ISO TC184/SC4/WG11 N140

Date: 2000-10-06

Supercedes ISO TC184/SC4 WG11 N135

ISO/PDTS 10303-28
Product data representation and exchange: Implementation methods: XML representation of EXPRESS schemas and data

ABSTRACT

This part specifies the way in which XML can be used to represent both EXPRESS schemas and corresponding data.

KEYWORDS:

Implementation methods XML representation, EXPRESS, schema

COMMENTS TO READER:

This document has been reviewed and noted by the ISO TC 184/SC4 Secretariat and has been determined to be ready for this ballot cycle.

Project Leader:	Peter Bergström	**Project Editor**:	David Price (acting)
Address:	EuroSTEP AB Vasagatan 38 SE-111 20 Stockholm Sweden	**Address**:	IBM 5300 International Blvd. N. Charleston, SC USA 29418
Telephone:	+46 708 111 966	**Telephone**:	+01 843 760 4341
Telefacsimile:	+46 708 111 965	**Telefacsimile**:	+01 843 760 3349
Electronic mail:	peter.bergstrom@eurostep.com	**Electronic-mail**:	dmprice@us.ibm.com

International Organization for Standardization
Case Postale 56 • CH-1211 Genève 20 • Switzerland
Tel. + 41 22 749 01 11
Fax + 41 22 734 1- 79
E-mail copyright@iso.ch
Web www.iso.ch

Contents

Page

Figure

Foreword

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

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

In other circumstances, particularly when there is an urgent market requirement for such documents, a technical committee may decide to publish other types of normative documents:

— an ISO Publicly Available Specification (ISO/PAS) represents an agreement between technical experts in an ISO working group and is accepted for publication if it is approved by more than 50% of the members of the parent committee casting a vote;

— an ISO Technical Specification (ISO/TS) represents an agreement between the members of a technical committee and is accepted for publication if it is approved for publicationif it is approved by 2/3 of the members of the committee casting a vote.

An ISO/PAS or ISO/TS is reviewed every three years with a view to deciding whether it can be transformed into an International Standard

ISO/TS 10303-28 was prepared by Technical Committtee 184, *Industrial automation systems and integration*, Subcommittee SC4, *Industrial data.*

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

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

This part of ISO 10303 is a member of the implementation methods series.

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

Introduction

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

This part of ISO 10303 specifies means by which schemas specified using the EXPRESS language (ISO 10303-11) and data governed by EXPRESS schemas can be represented as an XML document (Extensible Markup Language W3C Recommendation). Readers of this part of ISO 10303 should have knowledge of the EXPRESS language, XML and XML-related standards in order to understand its technical content.

For the representation of EXPRESS schemas, this part of ISO 10303 specifies an XML markup declaration set based on the syntax of the EXPRESS language. EXPRESS text representation of schemas is also supported.

For the representation of data corresponding to an EXPRESS schema, this part of ISO 10303 takes two broad approaches. One approach is to specify a single markup declaration set that is independent of the EXPRESS schema and can represent data for any schema. This approach is called late binding. The second approach is to specify the results of the generation of a markup declaration set that is dependent on the EXPRESS schema. This approach is called early binding.

This part of ISO 10303 specifies one late binding and two early bindings. Future editions of this part of ISO 10303 are expected to include additional early bindings.

The markup declaration sets in this part of ISO 10303 are intended as formal specifications for the appearance of markup in conforming XML documents. These declarations may appear as part of Document Type Definitions (DTDs) for such documents. Future editions of this part of ISO 10303 are expected to include specifications for the use of XML Schema[3] for this purpose.

EXAMPLE 1 - For the following EXPRESS schema:

```
SCHEMA my_schema;
  ENTITY an_entity;
    attr1 : STRING;
  END_ENTITY;
END_SCHEMA;
```

The corresponding XML markup would be:

```
<schema_decl>
  <schema_id>my_schema</schema_id>
  <entity_decl>
    <entity_id>an_entity</entity_id>
    <explicit_attr_block>
      <explicit_attr>
        <attribute_id>attr1</attribute_id>
        <base_type><string/></base_type>
      </explicit_attr>
```



```
    </explicit_attr_block>
  </entity_decl>
</schema_decl>
```

The late binding is a single markup declaration set that can be used to represent data corresponding to any EXPRESS schema. Additionally, the late binding can be used as the base architecture for early bindings as specified by the Architectural Form Definition Requirements in ISO 10744:1997 (HyTime). One such early binding is specified in this part of ISO 10303. That early binding, called the EXPRESS-typed Early Binding, preserves many of the data types defined in EXPRESS schemas.

