumeration data type, as specified in 9.3.1, and the `val` attribute of that element shall be the enumeration item value represented as specified in 9.9.7;

— if the value is to be interpreted as a value of another SELECT data type, the value shall be represented as specified in this subclause, and the select element of that representation shall be the value element of this representation;

— if the value is a value of an aggregate data type, the value shall be represented as specified in 9.9.10 and the collection element shall be the value element.

The value element shall have a local XML identifier, and that identifier shall be the value of the `x-id` attribute of the value element.

The value of the `val` attribute of the select element shall be the local XML identifier for the value element.

NOTE - The value element depends on the ultimate underlying type of the EXPRESS value, that is, the EXPRESS data type that defines the structure of the representation. The `utype` attribute of the select element identifies the data type that the EXPRESS data value instantiates.

The local XML identifier for the select element shall represent the SELECT value as the value of an attribute or as a member of an aggregate value.

9.9.10 Representation of aggregate values

The representation of EXPRESS aggregate values, excluding those that are members of a sparse array (ARRAY OF OPTIONAL), is specified for the various base types in 9.9.10.1 – 9.9.10.6.

The representation of EXPRESS aggregate values that are members of a sparse array (ARRAY OF OPTIONAL) is specified in 9.9.10.7.

9.9.10.1 Aggregates of simple values

For a value of any EXPRESS aggregate data type (SET, LIST, BAG, ARRAY) whose base type is NUMBER, REAL, INTEGER, BOOLEAN, LOGICAL, any ENUMERATION data type, or a defined data type based on any of these, the `uos` element shall contain a corresponding "collection" element as defined in 9.3.2. Specifically:

— for an aggregate of type NUMBER, the collection element shall be an `nctn` element;

— for an aggregate of type REAL, the collection element shall be a `dctn` element;

— for an aggregate of type INTEGER, the collection element shall be an `lctn` element;

— for an aggregate of type BOOLEAN, the collection element shall be a `bctn` element;

— for an aggregate of type LOGICAL, the collection element shall be an `lbctn` element;

— for an aggregate of any ENUMERATION data type, the collection element shall be an `ectn` element.

The collection element shall have a local XML identifier and that identifier shall be the value of the `x-id` attribute of the collection element. The value of the contents attribute, `c` , of the collection element shall be a list of the representations of the component values of the aggregate value. Each of the component values shall be represented according to its data type, as specified in 9.9.1-9.9.4 and 9.9.7. The list members shall be separated by whitespace as specified in the XML 1.0 Recommendation.

The local XML identifier for the collection element shall represent the aggregate value as the value of an attribute or as a member of another aggregate value.

EXAMPLE - The following example illustrates the representation of aggregates of simple values as XML instance data.

In EXPRESS:

```
TYPE flower_color = ENUMERATION OF (red, yellow, white);
END_TYPE;

ENTITY product_definition;
attr1 : SET OF INTEGER;
attr2 : BAG [0:?] OF REAL;
attr3 : ARRAY [1:3] OF NUMBER;
attr4 : LIST [1:?] OF BOOLEAN;
attr5 : SET OF LOGICAL;
attr6 : SET OF STRING;
attr7 : LIST OF flower_color;
END_ENTITY;
```

As markup declarations:

```
<!ELEMENT Product_definition EMPTY>
<!ATTLIST Product_definition
x-id ID #REQUIRED
Attr1-r IDREF  #REQUIRED
Attr2-r IDREF  #REQUIRED
Attr3-r IDREF  #REQUIRED
Attr4-r IDREF  #REQUIRED
Attr5-r IDREF  #REQUIRED
Attr6-r IDREF  #REQUIRED
Attr7-r IDREF  #REQUIRED>
```

In XML instance data:

```
<Product_definition x-id = “id10” Attr1-r = “id1” Attr2-r = “id2”
Attr3-r = “id3” Attr4-r = “id4” Attr5-r = “id5”
Attr6-r = “id6” Attr7-r = “id7”/>
<ctn x-id = “id6” ctype="string" c = “id60 id70” />
<lctn x-id = “id1” c = “0 1 2” />
<dctn x-id = “id2” c = “2.3 4.5” />
<nctn x-id = “id3” c = “1.2 2 3.4” />
```



```
<bctn x-id = “id4” c = “true true false” />
<lbctn x-id = “id5” c = “true unknown” />
<ectn x-id = “id7” c = “red red white” />
<string x-id = “id60”>hi</string>
<string x-id = “id70”>there you</string>
```

