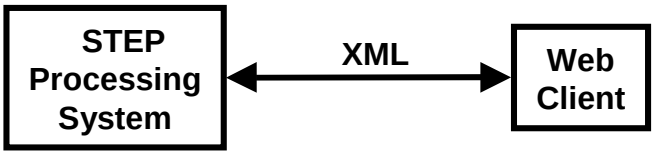
Containment Early Binding
Draft 1.6

1 Background and context (Informative)

1.1 Purpose of the Binding

The Containment Early Binding is designed to allow STEP processing systems to communicate information to client applications using XML. The binding allows information to be picked in a STEP processing system and sent to a web client in a form that is relatively easy to understand and process in the client.



The binding is called the Containment Early Binding (CEB) because it emphasizes the use of containment (or nesting) in the generated XML. With containment, a STEP entity that describes a value for another entity is nested inside the XML of the first entity. Nesting makes an XML description of STEP data easier to read because of the wide availability of browsers and data formatters for contained data, and it allows many features of the generated XML to be type checked using a DTD.

The CEB is a two phase binding. In the first phase an XML DTD is generated from an EXPRESS schema. This DTD may then be edited to contain only the XML elements and attributes required by the Web client. In the second phase the DTD and a data set defined by the EXPRESS schema are used to generate XML data for the elements and attributes that remain in the DTD.

1.2 In Scope Applications

The following applications are considered to be in scope for this binding:

- **Web Applications**. Many XML data formats use containment (see www.xml.org). A large range of tools are becoming available to process contained data.
- **Desktop and Front Office Applications**. Many applications need to display or change a small subset of the information in a STEP database. These subsets can be described as root instances in this binding.
- **Database Applications**. These applications need to process transactions against a STEP database. The content of those transactions can be described as roots instances in this binding.
- **Data Exchange Applications**. These applications exchange the necessary and sufficient root instances to describe a complete description of a model.

2 XML DTD Generation

2.1 Fundamental Concepts and Assumptions

2.1.1 Preconditions

This specification shall be applied to a list of entity definitions called roots. The creation of an XML DTD assumes that the EXPRESS definition of the roots meets the following preconditions:

(1) Syntactically correct EXPRESS (text file);

(2) No EXPRESS identifiers begin with the characters "xml";

(3) The roots belong to the same schema;

Section 2 of this specification describes the XML DTD elements and attributes that shall be generated for the roots and the EXPRESS entities and types that describe attribute values of the roots. The mechanism used to select the roots is not defined in this specification.

Section 2 of this specification also describes the data values that shall be used to populate an XML data file described by the XML DTD when an EXPRESS data file described by the EXPRESS is translated to EXPRESS.

Section 3 of this specification describes how a correspondence shall be established between the entity instances in the EXPRESS data file and the XML DTD when a translation is executed.

2.1.2 Entity Identification and Referencing

An entity that has been selected as a root shall be given an XML attribute with the name X-ID and the type ID.

An entity that is necessary to describe an attribute value of a root shall have an XML attribute with the name E-ID and the type CDATA unless it is a super-type or sub-type of a root.

An entity that is necessary to describe an attribute value of a root and is a sub-type or super-type of a root shall have an XML attribute with the name X-ID and the type ID.

An entity instance with an X-ID attribute shall be written into the XML data file once only. A value shall be generated for the X-ID attribute and this value shall be distinct from the values generated for any other X-ID or E-ID attribute in the XML file.

An entity instance with an E-ID attribute may be written into the XML data file zero, once or multiple times. An E-ID value shall be generated for the instance that is distinct from the E-ID and X-ID values generated for all other instances

in the file. Each copy of the instance written into the XML file shall have the same values for all its attributes including the E-ID attribute.

The referencing behavior (i.e., "pointing") of EXPRESS/Part 21 is mirrored in this binding using two methods. If the referenced entity has an E-ID attribute then the referencing shall be modeled by making the referenced entity a contained element of the referencing entity. In this method duplicate instances can be detected using the value of the E-ID attribute.

EXAMPLE 1 - The following is an example of how the "pointing behavior" of EXPRESS instances is managed in this binding XML when referencing is modeled using containment. For the EXPRESS declarations:

```
ENTITY parent;
first : child;
second : child;
END_ENTITY;

ENTITY child;
name : STRING;
END_ENTITY;
```

The Part 21 data is:

```
#10 = parent (#20, #20);
#20 = child ('Daphne');
```

With parent selected as root the XML declaration is:

```
<!ELEMENT STEP-XML (parent)*>
<!ELEMENT parent (first, second)>
<!ATTLIST parent
      X-ID  ID #REQUIRED>

<!ELEMENT first (child)>
<!ELEMENT second (child)>

<!ELEMENT child EMPTY>
<!ATTLIST child
      E-ID  CDATA #REQUIRED
      name  CDATA #REQUIRED>
```

The XML data is:

```
<STEP-XML>
   <parent X-ID = " #10" >
      <first>
            <child E-ID = " #20"  name = " Daphne" />
      </first>
      <second>
            <child E-ID = " #20"  name = " Daphne" />
      </second>
   </parent>
<STEP-XML/>
```

If the referenced entity has an X-ID attribute then the referencing shall be modeled by making the referencing entity contain an IDREF attribute that addresses the X-ID of the referencing entity.