EXAMPLE 2 - For the following EXPRESS schema:

```
SCHEMA my_schema;
  ENTITY an_entity;
    attr1 : STRING;
  END_ENTITY;
END_SCHEMA;
```

a set of late bound data as XML markup might be:

```
<schema_instance express_schema_name="my_schema" id="id1">
  <entity_instance express_entity_name="an_entity" id="id2">
    <attribute_instance express_attribute_name="attr1">
      <string_literal>an attr1 value</string_literal>
    </attribute_instance>
  </entity_instance>
</schema_instance>
```

and the same data using the EXPRESS-typed Early Binding is:

```
<My_schema-schema id="id1" express_schema_name="My_schema"
express_schema_identifier="My_schema Edition 2">
  <An_entity id="id2">
    <An_entity.attr1><string>an attr1 value</string></An_entity.attr1>
  </An_entity>
</My_schema-schema>
```

Early bindings can also be specified that are not architecturally compatible with the late binding. In this part of ISO 10303, these bindings are related to the late binding through the specification of a means of transforming data represented using that early binding into the same data represented using the late binding. Transformations of this nature are specified that use XSLT (XSL Transformations W3C Recommendation). XSLT is a language for transforming one XML document into another XML document. One such early binding and XSLT transformation generation is specified in this part of ISO 10303. That early binding, called the Object Serialization Early Binding, is based on object serialization patterns for programming languages.

EXAMPLE 3 - For the following EXPRESS schema:

```
SCHEMA my_schema;
  ENTITY an_entity;
    attr1 : STRING;
  END_ENTITY;
END_SCHEMA;
```


a set of Object Serialization Early Binding data as XML markup might be:

```
<uos c="id2" schema="My_schema" unset="unset">
  <An_entity x-id="id2" Attr1-r="id3"/>
  <string x-id="id3">an attr1 value</string>
</uos>
```

Several components of this part of ISO 10303 are available in electronic form. This access is provided through the specification of Universal Resource Locators (URLs) that identify the location of these files on the Internet. If there is difficulty accessing these files contact the ISO Central Secretariat, or contact the ISO TC 184/SC4 Secretariat directly at: sc4sec@cme.nist.gov.

Industrial automation systems and integration – Product data representation and exchange – Part 28: Implementation methods: XML representation of EXPRESS schemas and data

1 Scope

This part of ISO 10303 specifies use of the Extensible Markup Language (XML) to represent schemas specified using the EXPRESS data specification language, ISO 10303-11, and data that is governed by EXPRESS schemas.

The following are within the scope of this part of ISO 10303:

— specification of XML markup declarations that enable EXPRESS schemas to be represented using XML;

— specification of a single XML markup declaration set that is independent of the EXPRESS schema and formally describes the XML representation of data governed by any schema;

NOTE 1 - XML markup declarations specified using this method are referred to as late bound, in that they may be used without change to represent data governed by any EXPRESS schema. This part of ISO 10303 allows for a number of choices in representing the data.

— for an arbitrary EXPRESS schema, specification of an XML markup declaration set that corresponds to the schema and formally describes the XML representation of data governed by that schema;

NOTE 2 - XML markup declarations specified using these methods are referred to as early bound, in that they are specific to a given EXPRESS schema.

— specification of the mapping between XML markup declarations corresponding to a specific schema and the XML markup declarations independent of any schema;

— specification of the form of XML documents containing EXPRESS schemas and data governed by EXPRESS schemas;

— specification of the representation of EXPRESS primitive data type values as element content and as XML attribute values.

The following are outside the scope of this part of ISO 10303:

— specification of XML markup declarations corresponding to an EXPRESS schema that depend on the semantic intent of the EXPRESS schema;

— specification of mappings from XML markup declarations to an EXPRESS schema;

NOTE 3 - Given a set of XML markup declarations and one or more corresponding data sets, it is feasible to create an EXPRESS schema that captures the semantic intent of the data. However, this requires an understanding of the meaning and use of the data that may not be captured by the XML markup declarations.