9.9.10.2 Aggregates of STRING or BINARY values

A value of any EXPRESS aggregate data type (SET, LIST, BAG, ARRAY) whose base type is STRING or BINARY or a defined data type based on STRING or BINARY shall be represented as follows:

— each component of the aggregate value shall be represented according to its (underlying) data type, as specified in 9.9;

— the `uos` element shall contain an XML `ctn` element. The element shall have a local XML identifier and that identifier shall be the value of the `x-id` attribute of the `ctn` element. The value of the `ctype` attribute shall be the XML element type name of the string or binary representation element that makes up the collection appended with the character pair ‘[]’. The value of the contents attribute, `c`, of the `ctn` element shall be a list of the local XML identifiers for the elements containing the component values. The list shall have the XML IDREFS form.

NOTE - The appending of the ctype attribute value with the character pair '[ ]' is to facilitate XSLT mappings without DTDs. The '[ ]' serves as an indicator as to the dimension of the collection represented. If it appears once, the representation is of a one-dimensional array or collection. If it appears twice, the representation is that of a two-dimensional array or collection and so forth.

The local XML identifier for the collection (`ctn`) element shall represent the aggregate value as the value of an attribute or as a member of another aggregate value.

EXAMPLE - The following example illustrates the representation of aggregates of string values as XML instance data.

In EXPRESS:

```
ENTITY product_definition;
attr1 : SET OF STRING;
END_ENTITY;
```

As markup declarations:

```
<!ELEMENT Product_definition EMPTY>
<!ATTLIST Product_definition
x-id ID #REQUIRED
Attr1-r IDREF  #REQUIRED>
```

In XML instance data:

```
<Product_definition x-id = “id10” Attr1-r = “id1”/>
<ctn x-id = “id1” ctype="string[]" c = “id60 id70” />
```



```
<string x-id = “id60”>hi</string>
<string x-id = “id70”>there you</string>
```

9.9.10.3 Aggregates of aggregate values

A value of any EXPRESS aggregate data type (SET, LIST, BAG, ARRAY) whose base type is also an aggregate data type shall be represented as follows:

— each component of the aggregate value shall be represented according to its (underlying) data type, as specified in 9.9.10.

NOTE - Every component of the aggregate value is itself an aggregate value.

— The `uos` element shall contain a `ctn` element. The element shall have a local XML identifier and that identifier shall be the value of the `x-id` attribute of the `ctn` element. The value of the `ctype` attribute shall be the XML name of the base type collection element that is being aggregated appended with the character pair ‘[]’ for each level of nesting. The value of the contents attribute, `c`, of the `ctn` element shall be a list of the local XML identifiers for the `ctn` elements representing the component values. The list shall have the XML IDREFS form.

EXAMPLE - For the EXPRESS attributes of aggregates of aggregates:

In EXPRESS:

```
ENTITY spread_sheet;
attr1 : SET OF SET OF NUMBER;
attr2 : BAG [0:?] OF BAG [0:?] OF INTEGER;
attr3 : ARRAY [1:2] OF ARRAY [1:2] OF REAL;
attr4 : LIST [1:?] OF LIST [1:?] OF BOOLEAN;
END_ENTITY;
```

As markup declarations:

```
<!ELEMENT Spread_sheet EMPTY>
<!ATTLIST Spread_sheet
Attr1-r IDREF  #REQUIRED
Attr2-r IDREF  #REQUIRED
Attr3-r IDREF  #REQUIRED
Attr4-r IDREF  #REQUIRED>
```