EXAMPLE 2 - The following is an example of how the "pointing behavior" of EXPRESS instances is managed in this binding XML when referencing is modeled using IDREFs. For the EXPRESS declarations:

```
ENTITY parent;
first : child;
second : child;
END_ENTITY;

ENTITY child;
name : STRING;
END_ENTITY;
```

The Part 21 data is:

```
#10 = parent (#20, #20);
#20 = child ('Daphne');
```

With parent and child selected as roots the XML declaration is:

```
<!ELEMENT STEP-XML (parent | child)*>
<!ELEMENT parent EMPTY>
<!ATTLIST parent
      X-ID        ID    #REQUIRED
      first       IDREF #REQUIRED
      second      IDREF #REQUIRED>

<!ELEMENT child EMPTY>
<!ATTLIST child
      X-ID ID     #REQUIRED
      name CDATA  #REQUIRED>
```

The XML data is:

```
<STEP-XML>
      <parent X-ID = " #10"  first=" #20"  second=" #20" />
      <child X-ID = " #20"  name = " Daphne" />
</STEP-XML>
```

2.1.3 SIR Attributes

EXPRESS entity attributes defined by Simple Types, Enumerated Types and references to X-ID entities are called SIR (String, Integer and Real) attributes. EXPRESS entity attributes defined by Defined type of a SIR attribute type are also SIR attribute types provided the Defined type does not contain a WHERE rule.

EXAMPLE 3 - The following is an example of how SIR attributes are bound into XML. For the EXPRESS declarations:

```
ENTITY product;
name : label;
END_ENTITY;

ENTITY product_definition_formation;
of_product : product;
id : STRING;
END_ENTITY;

TYPE label = STRING; END_TYPE;
```

The Part 21 data is:

```
#10 = product ('a product');
#20 = product_definition_formation (#10, 'A1');
```

With product_definition_formation selected as root the XML declaration is:

```
<!ELEMENT STEP-XML (product_definition_formation)*>

<!ELEMENT product_definition_formation (of_product)>
<!ATTLIST product_definition_formation
      X-ID  ID          #REQUIRED
      id    CDATA       #REQUIRED>

<!ELEMENT of_product (product)>

<!ELEMENT product EMPTY>
<ATTLIST product
      E-ID  CDATA       #REQUIRED
      name  CDATA       #REQUIRED>
```

The XML data is:

```
<STEP-XML>
   <product_definition_formation X-ID=" #20"  id=" A1" >
      <of_product>
            <product E-ID=" #10"  name = " a product" />
      </of_product>
   </product_definition_formation>
</STEP-XML>
```

2.1.4 Synthetic Entities

When synthetic entities are generated for aggregate attributes and defined type attributes those synthetic entities shall be mapped into XML using the rules described in this specification for regular entities with the exception that they shall not be given an E-ID or X-ID attribute.

EXAMPLE 4 – In Example 5 information about the size and properties of the coordinates attributes is mapped into XML by defining a synthetic entity called "coordinates-item" as described in clause 2.2.6. The coordinates-item does not have an E-ID attribute.

2.1.5 Derived, Inverse, Unique and Where

EXPRESS Derived attributes shall be bound into the XML using the rules defined for Optional attributes of its type. Instances of these attributes shall be evaluated in the context of the EXPRESS data population. The string "D-" shall be added to the front of the name of the attribute to indicate that it is derived.

EXPRESS Inverse attributes that address a root entity shall be bound into the XML using the rules defined for Optional attributes. Instances of these attributes shall be evaluated in the context of the EXPRESS data population. The string "IV-" shall be added to the front of the name of the attribute to indicate that it is an Inverse.

EXPRESS inverse attributes shall be bound into the XML using the rules defined for attributes of the type Logical. For Inverse attributes that address a

root entity this shall produce a second binding of the attribute. For this binding instances of these attributes shall be evaluated in the context of the EXPRESS data population. If they conform to the data structure constraints described by the Inverse then the value “true” shall be written, if they do not conform then the value “false” shall be written. If the conformance cannot be determined then the value “unknown” shall be written. The string “IC-“ shall be added to the front of the name of the attribute to indicate that it is an Inverse.

If labeled, a Unique rule shall be bound into the XML as a Logical type with the name given by the label. Instances of these attributes shall be evaluated in the context of the EXPRESS data population. If another instance in the population has the same value in the Unique rule then the value “false” shall be written, if no other instance has this Unique value then the value “true” shall be written. If the uniqueness cannot be determined then the value “unknown” shall be written. The string “U-“ shall be added to the front of the name of the attribute to indicate that it is a Unique.

If labeled, a Where rule shall be bound into the XML as a Logical type with the name given by the label. Instances of these attributes shall be evaluated in the context of the EXPRESS data population. If they conform to the constraint described by the Where rule then the value “true” shall be written, if they do not conform then the value “false” shall be written. If the conformance cannot be determined then the value “unknown” shall be written. The string “R-“ shall be added to the front of the name of the attribute to indicate that it is a Where rule.

Where ordering is significant in the XML, attributes shall be bound to the XML file in the order Explicit, Derived, Inverse, Unique and WHERE. This ordering shall take precedence over the ordering defined in Section 2.1.6.