— specification of the mapping to an EXPRESS schema from an XML representation of that schema;

— specification of the mapping to an EXPRESS schema from XML markup declarations that have been derived from that schema;

— any mapping to or use of XML Schema [3].

2 Normative references

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

ISO 6093: 1985, *Information processing -- Representation of numerical values in character strings for information interchange.*

ISO/IEC 6429:1992, *Control functions for coded character sets.*

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

ISO/IEC 8824-1:1995, *Information technology – Open systems interconnection – Abstract syntax notation one (ASN.1) – Part 1: Specification of basic notation.*

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

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

ISO 10303-22:1997, *Industrial automation systems and integration – Product data representation and exchange – Part 22: Standard data access interface*.

ISO 10303-23:2000, *Industrial automation systems and integration – Product data representation and exchange – Part 23: C++ language binding to standard data access interface*.

ISO 10744:1997, *Information processing -- Text and office systems -- Hypermedia/Time-based Structuring Language (HyTime)* and its Amendment 1.

Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. Internet Engineering Task Force RFC 2045 November 1996 [cited 2000-08-15]. Available from World Wide Web: <http://www.ietf.org/rfc/rfc2045.txt>.

Uniform Resource Identifiers (URI): Generic Syntax. Internet Engineering Task Force RFC 2396 August 1998 [cited 2000-08-07]. Available from World Wide Web: <http://www.ietf.org/rfc/rfc2396.txt>.

URN Syntax. Internet Engineering Task Force RFC 2141 May 1997 [cited 2000-09-28]. Available from World Wide Web: <http://www.ietf.org/rfc/rfc2141.txt>.

Extensible Markup Language (XML) 1.0. World Wide Web Consortium Recommendation 10-February-1998 [cited 2000-04-26]. Available from World Wide Web: <http://www.w3.org/TR/1998/REC-xml-19980210>.

Namespaces in XML. World Wide Web Consortium Recommendation 14 January 1999 [cited 2000-04-26]. Available from World Wide Web: <http://www.w3.org/TR/1998/REC-xml-19990114>.

XML Linking Language (XLink) Version 1.0. World Wide Web Consortium Candidate Recommendation 3 July 2000 [cited 2000-08-08]. Available from World Wide Web: <http://www.w3.org/TR/2000/CR-xlink-20000703>.

XML Transformations (XSLT) Version 1.0. World Wide Web Consortium Recommendation 16 November 1999 [cited 2000-08-07]. Available from World Wide Web: <http://www.w3.org/TR/1999/REC-xslt-19991116>.

3 Terms, definitions, and abbreviations

3.1 Terms defined in ISO 10303-1

For the purpose of this part of ISO 10303, the following terms defined in ISO 10303-1 apply.

— data;

— information.

3.2 Other terms and definitions

For the purposes of this part of ISO 10303, the following terms and definitions apply. Terms defined in the XML 1.0 Recommendation and ISO 10303-11 are repeated below for convenience.

NOTE - Definitions copied verbatim from other standards are followed by a reference to the source standard in brackets. Definitions that have been adapted from other standards are followed by an explanatory note.

3.2.1
element
A bounded component of the logical structure of an XML document that has a type and that may have XML attributes and content.

NOTE – Adapted from XML 1.0 Recommendation.

3.2.2
entity data type reference
An XML element that specifies the instance identifier of an element and serves as a reference *to* that element *from* the point at which the entity data type reference element appears.

NOTE - An entity data type reference is a "pointer" to another element instance. The entity data type reference declaration does not correspond to any EXPRESS declaration, but rather is derived from an EXPRESS entity data type declaration. The name of entity data type reference declaration is called the entity data type reference identifier. See 8.1.2 for an explanation of this concept.

3.2.3
EXPRESS attribute
representation of a property of an EXPRESS entity data type

NOTE – Adapted from ISO 10303-11.