In XML instance data:

```
<Spread_sheet x-id = “id10” Attr1-r = “id20” Attr2-r = “id30” Attr3-r
= “id40” Attr4-r = “id50”/>
<ctn x-id = “id20” ctype="nctn[][]" c = “id60  id70”/>
<nctn x-id = “id60” c = “1.0  2.0”/>
<nctn x-id = “id70” c = “3.4  4.4”/>
<ctn x-id = “id30” ctype="lctn[][]" c = “id80  id90” />
<lctn x-id = “id80” c = “1.1  2.1”/>
<lctn x-id = “id90” c = “3.3  2.1”/>
<ctn x-id = “id40” ctype="dctn[][]" c = “id100  id110”/>
```



```
<dctn x-id = “id100” c = “1.1  2.1”/>
<dctn x-id = “id110” c = “3.3  2.1”/>
<ctn x-id = “id50” ctype="bctn[][]" c = “id120 id130”/>
<bctn x-id = “id120” c = “true”/>
<bctn x-id = “id130” c = “false false”/>
```

9.9.10.4 Aggregates of values of SELECT types

A value of any EXPRESS aggregate data type (SET, LIST, BAG, ARRAY) whose base type is a SELECT data type shall be represented as follows:

— each component of the aggregate value shall be represented as a value of the SELECT data type, as specified in 9.9.9;

— the `uos` element shall contain an XML `ctn` element. The element shall have a local XML identifier and that identifier shall be the value of the `x-id` attribute of the `ctn` element. The value of the `ctype` attribute shall be the XML name of the EXPRESS SELECT element that makes up the collection appended with the character pair ‘[]’. The value of the contents attribute, `c`, of the `ctn` element shall be a list of the local XML identifiers for the select elements for the component values. The list shall have the XML IDREFS form.

The local XML identifier for the collection (`ctn`) element shall represent the aggregate value as the value of an attribute or as a member of another aggregate value.

EXAMPLE - The following examples illustrate the representation of aggregates of EXPRESS SELECT types in the XML markup.

In EXPRESS:

```
TYPE vehicle = SELECT(car, plane);
END_TYPE;

ENTITY car;
make : INTEGER;
END_ENTITY;

TYPE plane = INTEGER;
END_TYPE;

ENTITY person;
attr1 : SET OF vehicle;
attr2 : BAG [0:?] OF vehicle;
attr3 : ARRAY [1:2] OF vehicle;
attr4 : LIST [1:?] OF vehicle;
END_ENTITY;
```

As markup declarations:

```
<!ELEMENT Vehicle EMPTY>
<!ATTLIST Vehicle
x-id ID  #REQUIRED
```



```
utype NMTOKEN  #REQUIRED
val IDREF  #REQUIRED>

<!ELEMENT Person EMPTY>
<!ATTLIST Person
x-id ID  #REQUIRED
Attr1-r IDREF  #REQUIRED
Attr2-r IDREF  #REQUIRED
Attr3-r IDREF  #REQUIRED
Attr4-r IDREF  #REQUIRED>
```

In XML instance data:

```
   <osb:uos xlmns:ex = “urn:iso10303-28:oseb/Sample”
 xmlns:osb = “urn:iso10303-28:oseb”
 c="id10">
<ex:Vehicle x-id = “id100” utype =”ex:Car”
val = “id110”/>
<ex:Car x-id = “id110”  Make = “14”/>
<ex:Vehicle x-id = “id120” utype = “ex:Car”
val = “id130”/>
<ex:Car x-id = “id130”  Make = “15”/>
<ex:Vehicle x-id = “id140” utype = “ex:Plane”
val = “id150”/>
<osb:long x-id = “id150” val = “152”/>
<ex:Vehicle x-id = “id160” utype = “ex:Car”
val = “id130”/>
<ex:Vehicle x-id = “id180” utype = “ex:Plane”
val = “id190”/>
<osb:long x-id = “id190” val = “80”/>
<ex:Vehicle x-id = “id200” utype = “ex:Plane”
val = “id210”/>
<osb:long x-id = “id210” val = “727”/>
<ex:Person x-id = “id10” Attr1-r = “id20” Attr2-r = “id30” Attr3-r =
“id40”
Attr4-r = “id50”/>
<osb:ctn x-id = “id20” ctype="Vehicle[]" c = “id100 id120”/>
<osb:ctn x-id = “id30” ctype="Vehicle[]" c = “id100”/>
<osb:ctn x-id = “id40” ctype="Vehicle[]" c = “id140  id160”/>
<osb:ctn x-id = “id50” ctype="Vehicle[]" c = “id180  id200”/>
</osb:uos>
```