EXAMPLE 5 - The following is an example of how Derived and Where attributes are managed. For the EXPRESS declaration:

```
CONSTANT
Pie : REAL := 3.142;
END_CONSTANT;

ENTITY circle;
radius : positive_measure;
center : point;
DERIVE
area : Real := Pie * radius * radius;
END_ENTITY;

ENTITY point;
coordinates : LIST [2:3] OF measure;
END_ENTITY;

TYPE measure = REAL; END_TYPE;

TYPE positive_measure = measure;
WHERE valid: SELF > 0;
END_TYPE;
```

The Part 21 data is:

```
#10 = circle (10.0, #20);
#20 = point ((1.0, 1.0));
```

With circle selected as root the XML declaration is:

```
<!ELEMENT STEP-XML (circle)*>
<!ELEMENT circle (center)>
<!ATLLIST circle
      X-ID              ID    #REQUIRED
      radius            CDATA #REQUIRED
      D-area            CDATA #IMPLIED
      T-radius-valid    CDATA #REQUIRED>

<!ELEMENT center (point)>
<!ELEMENT point (coordinate)>
<!ATTLIST point
      E-ID        CDATA       #REQUIRED>

<!ELEMENT coordinates (coordinates-item)*>
<!ATTLIST coordinates
      C-unique          false       #FIXED
      C-ordered         true        #FIXED
      C-low_bound       2           #FIXED
      C-high_bound      3           #FIXED>

<!ELEMENT coordinates-item EMPTY>
<!ATTLIST coordinates-item
      value       CDATA       #REQUIRED>
```

The XML data is:

```
<STEP-XML>
   <circle X-ID=" #10"  radius=" 10" D-area=" true" >
                        T-radius-valid=" true" >
      <center>
            <point E-ID=" #20" >
                  <coordinates>
                        <coordinates-item value=" 1.0" />
                        <coordinates-item value=" 1.0" />
                  </coordinates>
            </point>
      </center>
   </circle>
</STEP-XML>
```

EXAMPLE 6 – Example 7 shows the XML mapping used for an Inverse attribute that addresses a non-root entity. Example 9 shows an Inverse attribute that addresses a root entity.

2.1.6 Entity and Attribute names

The names of attributes and entities shall be normalized to lower case.

When the names of EXPRESS attributes will cause a conflict in the XML DTD they shall be renamed as follows:

— The first part shall be the name of the entity that defines the attribute.
— A period symbol shall separate the first and third parts.
— The name of the attribute shall be the third part.

Two EXPRESS attributes belonging to the same schema that are not SIR attributes shall conflict if they are both necessary to describe one or more of the selected roots and if they have the same name. In this case both EXPRESS attributes shall be renamed.

An EXPRESS attribute shall conflict with an EXPRESS entity in the same schema if they have the same name and they are both necessary to describe one or more of the selected roots. In this case the EXPRESS attribute shall be renamed.

Two EXPRESS attributes that are SIR attributes and part of the same XML element as described in Clause 2.1.7 shall conflict if they have the same name.

EXAMPLE 7 - The following is an example of how attributes of EXPRESS instances are managed when they have the same name. The EXPRESS declarations in this example are taken from ISO 10303-203:1994. The STEP data is a subset of a valid Part 21 file defined by ISO 10303-203:1994.

NOTE - Most of the attribute names that might conflict do not because the attributes in conflict are SIR attributes.

```
SCHEMA configuration_controlled_design
ENTITY product;
  id                : identifier;
  name              : label;
  description       : text;
  frame_of_reference : SET [1:?] OF product_context;
UNIQUE
  UR1 : id;
END_ENTITY; -- product

ENTITY product_definition_formation;
  id          : identifier;
  description : text;
  of_product  : product;
UNIQUE
  UR1 : id, of_product;
END_ENTITY; -- product_definition_formation

ENTITY product_definition_formation_with_specified_source
  SUBTYPE OF (product_definition_formation);
  make_or_buy : source;
END_ENTITY; -- product_definition_formation_with_specified_source

ENTITY product_definition;
  id                : identifier;
  description       : text;
  formation         : product_definition_formation;
  frame_of_reference : product_definition_context;
END_ENTITY; -- product_definition

ENTITY design_context
  SUBTYPE OF (product_definition_context);
```

```
WHERE
  WR1: SELF.life_cycle_stage = 'design';
END_ENTITY; -- design_context

ENTITY product_definition_context
  SUBTYPE OF (application_context_element);
  life_cycle_stage : label;
END_ENTITY; -- product_definition_context

ENTITY mechanical_context
  SUBTYPE OF (product_context);
WHERE
  WR1: SELF.discipline_type = 'mechanical';
END_ENTITY; -- mechanical_context

ENTITY product_context
  SUBTYPE OF (application_context_element);
  discipline_type : label;
END_ENTITY; -- product_context

ENTITY application_context_element
  SUPERTYPE OF (ONEOF (product_context,
product_definition_context,
  product_concept_context));
  name              : label;
  frame_of_reference : application_context;
END_ENTITY; -- application_context_element

ENTITY application_context;
  application : text;
INVERSE
  context_elements : SET [1:?] OF application_context_element FOR
                        frame_of_reference;
END_ENTITY; -- application_context

TYPE identifier = STRING;
END_TYPE; -- identifier

TYPE label = STRING;
END_TYPE; -- label

TYPE text = STRING;
END_TYPE; -- text

TYPE source = ENUMERATION OF
  (bought,
   not_known,
   made);
END_TYPE; -- source
END_SCHEMA; -- configuration controlled design
```