3.2.1
EXPRESS entity instance
entity (data type) instance
A named unit of data which represents a unit of information within the class defined by an entity. It is a member of the domain established by an entity data type. [ISO 10303-11]

3.2.5
markup declarations
XML element type, XML attribute-list, XML entity and XML notation declarations that provide a grammar for a class of XML documents

NOTE – Adapted from XML 1.0 Recommendation.

3.2.6
schema instance
an information set, grouped for some purpose, that is governed by a single EXPRESS schema

3.2.7
XML attribute
name-value pair associated with an XML element

NOTE – Adapted from XML 1.0 Recommendation.

3.2.8
XML document
A (text) data object that conforms to the XML requirements for being well-formed.

NOTE – Adapted from XML 1.0 Recommendation.

3.3 Abbreviations

For the purpose of this part of ISO 10303, the following abbreviations apply.

DTD	Document Type Definition
ETEB	EXPRESS-Typed Early Binding
HyTime	Hypermedia/Time-based structuring language
OSEB	Object Serialization Early Binding
SGML	Standard Generalized Markup Language
URI	Uniform Resource Identifier
URN	Uniform Resource Name
XLink	XML Linking Language
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations

3.4 Terminology

EXPRESS and XML use similar or identical words for different concepts. The qualification ‘XML’ and ‘EXPRESS’ are used to distinguish between the different contexts. These are used with the following terms:

— attribute;

— entity.

4 Fundamental concepts and assumptions

The EXPRESS language is a data specification language used to specify data types and constraints on data representing information to be exchanged or shared. Such a specification is given as an EXPRESS schema. ISO 10303 provides multiple implementation methods that can be used for data described by an EXPRESS schema.

The Extensible Markup Language (XML) is a subset of ISO 8879 Standard Generalized Markup Language (SGML) that has been specified to enable generic SGML to be served, received, and processed on the World Wide Web. It provides a syntax for constructing XML documents where the content of the XML document may be structured information as well as, or instead of, free text.

This part of ISO 10303 specifies how XML can be used to represent both EXPRESS schemas and data governed by EXPRESS schemas.

4.1 Early and late binding

Given an EXPRESS schema, this part of ISO 10303 describes two approaches for defining an XML markup declaration set for the schema. These two approaches are called *late binding* (or late bound) and *early binding* (or early bound).

A late bound XML markup declaration set is one that can be used in the same manner for any EXPRESS schema. It does not define any constructs that are specific to the schema.

An early bound XML markup declaration set is based on the specific EXPRESS schema and embeds specific aspects, such as names or structures, from the EXPRESS schema in the markup declaration set.

There are many possible markup declaration sets that can be constructed for both the late and early bindings. In this part of ISO 10303, one markup declaration set is specified for the late bound case in clause 7. For an arbitrary EXPRESS schema, clause 8 and clause 9 specify two different early bound markup declaration sets.

4.2 Mapping between bindings

This part of ISO 10303 specifies multiple bindings for the same schema resulting in different representations for a given set of data. Applications may need to transform data between these different representations. This part of ISO 10303 applies two approaches for mapping between bindings, architectures and XSL Transformations (XSLT).

For an arbitrary EXPRESS schema, clause 8 specifies an early bound markup declaration set that uses the late bound markup declaration set as architectural forms, as specified in ISO 10744 Annex A.3.

For an arbitrary EXPRESS schema, clause 9 specifies an early bound markup declaration for which an XSLT mapping to the late binding is specified.

NOTE - Figure 1 shows the relationship between different markup declaration sets included and enabled by this part of ISO 10303. This use of architectural forms enables the possibility to define other early-bound markup declaration sets that can be optimised for different purposes, e.g., for display, for data exchange, for compactness.