9.9.10.5 Aggregates of EXPRESS entity instances

A value of any EXPRESS aggregate data type (SET, LIST, BAG, ARRAY) whose base type is an EXPRESS entity data type shall be represented as follows:

— the `uos` element shall contain an element representing each EXPRESS entity instance in the aggregate value as specified in 9.8.2;

— the `uos` element shall contain an XML `ctn` element. The element shall have a local XML identifier and that identifier shall be the value of the `x-id` attribute of the `ctn` element. The value

of the `ctype` attribute shall be the XML name of the element representing the EXPRESS entity data type that makes up the collection appended with the character pair '[]'. The value of the contents attribute, `c`, of the `ctn` element shall be a list of the local XML identifiers for the elements corresponding to the component entity instances. The list shall have the XML IDREFS form.

The local XML identifier for the collection (`ctn`) element shall represent the aggregate value as the value of an EXPRESS attribute or as a member of another aggregate value.

EXAMPLE - The following example illustrates the representation of an aggregate of an EXPRESS entity data type in the XML markup.

In EXPRESS:

```
ENTITY bed;
 holds_plants : SET [1 : ?] OF outdoors_plant;
END_ENTITY;

ENTITY outdoors_plant;
   colour                    : flower_colour;
END_ENTITY;
```

In XML instance data:

```
<uos xlmns:mjg = "urn:iso10303-28:oseb/Mr_jones_garden"
      xmlns:osb = "urn:iso10303-28:oseb"
      c="bed1">

  <mjg:Bed x-id="bed1" I-Holds_plants="plant_set3" />
  <osb:ctn x-id="plant_set3" ctype="mjg:Outdoor_plant[]" c="op1
op2"/>
  <mjg:Outdoors_plant x-id="op1" Colour="red" />
  <mjg:Outdoors_plant x-id="op2" Colour="yellow" />
</uos>
```

9.9.10.6 Aggregates of values of defined data types

A value of any EXPRESS aggregate data type (SET, LIST, BAG, ARRAY) whose base-type is a non-constructed defined data type shall be represented as if it were an aggregate of the underlying type of that defined data type. This rule shall be applied recursively until the underlying type is not a non-constructed defined data type.

9.9.10.7 ARRAY OF OPTIONAL

A value of an EXPRESS array data type with the OPTIONAL keyword may have array members with no assigned value. For each such EXPRESS array data type:

— the `ctn` element shall be used to represent the EXPRESS array data type;

— each array member with no assigned value shall be represented by the local XML identifier of the `unset` element, as specified in 9.8.3.

EXAMPLE - The following example illustrates the mapping of EXPRESS attributes which are sparse ARRAYs into its corresponding XML markup.

In EXPRESS:

```
ENTITY product_definition;
attr1 : OPTIONAL SET OF INTEGER;
attr2 : ARRAY [0:?] OF OPTIONAL REAL;
END_ENTITY;
```

As markup declarations:

```
<!ELEMENT Product_definition EMPTY>
<!ATTLIST Product_definition
x-id ID #REQUIRED
Attr1-r IDREF  #IMPLIED
Attr2-r IDREF  #REQUIRED>
```

In XML instance data:

```
<osb:uos xmlns = "urn:iso10303-28:oseb/Sample"
xmlns:osb = “urn:iso10303-28:oseb”
c=”id10”>
 <Product_definition x-id = "id10" Attr2-r = "id20"/>
 <osb:ctn x-id = "id20" ctype="osb:double[]" c = "id30  idnull
idnull  id40  id50"/>
    <osb:double x-id = "id30" val = "2.3"/>
    <osb:double x-id = "id40" val = "7.6"/>
    <osb:double x-id = "id50" val = "9.6"/>
        <osb:unset  x-id="idnull"/>
</osb:uos>
```

9.9.11 Representation of values of non-constructed defined data types

In the instance data, the value of each XML attribute that references an EXPRESS defined data type with a final underlying type of NUMBER, BOOLEAN, LOGICAL, INTEGER or REAL shall be represented as a value of the final underlying type. In the instance data, the value of each XML attribute that references element representing an EXPRESS defined data type shall be the value of the `x-id` attribute of the element representing that EXPRESS defined data type as defined in 9.4.4.