The Part 21 data is:

```
#421420 = PRODUCT_DEFINITION ('DESIGN', '', #670750, #670720);
#670700 = APPLICATION_CONTEXT ('CONFIGURATION CONTROLLED 3D
DESIGNS OF MECHANICAL PARTS AND ASSEMBLIES');
#670720 = PRODUCT_DEFINITION_CONTEXT ('', #670700, 'DESIGN');
#670730 = MECHANICAL_CONTEXT ('', #670700, 'MECHANICAL');
```

```
#670740 = PRODUCT ('TIRE', 'TIRE', 'NOT SPECIFIED', (#670730));
#670750 = PRODUCT_DEFINITION_FORMATION_WITH_SPECIFIED_SOURCE
('3', 'LAST_VERSION', #670740, .MADE.);
```

With product definition and application context selected as roots the XML declaration is:

```
<!ELEMENT STEP-XML (product_definition | application_context)*>

<!ELEMENT product (product.frame_of_reference*)>
<ATTLIST product
      E-ID        CDATA       #REQUIRED
      id          CDATA       #REQUIRED
      name        CDATA       #REQUIRED
      description CDATA       #REQUIRED
      U-UR1       (true | false | unknown)      #REQUIRED>

<!ELEMENT product.frame_of_reference (product_context |
mechanical_context)*>
<!ATTLIST product.frame_of_reference
      C-unique    true        #FIXED
      C-ordered   false       #FIXED
      C-low_bound 1           #FIXED>

<!ELEMENT product_definition_formation (of_product)>
<!ATTLIST product_definition_formation
      E-ID        CDATA       #REQUIRED
      id          CDATA       #REQUIRED
      description CDATA       #REQUIRED
      U-UR1       (true | false | unknown)      #REQUIRED>

<!ELEMENT product_definition_formation_with_specified_source
(of_product)>
<!ATTLIST product_definition_formation_with_specificed_source
      E-ID        CDATA       #REQUIRED
      id          CDATA       #REQUIRED
      description CDATA       #REQUIRED
      make_or_buy (bought | not_known | made)   #REQUIRED
      U-UR1       (true |false |unknown)        #REQUIRED>

<!ELEMENT product_definition (formation,
product_definition.frame_of_reference)>
<!ATTLIST product_definition
      X-ID        ID          #REQUIRED
      id          CDATA       #REQUIRED
      description CDATA       #REQUIRED>

<!ELEMENT formation (product_definition_formation |
product_definition_formation_with_specified_source)>

<!ELEMENT product_definition.frame_of_reference
(product_definition_context | design_context)>

<!ELEMENT design_context EMPTY>
<!ATTLIST design_context
      E-ID                    CDATA       #REQUIRED
      name                    CDATA       #REQUIRED
      life_cycle_stage        CDATA       #REQUIRED
      frame_of_reference      IDREF       #REQUIRED
      R-WR1                   (true | false |unknown) #REQUIRED>

<!ELEMENT product_definition_context EMPTY>
<!ATTLIST product_definition_context
      E-ID                    CDATA       #REQUIRED
```

```
      name                    CDATA       #REQUIRED
      life_cycle_stage        CDATA       #REQUIRED
      frame_of_reference      IDREF       #REQUIRED>

<!ELEMENT mechanical_context EMPTY>
<!ATTLIST mechanical_context
      E-ID                    CDATA       #REQUIRED
      name                    CDATA       #REQUIRED
      discipline_type         CDATA       #REQUIRED
      frame_of_reference      IDREF       #REQUIRED
      R-WR1                   (true | false |unknown) #REQUIRED>

<!ELEMENT product_context EMPTY>
<!ATTLIST product_context
      E-ID                    CDATA       #REQUIRED
      name                    CDATA       #REQUIRED
      discipline_type         CDATA       #REQUIRED
      frame_of_reference      IDREF       #REQUIRED>

<!ELEMENT application_context_element EMPTY>
<!ATTLIST application_context_element
      E-ID                    CDATA       #REQUIRED
      name                    CDATA       #REQUIRED
      frame_of_reference      IDREF       #REQUIRED>

<!ELEMENT application_context EMPTY>
<!ATTLIST application_context
      X-ID                    ID          #REQUIRED
      text                    CDATA       #REQUIRED
      IC-context_elements     (true | false | unknown) #REQUIRED>
```