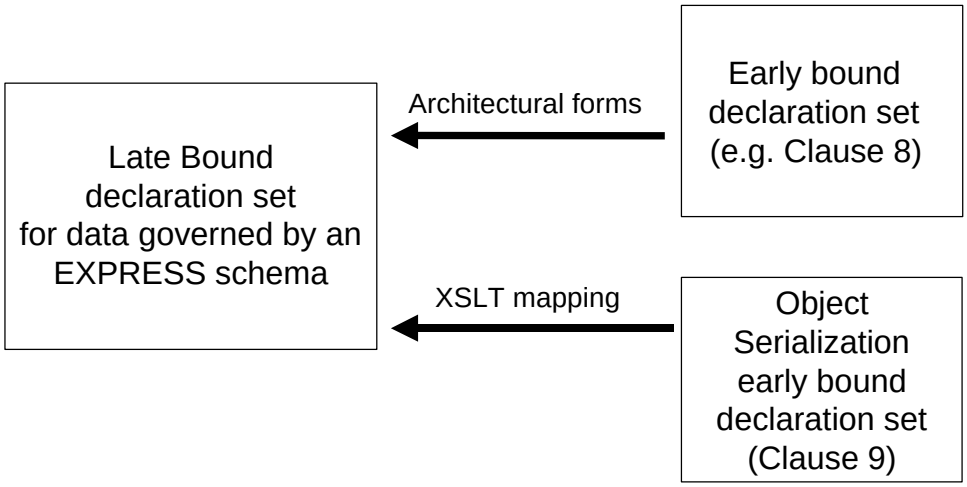

Figure 1 - Relationship between markup declaration sets

4.2.1 Use of Architectural forms

This part of ISO 10303 makes use of the concept of architectural forms (see ISO/IEC 10744 Annex A.3). Given an XML document that corresponds to a particular DTD, architectural forms provide a standard mechanism for processing it as if it (the client XML document) were consistent with another DTD (the meta-DTD, or base architecture).

The mapping from a client XML document to a base architecture is a property of the client XML document. With the XSLT mapping technique, the mapping is separated from the document. The architectural mapping is achieved by identifying, by means of XML attributes, the relationship between the client XML document and the base architecture, such that an application can recognise an element in the client XML document as equivalent to an architectural element. The client XML document can therefore be processed in terms of the base architecture, without actually requiring the instantiation of the architectural instance as a new XML document

Architectures place constraints on the extent of the differences between the client DTD and the base architecture DTD. The allowed flexibility is as follows:

— choice of names for element types;

— choice of names for XML attributes;

— choice of using XML attributes or content for data items;

— ability to define additional ‘wrapper’ element types that do not appear in the base architecture;

— ability to define extra data that does not appear at all in the base architecture.

This part of ISO 10303 defines a DTD (see Annex D) based on the late binding that is used as the base architecture for one early bound XML declaration set (see clause 8). Other architecturally compliant bindings may be defined. This will allow data sets represented using such early bound markup declaration sets to be processed as if they were defined in terms of the late-bound markup

declaration set. Thus, software can be written against the late bound markup declaration set that can, without modification, process data conforming to any compliant early bound declaration set.

An early bound markup declaration set is architecturally compliant to the late bound markup declaration set if it has that set as its base architecture.

4.2.2 Use of XSLT

This part of ISO 10303 makes use of XSLT in specifying the mapping between bindings. An XSLT stylesheet can be created that instructs an XSL processor in how to transform an input XML document based on one binding into an XML document based on a different binding.

In particular XSLT allows:

— an XML document, with or without a DTD, to be transformed to an XML document with a completely different structure;

— XML elements to be transformed into XML elements with a different name, ordering and/or structure;

— XML elements to be mapped to XML attributes;

— XML attributes to be mapped to XML elements;

— the ability to copy or omit data from a source XML document when mapping to the target XML document;

— the ability to add markup and data not present in the source XML document when mapping to the target document.

Annex H presents an informative algorithm for generating an XSLT stylesheet to transform an early bound XML data document, as specified in clause 9, into a late bound document, as specified in clause 7. The resulting XSLT stylesheet is schema-specific and can be used to transform any OSEB XML data governed by that particular EXPRESS schema into its corresponding late bound representation.

NOTE - This part of ISO 10303 does not specify conformance of processors in XSLT stylesheet generation for the purpose of mapping early bindings to the late binding. Annex H is intended to demonstrate guidelines for the development of software to generate XSLT stylesheets that provide a mapping of OSEB XML data to its late bound representation.

4.3 XML markup declaration sets and Document Type Declarations

This part of ISO 10303 specifies multiple ways to represent data corresponding to an EXPRESS schema in XML. A given binding specifies a set of XML markup declarations that corresponds to an EXPRESS schema. These markup declarations may be used in a DTD for an XML document. These markup declarations may also be combined with markup declarations based on another binding in a single DTD for an XML document. Any resulting name clashes can be resolved by the use of XML Namespaces (see clause 10).