An EXPRESS defined data type is defined in an EXPRESS type declaration in terms of an underlying type. The underlying type of an EXPRESS defined data type may itself be an EXPRESS defined data type, which is defined by an EXPRESS type declaration to be based on a “deeper” underlying type, and so on. But ultimately there must be a “final underlying type” that is not itself an EXPRESS defined data type. An EXPRESS defined data type is said to be “based on” its “final underlying type”.

The representation of a value of a non-constructed defined data type depends on how the value is used and on the “final underlying data type”.

9.9.11.1 Representation as a value of an attribute

When a value of a non-constructed defined data type appears as the value of an EXPRESS attribute, it is represented in the value of the corresponding XML attribute as if it were a value of its final underlying type, according to the appropriate subclause of 9.9.

9.9.11.2 Representation as a member of an aggregate value

When a value of a non-constructed defined data type appears as the member of an aggregate value, except when it is a member of an ARRAY OF OPTIONAL (see 9.9.11.3), it shall be represented as if it were a member of an aggregate of the underlying type of that defined data type, according to the appropriate subclause of 9.9.10.

9.9.11.3 Representation as a member of a select type or an array of OPTIONAL

When a value of a non-constructed defined data type appears as a member of an array of optional values, or as a value of a SELECT type, the `uos` shall contain an XML element of the type corresponding to the non-constructed defined data type as defined in 9.4.4.

The XML element shall have a local XML identifier and that identifier shall be the value of the `x-id` attribute of the XML element. The local XML identifier shall represent the value of the non-constructed defined data type in the representation of the value of the aggregate or select type.

— if the final underlying type is NUMBER or REAL, the `val` attribute of the XML element shall be the value of the non-constructed defined data type represented as specified in 9.9.2;

— if the final underlying type is INTEGER, the `val` attribute of the XML element shall be the value of the non-constructed defined data type represented as specified in 9.9.1;

— if the final underlying type is BOOLEAN, the `val` attribute of the XML element shall be the value of the non-constructed defined data type represented as specified in 9.9.3;

— if the final underlying type is LOGICAL, the `val` attribute of the XML element shall be the value of the non-constructed defined data type represented as specified in 9.9.4;

— if the final underlying type is STRING, the content of the XML element shall be the encoding of the value of the non-constructed defined data type using the same encoding as specified in 7.4.1.4;

— if the final underlying type is BINARY, the content of the XML element shall be the encoding of the value of the non-constructed defined data type using the “hex” or “base64” encoding as specified in IETF RFC 2045, and the `notation` attribute shall specify which of these is used.

— if the final underlying type is an ENUMERATION type, the `val` attribute of the XML element shall be the value of the non-constructed defined data type represented as specified in 9.9.7;

— if the final underlying type is an aggregate type, the value of the non-constructed defined data type shall be represented as specified in 9.9.10, and the value of the `val-r` attribute of the XML element shall match the `x-id` value of the collection element.

— if the final underlying type is a select type, the value of the non-constructed defined data type shall be represented as specified in 9.9.9, and the value of the `val-r` attribute of the XML element shall match the `x-id` value of the select element.

10 XML document creation

An XML document based on this part of ISO 10303 may contain the representation of one or more EXPRESS schemas, the representation of one or more sets of data based on EXPRESS schemas or a combination of the representation of EXPRESS schemas and data. Any combination of the possible markups specified in this part of ISO 10303 for the representation of data may also appear in the same XML document.

The XML Namespace Recommendation shall be used to resolve any XML name clashes in an XML document resulting from the inclusion of data or EXPRESS schemas represented using any markup declaration set specified in, resulting from or architecturally compatible with this part of ISO 10303. If the document is to contain identically named elements from different EXPRESS schemas or from different bindings, then XML namespace prefixes shall be applied to the element information items. If necessary, the appropriate namespace prefix shall be applied to the corresponding ELEMENT and ATTLIST in the markup declarations.

10.1 General XML document structure using binding representation

An XML document based on this part of ISO 10303 shall begin with an XML declaration where the `version` is “1.0”. Additional XML attributes may be required in the remainder of this clause.

An XML document based on this part of ISO 10303 shall include the `iso_10303_28` element. In the case that the XML document contains document header information, that information shall be represented using the `iso_10303_28_header` element.