The XML data is:

```
<STEP-XML>
<product_definition X-ID=" #421420" id=" design"
   description = " " >
   <formation>
      <product_definition_formation_with_specified_source
         E-ID ="#670750" id = " 3"  description = " LAST_VERSION"
         make_or_buy = " made"  U-UR1 = " true" >
            <of_product>
               <product E-ID =" #670740" id =" TIRE"
                    name =" TIRE"  description =" NOT SPECIFIED"
                    U-UR1 =" true" >
                    <product.frame_of_reference >
                        <mechanical_context E-ID=" #670730"
                          name=" "
                           discipline_type = " MECHANICAL"
                           R-WR1 = " true"
                           frame_of_reference=" #670700" />
                     </product.frame_of_reference>
               </product>
            </of_product>
      </product_definition_formation_with_specified_source>
   </formation>
   <product_definition.frame_of_reference>
      <product_definition_context E-ID =" #670720" name=" "
            life_cycle_stage = " DESIGN"
            frame_of_reference=" #670700" />
   </product_definition.frame_of_reference>
</product_definition>
<application_context X-ID=" #670700" text = "CONFIGURATION
            CONTROLLED 3D DESIGNS OF MECHANICAL PARTS AND
            ASSEMBLIES" IC-context_elements="true" />
```

```
</STEP-XML>
```

2.1.7 Supertypes

The attributes of super-types of an entity shall be included as attributes of the entity using the algorithm defined in ISO 10303-21:1994 Clause 11.2.5.2 as reproduced below:

— All inherited attributes shall appear sequentially prior to the 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.

This procedure may result in an attribute being redefined. In this case the new definition shall be applied to the original definition of the attribute.

If a value is a complex instance then an entity name and SUBTYPE OF relationship is constructed for that instance using the algorithm defined in ISO 10303-22:1998 Clause A.1.3 with the character minus (“-”) replacing with the character minus (“+“).

An error shall be reported if a complex instance combines an X-ID entity with an entity that is not an X-ID.

EXAMPLE 8 - The following is an example of how EXPRESS inheritance is managed. For the EXPRESS declarations:

```
ENTITY x; END_ENTITY;
ENTITY y SUBTYPE OF (x); END_ENTITY;

ENTITY a;
att : x;
END_ENTITY;

ENTITY b SUBTYPE OF (a);
SELF\a.att : y;  (* Redeclare att *)
END_ENTITY;

ENTITY c SUBTYPE OF (a); END_ENTITY;
ENTITY d SUBTYPE OF (b, c); END_ENTITY;
```

With no root selection the XML declaration is:

```
<!ELEMENT x EMPTY>
<!ELEMENT y EMPTY>
<!ELEMENT a (a.att)>
<!ELEMENT a.att (x | y)>
<!ELEMENT b (b.att)>
<!ELEMENT b.att (y)>
<!ELEMENT c (a.att)>
<!ELEMENT d (b.att)>
<!ATTLIST x E-ID CDATA #REQUIRED>
<!ATTLIST y E-ID CDATA #REQUIRED>
<!ATTLIST a E-ID CDATA #REQUIRED>
<!ATTLIST b E-ID CDATA #REQUIRED>
<!ATTLIST c E-ID CDATA #REQUIRED>
<!ATTLIST d E-ID CDATA #REQUIRED>
```

2.1.8 Use/Reference Entities

If an entity or attribute is implicitly or explicitly interfaced into the schema that defines the roots and it is not given an Alias in the schema that defines the roots, then that entity or attribute shall be renamed as follows:

— The first part shall be the name of the schema that defines the interfaced entity or attribute.
— A period symbol shall separate the first and third parts.
— The name of the entity or attribute shall be the third part.

2.1.9 Limitations

This binding specification does not address the following EXPRESS features:

— functions;
— global rules;
— procedures;
— binary data type; (* because I do not understand why an information modeler who wants to treat binary as a reference did not model it that way *)

2.2 XML DTD

The following subclauses specify the declarations that must be present in the XML DTD for each feature of a root entity.

2.2.1 Schema independent declarations

2.2.2 Root Declaration

The XML DTD shall contain an element declaration that corresponds to the root of the binding declaration. The name of this element shall be “STEP-XML“;

The content model shall contain a content particle for each EXPRESS entity selected as a root and each non-abstract subtype of those EXPRESS entities. If there is more than one element then they shall be separated by the choice operator, “|”. The list of elements shall be followed by the XML cardinality operator zero-or-more applied (“*”).

EXAMPLE 9 - For the car_reg schema:

```
SCHEMA car_reg;

ENTITY person
name : label;
INVERSE
rich : BAG [2:?] car FOR owner;
END_ENTITY;

ENTITY car;
make : label;
owner : OPTIONAL person;
END_ENTITY;

TYPE label = STRING; END_TYPE;
END_SCHEMA;
```

the Part 21 data is:

```
#10 = CAR (“ Land Rover” , #40);
#20 = CAR (“ Jaguar” , #40);
#30 = CAR (“ Mini” );
#40 = PERSON (“ joe” );
#50 = PERSON (“ sue” );
```