4.4 XML names

4.4.1 Name restrictions in XML

The XML 1.0 Recommendation places the following restriction on the use of the characters in element type names:

> "Names beginning with the string "`xml`", or any string which would match `(('X'|'x') ('M'|'m') ('L'|'l'))`, are reserved for standardization in this or future versions of this specification."

Any early binding algorithm that derives XML markup declaration names from the EXPRESS entity data type, defined type, or attribute declarations must take this restriction into account.

4.4.2 Use of case in XML element type names

The EXPRESS language identifiers are case insensitive. XML names are case sensitive. In this part of ISO 10303, conventions are used for creating XML names based on EXPRESS identifiers in a uniform manner. The convention used is that the XML names are the same as the EXPRESS identifiers except that the initial letter is upper case and all other letters are lower case. Exceptions to this convention are enumerated in each early binding.

All XML names specified in this part of ISO 10303 that are not based on EXPRESS identifiers use the convention that all letters are lower case.

4.5 Use of ID, IDREF and the reftype XML attribute

XML ID is an XML attribute type. The value of an ID-typed attribute uniquely identifies the element that has the ID-typed attribute within the XML document. XML IDREF is an XML attribute type whose value must match the value of some ID-typed attribute within the XML document and is interpreted as a reference to the element so identified.

In the general case, a data set governed by an EXPRESS schema is a set of EXPRESS entity instances linked by references to one another. To reflect this, most of the bindings specified in this part of ISO 10303:

— require XML elements representing EXPRESS entity instances to have XML identifiers, that is, XML attributes of type ID; and

— require XML elements that refer to other XML elements representing EXPRESS entity instances to do so by use of XML attributes of type IDREF.

Certain other XML elements are also linked to one another in this way in various bindings.

The `reftype` attribute is commonly declared for elements that have `IDREF` attributes. This attribute uses a syntax specified in ISO 10744 that declares a constraint on the `IDREF` attribute(s) of the element. The general form of the value of this attribute is:


```
attribute-name ( element-type1 | ... | element-typeN )
```

where `attribute-name` is the name of an attribute of type `IDREF` and the list of alternative `element-type`s is a list of element types that may appear in the XML document. The interpretation of this constraint is that the type of the element identified by the (IDREF) value of the attribute named `attribute-name` shall be one of the element-types in the list.

5 Conformance

This clause specifies the requirements that must be satisfied by an XML document or a software application to claim conformance to this part of ISO 10303. Conformance is defined for the following items:

— an XML document;

— an XML document element;

— pre-processors that produce XML documents;

— post-processors that consume XML documents;

— processors that produce markup declarations from EXPRESS schemas.

5.1 Conformance of an XML document

5.1.1 Binding conformance

XML documents that conform to bindings specified in this part of ISO 10303 shall satisfy the following conditions:

— the XML document shall be a well-formed XML document as specified in XML 1.0;

— the root element of the document is an `iso_10303_28` element;

— if the XML document specifies a DTD, the markup declarations in the DTD shall be as specified in 5.1.3 as appropriate for the binding used;

— the XML document would be valid with respect to that DTD, even if the DTD is not present, adding the appropriate document type declaration if necessary.

5.1.2 Architectural conformance

XML documents that architecturally conform to this part of ISO shall satisfy the following conditions:

— the XML document shall be a well-formed XML document as specified in XML 1.0;

— the root element of the XML document uses the `iso_10303_28` element type declaration as an architectural form;

— the XML document shall be structured such that an XML architectural engine conforming to ISO/IEC 10744 Annex A.3 can process the client XML document architecturally with the DTD specified in Annex D as its base architecture.

5.1.3 XML document representation categories

XML document representation categories are classes of requirements on XML documents in addition to the requirements on all XML documents conforming to this part of ISO 10303.

NOTE - The markup declarations specified in clause 6 are included in all these categories except ARCH.

Category : SCHEMA_DECL

EXPRESS schema representation as markup. This category of representation consists of the XML document conforming to the `schema_decl` element type declaration specified in Annex C used in an XML document as specified in clause 10 and specifically 10.2.