EXAMPLE - An XML document containing only a schema could begin as follows.

```
<?XML version=”1.0” encoding=”utf-8”?>
<iso_10303_28 representation_category=”SCHEMA_DECL” version=”PDTS”>
```


10.2 Representation of EXPRESS schemas

Any XML document that includes the representation of one or more EXPRESS schemas as specified in this part of ISO 10303 shall include an `express_schema` element for each EXPRESS schema and the necessary elements for the representation of, or external reference to, that EXPRESS schema.

The content of each `express_schema` element shall represent the EXPRESS schema using one of the following:

— the `schema_decl` element and the markup specified in Annex C;

— the `schema_text` element as specified in 6.3.2; or

— reference to an EXPRESS schema that is outside the XML document, using the `external_refid` element. Any external reference to a EXPRESS schema that is not represented using the `schema_decl` or `schema_text` elements is outside the scope of this part of ISO 10303.

If the representation of any schema in the XML document is specified using the markup declaration set specified in Annex C, the, `schema_decl` element shall be included in the `schema_decl` XML parameter entity declaration.

10.3 Representation of data

Any XML document that includes the representation of one or more sets of data corresponding to a EXPRESS schema as specified in this part of ISO 10303 shall include one or more `express_data` elements. The content of each `express_data` element shall be one of the following:

— the `schema_instance` element as specified in 7.1;

— an element replacing the `schema_instance` element as specified in 10.3.2.1 or 10.3.3.

This part of ISO 10303 specifies several methods for representing data corresponding to an EXPRESS schema in an XML document. These methods include the following:

— use of the late binding for data representation;

— use of the EXPRESS-typed Early Binding for data representation;

— use of the Object Serialization Early Binding for data representation;

— use of any markup for data representation that is architecturally compatible with the late binding for data representation.

Notations may be added to the internal subset for any external files referenced in the XML document.

EXAMPLE - This shows two graphics files being referenced.


```
<!DOCTYPE iso_10303_28 SYSTEM "URI of governing markup declaration
set" >
<!NOTATION jpg SYSTEM "something or other">
<!ENTITY xyz "file1.jpg" NDATA jpg>
<!ENTITY xxx "file2.jpg" NDATA jpg>
]>
```

10.3.1 Late binding XML documents

Annex B specifies the late binding markup declaration set for the representation of data corresponding to an EXPRESS schema. Clause 7 specifies the rules for the use of that markup declaration set to represent data. An XML document including data represented using the late binding shall conform to Annex B and Clause 7.

In the case that an XML document includes data represented using the late binding, the EXPRESS schema corresponding to each `schema_instance` element shall be identified using the `express_schema_name` XML attribute of the `schema_instance` element. The EXPRESS schema need not be referenced or represented in the XML document.

10.3.2 EXPRESS-typed Early Binding XML documents

Clause 8 specifies how to generate an ETEB markup declaration set for the representation of data corresponding to an EXPRESS schema. The XML document may be only an ETEB XML document (i.e., contains data which only conforms to the ETEB), or it may be a Mixed binding XML document. Requirements for the creation of Mixed binding XML document are specified in 10.3.4. This subclause specifies how an XML document conforming to the ETEB markup declaration set shall be constructed.

10.3.2.1 Construction of Document Type Declaration

The markup declarations specified in clause 8 constitute the DTD for an XML document that conforms to the ETEB as modified in this subclause.

A `schema_instance` XML parameter entity shall be included in the DTD. The value of the XML parameter entity shall be as follows.

If the data in the XML document is governed by a single schema, the value of the `schema_instance` XML parameter entity shall be the name of the context schema element type declaration as specified in 8.2.3.

EXAMPLE 1 - For the following EXPRESS schema :

```
SCHEMA mr_jones_garden; … END_SCHEMA;
```

The XML parameter entity is

```
<!ENTITY % schema_instance “Mr_jones_garden-schema”>
```


If the data in the ETEB XML document is governed by multiple schemas, the value of the `schema_instance` XML parameter entity shall be a choice of the names of the schema element type declarations as specified in 8.2.3.