With car and person selected as roots the corresponding XML element declarations are:

```
<!ELEMENT STEP-XML (car | person)*>

<!ELEMENT car EMPTY>
<!ATTLIST car
      X-ID  ID          #REQUIRED
      make  CDATA       #REQUIRED
      owner IDREF       #IMPLIED>

<!ELEMENT person (rich?)>
<!ATTLIST
      X-ID  ID                              #REQUIRED
      name  CDATA                           #REQUIRED
      IC-rich (true | false | unknown)    #REQUIRED>

<!ELEMENT IV-rich (IV-rich-item*)>
<!ATTLIST IV-rich
      C-ordered   false #FIXED
      C-unique    false #FIXED
      C-low_bound 2     #FIXED>

<!ELEMENT I-rich-item EMPTY>
<!ATTLIST I-rich-item
      value       IDREF       #REQUIRED>
```

the XML data is:

```
<STEP-XML>
      <car E-ID=” #10”  make=“ Land Rover”  owner=” #40” />
      <car E-ID=” #20”  make=“ Jaguar”  owner = “ #40” />
      <car E-ID=” #30”  make=“ Mini” />
      <person E-ID=” #40”  name=“ joe” IC-rich=” true” >
            <IV-rich>
                  <IV-rich-item value=” #10” >
                  <IV-rich-item value=” #20” >
            </IV-rich>
      </person>
      <person E-ID=” #50”  name=“ sue” IC-rich=” false” >
            <I-rich />
```

```
        </person>
    </STEP-XML>
```

2.2.3 Entity declarations

For each root EXPRESS entity and each EXPRESS entity necessary to describe a root, the DTD shall contain an element declaration corresponding to the EXPRESS entity. The name of the element declaration shall be the same as the name of the entity.

The content model shall contain a particle for each Non SIR attribute of the entity declaration. The name of the particle shall be as defined in Clause 2.1.6. The order of the particles in the content model shall be as defined in Clause 2.1.7; commas shall separate the particles.

If the entity contains SIR attributes only then the content model shall be EMPTY.

If a Non SIR attribute is OPTIONAL then the zero-or-one XML operator, "?", shall be appended to the corresponding particle in the content model.

2.2.4 SIR Attributes

For each SIR attribute, an XML attribute shall be defined for the content model of the entity containing the attribute. The name of the attribute shall be as described in Clause 2.1.6. The XML type of the attribute shall be:

— If the EXPRESS type is STRING or a defined type of STRING then the XML type shall be CDATA with the content encoded using the rules of XML.
— If the EXPRESS type is INTEGER or a defined type INTEGER then XML type shall be CDATA with the content encoded using the rules defined in ISO 10303-21:1994(E) Clause 7.3.1
— If the EXPRESS type is REAL or NUMBER or a defined type of REAL or NUMBER then the XML type shall be CDATA with the content encoded using the rules defined in ISO 10303-21:1994(E) Clause 7.3.2
— If the EXPRESS type is Boolean or a defined type of Boolean then the XML type shall be defined as the values (`true | false`).
— If the EXPRESS type is Logical or a defined type of Logical then the XML type shall be defined as the values (`false | true | unknown`)
— If the EXPRESS type is an enumerated type then the XML type shall be defined as a list of the values defined for the enumeration in the EXPRESS.
— If the EXPRESS type is a reference to an X-ID entity then the XML type of the attribute shall be IDREF.

If the EXPRESS attribute is OPTIONAL then it shall be defined as Implied otherwise it shall be defined as #Required.

2.2.5 Entity Attributes

An element shall be added to the DTD for each attribute that contains a reference to an entity. The name of the element shall be as described in Clause 2.1.6.

The content model shall be a set of particles separated by the XML choice (“|”) operator. A particle shall be generated for the entity if it is not Abstract and for all of its sub-types that are not Abstract.

2.2.6 Aggregate Attributes

An element shall be added to the DTD for each aggregate attribute. The name of the element shall be as described in Clause 2.1.6.

If aggregate is not an ARRAY OF OPTIONAL and the immediate underlying type of the aggregate is an entity that is not a root, then the content model shall be set of particles separated by the XML choice (“|”) operator. A particle shall be generated for the entity if it is not Abstract and for all of its sub-types that are not Abstract. The cardinality operator of zero-or-more, "*", shall be applied to the content model.

If the aggregate is an ARRAY OF OPTIONAL or the immediate underlying type is not an entity or the immediate underlying type is an entity that is a root then the content model shall be a particle defined as the name of the synthetic entity described below and the cardinality operator of zero-or-more, "*", shall be applied to the content model.

The synthetic entity shall be defined as follows.

— If the entity that owns the attribute is not a synthetic entity, then the name of the entity shall be the name of the attribute expanded as described in Clause 2.1.6 followed by the string “-item”. If the entity that owns the attribute is a synthetic entity, then the name of the new synthetic entity shall be the name of the old synthetic entity with an additional “-item” string appended.
— The entity shall have an attribute with the name “value”. This attribute shall have the underlying type of the aggregate.
— If the aggregate is an ARRAY OF OPTIONAL, then the entity shall have an attribute with the name “index”. This attribute shall have the integer data type. Instances of the attribute shall be set to the index of the corresponding non-null value in the Array.

DTD elements shall be generated for the synthetic entity using the rules described in this specification.

EXAMPLE 10 - the EXPRESS Declarations:

```
ENTITY a;
b : ARRAY[10:15] OF OPTIONAL something;
END_ENTITY;
```

shall cause the following synthetic entity to be generated:

```
ENTITY b-item;
value : something;
index : INTEGER;
END_ENTITY;
```

EXAMPLE 11 - the EXPRESS Declarations:

```
ENTITY a;
b : LIST[10:15] OF LIST OF REAL;
END_ENTITY;
```

shall cause the following synthetic entity to be generated:

```
ENTITY b-item;
value : LIST OF REAL;
END_ENTITY;
```

EXAMPLE 12 - the synthetic EXPRESS Declarations:

```
ENTITY b-item;
value : LIST OF REAL;
END_ENTITY;
```

shall cause the following synthetic entity to be generated:

```
ENTITY b-item-item;
value : REAL;
END_ENTITY;
```

EXAMPLE 13 - the EXPRESS Declarations:

```
ENTITY a;
b : ARRAY[10:15] OF something;
END_ENTITY;
ENTITY something; END_ENTITY;
```

shall not require a synthetic entity to be generated.

The XML element for the aggregate attribute shall be given the following XML attributes:

— A Fixed attribute with the name “C-unique”. This attribute shall be set to the value “true” if members of the aggregate are required to be unique and “false” otherwise.
— A Fixed attribute with the name “C-ordered”. This attribute shall be set to the value “true” if members of the aggregate are required to be ordered and “false” otherwise.
— If a low bound is declared for the aggregate an attribute with the name “C-low_bound”. This attribute shall be Fixed if the low bound is defined by a constant in the EXPRESS and the value shall be set to the value of this constant. This attribute shall be Implied and of the type CDATA if the low bound is defined by an expression and instances of this attribute shall be set to the computed value of the attribute if one can be evaluated.
— If a high bound is declared for the aggregate an attribute with the name “C-high_bound”. This attribute shall be Fixed if the high bound is defined by a constant in the EXPRESS and the value shall be set to the value of this constant. This attribute shall be Implied and of the type CDATA if the high bound is defined by an expression and instances of this attribute shall be set to the computed value of the attribute if one can be evaluated.

EXAMPLE 14 – For the EXPRESS Declarations:

```
ENTITY a;
b : LIST OF UNIQUE something;
END_ENTITY;
ENTITY something; END_ENTITY;
```

The following XML DTD elements and attributes shall be generated:

```
<!ELEMENT a b>
<!ELEMENT b something*>
<!ATTLIST b
      C-unique    false       #FIXED
      C-ordered   true        #FIXED>
<!ELEMENT something EMPTY>
```

EXAMPLE 15 – For the EXPRESS Declarations:

```
ENTITY a;
x : INTEGER
b : ARRAY[x:x+5] OF OPTIONAL INTEGER;
END_ENTITY;
```

The following XML DTD elements and attributes shall be generated:

```
<!ELEMENT a b>
<!ELEMENT b b-item*>
<!ATTLIST b
      C-unique    false       #FIXED
      C-ordered   true        #FIXED
      C-low_bound CDATA       #IMPLIED
      C-high_bound CDATA      #IMPLIED>

<!ELEMENT b-item  EMPTY>
<!ATTLIST b-item
      value       CDATA             #REQUIRED
      index       CDATA             #REQUIRED>
```

2.2.7 Defined Type Attributes

An element shall be added to the DTD for each the attribute. The name of the element shall be as described in Clause 2.1.6.

The content model shall be a set of particles separated by the XML choice ("|") operator. A particle shall be generated for each underlying non-root entity of the Defined type that is not abstract and for each subtype of those underlying non-root entities.

For each underlying type that defines an aggregate a synthetic entity shall be generated as follows:

— The name of the entity shall be the name of the underlying type that defines the aggregate.
— The entity shall have an attribute with the name "choice". This attribute shall have the type of the aggregate.

The name of the synthetic entity shall then be added to the set of particles generated for the content model and DTD elements shall be generated for the synthetic entity using the rules described in Clause 2.2.

If there are no underlying entity types or defined types of aggregate then the content model shall be EMPTY.

If any of the underlying types of the Defined type are Simple types or Enumerated types then an XML attribute shall be defined for the type as described in Clause 2.2.4. The name of this attribute shall be the name of the outermost type that is not a SELECT type and contains the underlying Simple type or Enumerated type. The attribute shall be Required if it defines the only value allowed for the attribute and Implied otherwise.

For each labeled Where rule defined for the type or any of the underlying types an attribute shall be added to the XML attribute list of the XML element. The name of the XML attribute for the rule shall be the character “T-“ followed by the name of the EXPRESS attribute followed by the “-“ character followed by the name of the label. The type of the attribute shall be the type described for Logical SIR attributes in Clause 2.2.4. The attribute shall be Implied if the type that contains the Where rule is Implied and Required otherwise.

If the XML element has one or more XML attributes then the zero-or-one XML operator, "?", shall be appended to the content model of the element.

A Required XML attribute called “S-path” shall be defined for the XML element for the Defined type attribute. The attribute shall have the type CDATA. Instances of the attribute shall include the name of the underlying type chosen for each SELECT in the path between the defined type attribute and the underlying value. The names shall be given in order starting with the name of the first selected. Each name shall be separated from any subsequent names by a period “.”.