EXAMPLE 2 - For the following EXPRESS schemas:

```
SCHEMA mr_jones_garden; … END_SCHEMA;
SCHEMA mr_smiths_garden; … END_SCHEMA;
```

The XML parameter entity is:

```
<!ENTITY % schema_instance "(Mr_jones_garden-schema |
Mr_smiths_garden-schema)">
```

10.3.2.2 Creating an ETEB XML document

An ETEB XML document shall begin with an XML declaration that includes the `standalone` XML attribute with a value of "no".

EXAMPLE 1- This is an example ETEB XML declaration.

```
<?XML version="1.0" standalone="no"?>
```

NOTE - The attribute "`standalone='no'`" is required because the markup declaration set specified in clause 8 uses default and fixed attribute values for many elements.

The XML declaration shall be immediately followed in the XML document by the following processing instruction (see Annex A.3 of ISO/IEC 10744:1997, as amended via Amendment 1) to establish the relationship between the ETEB as a client and the late binding as an architectural form replacing "iso_10303_28.dtd" with the appropriate `dtd-system-id` value:

```
<?IS10744 arch name="iso_10303_28"
    dtd-system-id="iso_10303_28.dtd"
    dtd-public-id="ISO 10303-28:2000//DTD
10303_28_Architectural_DTD//EN"
    form-att="late-bound-element"
    suppressor-att="late-bound-processing"
    renamer-att="late-bound-name"
    doc-elem-form="iso_10303_28"
    auto="nArcAuto"
    ?>
```

NOTE - Systems that recognize this as an architectural processing instruction will use the information in this element to process a document conforming to this clause as if it were a document that conforms to the late bound requirements specified in clauses 6 and 7. It will be ignored by non-architecturally-aware processors.

The XML declaration and architectural processing instruction shall be followed by the document type declaration. The name of the document type declaration shall be `iso_10303_28` and shall reference the markup declaration set specified in 10.3.2.1 that governs the document with a URI.

EXAMPLE 2 – An example DOCTYPE declaration follows:

```
<!DOCTYPE iso_10303_28 SYSTEM "Mr_jones_garden.dtd" []>
```

No element type declarations shall be appear in the local subset.

The root element of the XML document shall be an `iso_10303_28` element, which shall govern the remaining content of the document.

EXAMPLE 3 – A complete XML document example follows:

```
<?xml version="1.0" standalone="no"?>
<?IS10744 arch name="iso_10303_28"
    dtd-system-id="iso-10303-28.dtd"
    dtd-public-id="ISO 10303-28:2000//DTD
10303_28_Architectural_DTD//EN"
    form-att="late-bound-element"
    suppressor-att="late-bound-processing"
    renamer-att="late-bound-name"
    doc-elem-form="iso_10303_28"
    auto="nArcAuto"
    ?>
<!DOCTYPE iso_10303_28 SYSTEM "car_ownership.dtd">
<iso_10303_28 representation_category="ETEB">
 <express_data id="eteb_expdata1">
  <Car_ownership id="s1"
     express_schema_name="car_ownership"
     express_schema_identifier="C02000-03-21">
    <Person id="p1">
     <Person.name><string>John Q. Public</string>
      </Person.name>
     <Person.owns><car-ref refid="c1"/></Person.owns>
    </Person>
    <Car id="c1">
     <Car.make><string>BMW</string></Car.make>
     <Car.car_model><string>Z3</string></Car.car_model>
    </Car>
  </Car_ownership>
 </express_data>
</iso_10303_28>
```

10.3.3 Object Serialization Early Binding XML documents

Clause 9 specifies how to generate an OSEB markup declaration set for the representation of data corresponding to an EXPRESS schema. An XML document including data represented using the OSEB shall conform to clause 9.

For any `express_data` element representing data using the OSEB, the content of the element shall be a single `uos` element as specified in 9.3.4. A `schema_instance` XML parameter entity shall be included in the markup declaration set. The value of the XML parameter entity shall include “uos”

possibly qualified by an XML Namespace prefix. An element governed by the `uos` element type declaration shall be inserted into the content of the `express_data` element.

10.3.4 Mixed representation XML documents

For an XML document categorized as Mixed representation (see 5.1.3) the `schema_decl` and `schema_instance` XML parameter entities shall include the names as specified in this clause as appropriate for the EXPRESS schema representation and data binding included in the XML document.