EXAMPLE 16 - the EXPRESS Declarations:

```
ENTITY a;
b : something;
END_ENTITY;

TYPE something = SELECT (x, y, z, REAL); END_TYPE;
ENTITY x; END_ENTITY;
TYPE y = SELECT (m, INTEGER); END_TYPE;
TYPE m = INTEGER;
WHERE valid: SELF < 0;
END_TYPE;
TYPE z = LIST OF INTEGER; END_TYPE;
```

The following synthetic EXPRESS entities shall first be generated:

```
ENTITY z;
choice : LIST OF INTEGER;
END_ENTITY;
```

The following synthetic EXPRESS entities shall then be generated:

```
ENTITY z;
```

```
choice : LIST OF choice-item;
END_ENTITY;
ENTITY choice-item
value : INTEGER;
END_ENTITY;
```

The following XML DTD elements and attributes shall be generated:

```
<!ELEMENT a (b)>
<!ELEMENT b (x | z)?>
<!ATTLIST b
      m           CDATA       #IMPLIED
      real        CDATA       #IMPLIED
      integer     CDATA       #IMPLIED
      S-path      CDATA       #REQUIRED
      T-b-valid   CDATA       #IMPLIED>

<!ELEMENT z choice>
<!ELEMENT choice (choice-item)*>
<!ATTLIST choice
      C-unique    false       #FIXED
      C-ordered   true        #FIXED>

<!ELEMENT choice-item EMPTY>
<!ATTLIST choice-item
      value       CDATA             #REQUIRED>
```

EXAMPLE 17 - For the EXPRESS Declarations:

```
ENTITY person
owns : vehicle;
END_ENTITY;

ENTITY car;
make : label;
END_ENTITY;

ENTITY boat;
make : label;
END_ENTITY;

ENTITY amphibious_car SUBTYPE OF (car, boat);
END_ENTITY;

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

The Part 21 is:

```
#10 = PERSON(#40)
#20 = PERSON(#50)
#30 = PERSON('Boeing 77');
#40 = CAR ('Jeep');
#50 = BOAT ('Ingalls');
#60 = AMPHIBIOUS_CAR ('Sea Rover', 'BMW');
```

With person selected as root the XML element declarations are:

```
<!ELEMENT STEP-XML (person)*>
<!ELEMENT person  (owns)>
<!ATTLIST person
      X-ID        ID    #REQUIRED>

<!ELEMENT owns (car | amphibious_car)?>
<!ATTLIST owns
```

```
      S-path      CDATA       #REQUIRED
      plane       CDATA       #IMPLIED>

<!ELEMENT car EMPTY>
<!ATTLIST car
      E-ID        CDATA       #IMPLIED
      make        CDATA       #REQUIRED>

<!ELEMENT amphibious_car EMPTY>
<!ATTLIST amphibious_car
      E-ID              CDATA       #REQUIRED
      car.make          CDATA       #REQUIRED
      boat.make         CDATA       #REQUIRED>
```

The XML data is:

```
<STEP-XML>
      <person X-ID=" #10" >
            <owns S-Path=" car" >
                  <car E-ID=" #40"  make=" Jeep" />
            </owns>
      </person>
      <person X-ID=" #20" >
            <owns S-Path=" amphibious_car" >
                  <car E-ID=" #50"  car.make=" Sea Rover"
                        boat.make=" BMW" />
            </owns>
      <person X-ID=" #30" >
            <owns plane=" Boeing 77"
                  S-Path=" plane" />
      </person>
</STEP-XML>
```

3 XML Data Generation

3.1 Fundamental Concepts and Assumptions

3.1.1 Preconditions

This specification shall be applied to an XML DTD and a list of entity instances called roots. The creation of an XML file assumes that the entity instances meet the following preconditions:

(1) No strings containing syntactically significant white space;

(2) All the root instances belong to the same schema instance;

An error shall be reported if a direct or indirect cyclic reference is detected from a contained instance to itself while generating the XML data. Such errors can be avoided by appropriate selection of X-ID entities.

3.1.2 Operation

The XML DTD used to generate XML data from EXPRESS defined data need not be identical to the one generated for the EXPRESS by the specification given in Clause 2. The following types of changes can be made to the DTD after it has been generated and before it is applied to the EXPRESS defined data to generate XML.

— XML Elements and attributes may be deleted
— XML attributes may be redefined as XML elements

3.2 Name Correspondence

3.2.1 Name Correspondence for entities

A correspondence shall be established between a DTD element and an ordinary EXPRESS entity if it has the same name as that entity.

A correspondence shall be established between a DTD element and an EXPRESS AND/OR entity if the name of the DTD element is a subset of the name of the AND/OR entity.

3.2.2 Name Correspondence for SIR attributes

A correspondence shall be established between an XML attribute and an EXPRESS attribute if the XML element that owns the XML attribute corresponds to the EXPRESS entity that owns the attribute, and if the name of the XML attribute is the same as the name described in Clause 2.1.6.

3.2.3 Name Correspondence for Non SIR attributes

A correspondence shall be established between an XML element and an EXPRESS attribute if no correspondence has been established for that EXPRES attribute with an XML attribute and if the name of the XML element is the same as the name described in Clause 2.1.6.

3.3 XML Value generation

The XML value of each root entity instance shall be developed by traversing the XML DTD and finding the EXPRESS entity or EXPRESS attribute instance that corresponds to each element or attribute in the DTD.

If there is no corresponding EXPRESS entity or attribute instance for an XML element or attribute then the XML value for that XML element or XML attribute shall be empty.

An error shall be reported if the XML data generated from the EXPRESS data does not conform to the rules described by the DTD.