Category : SCHEMA_TEXT

EXPRESS schema representation as text. This category of representation consists of the XML document conforming to the `schema_text` element type declaration specified in Annex C used in an XML document as specified in clause 10 and specifically 10.2.

Category : LB

Late binding representation. This category of representation consists of the XML document conforming to the late binding markup declaration set specified in clause 7 used in an XML document as specified in clause 10 and specifically 10.2.1.

Category : ETEB

EXPRESS-Typed Early Binding representation. This category of representation consists of the XML document conforming to an ETEB markup declaration set generated as specified in clause 8 used in an XML document as specified in clause 10 and specifically 10.2.2.

Category : OSEB

Object Serialization Early Binding representation. This category of representation consists of the XML document conforming to an OSEB markup declaration set generated as specified in clause 9 used in an XML document as specified in clause 10 and specifically 10.2.3.

Category : Mixed

Mixed representation. This category of representation consists of the XML document conforming to more than one of the markup declaration sets specified in the SCHEMA, LB, ETEB and OSEB categories.

Category : ARCH

Architecturally conformant. This category of representation consists of the XML document conforming to 5.1.2.

5.2 Conformance of an XML document element

In specifying the conformance of XML documents, other standards may require conformance of to one or more structural elements to this part of ISO 10303. This subclause defines conformance of such elements, although the referencing standard may specify additional conformance requirements that further restrict the set of conforming elements.

A structural element of an XML document that "conforms as a structural element" to this part of ISO 10303 shall satisfy the following conditions:

— the element is an `iso_10303_28` element, as defined in clause 6;

— the element is well-formed, as defined in the XML 1.0 Recommendation;

— if the document contains a DTD, the XML declarations for the `iso_10303_28` element and all elements in the content of that element shall be as specified in clauses 6, 7, 8, 9, 10, and Annex C as appropriate for the binding(s) used;

— The content of the `iso_10303_28` element would be valid with respect to that DTD, even if the DTD is not present, adding the appropriate document type declaration if necessary.

The content of any child element of the `iso_10303_28` element that corresponds to an EXPRESS simple data type or enumeration data type shall be encoded as specified in clause 10, 7.3, 8.3, or 9.9 as appropriate for the binding used.

NOTE 1 – The referencing specification should specify XML validity, if required, since XML validity is only defined for documents.

NOTE 2 – If the referencing specification allows more than one iso_10303_28 element to appear, then it should require each such element to satisfy the above requirements independently.

5.3 Conformance of a pre-processor

A software application is said to conform *as a pre-processor* if it produces XML documents that conform to this part of ISO 10303 as specified in 5.1. An application may conform as a pre-processor for arbitrary EXPRESS schemas, or it may conform as a pre-processor for certain EXPRESS schemas only. Conformance of a pre-processor shall be claimed based on the category of conforming XML documents it can produce and, if applicable, the EXPRESS schemas to which its XML document production is limited.

Conformance of a pre-processor that supports the ‘Mixed’ representation category specified in 5.1.3 shall be claimed based on the set of other representation categories it supports.

5.4 Conformance of a post-processor

A software application is said to conform *as a post-processor* if it processes XML documents that conform to one of the XML document representation categories specified in 5.1 and interprets them as

(a) EXPRESS schema(s) as specified in ISO 10303-11, or

(b) data governed by an EXPRESS schema(s) as specified in ISO 10303-11.

An application may conform as a post-processor for arbitrary EXPRESS schemas, or it may conform as a post-processor for certain EXPRESS schemas only. Conformance of a post-processor shall be claimed based on the category of conforming XML documents it can accept and, if applicable, the EXPRESS schemas to which its XML document processing is limited.

Conformance of a post-processor that supports the 'Mixed' representation category specified in 5.1.3 shall be claimed based on the set of other representation categories it can accept.

5.5 Conformance of a markup declaration generator

A software application conforms as a *markup declaration generator* if it can process any valid EXPRESS schema and produce markup declarations corresponding to the schema as specified in clause 8 or 9. The markup declarations shall be syntactically correct as specified in the XML 1.0 Recommendation.

6 Document level markup declarations

This clause specifies markup declarations that enable the association of XML document management and descriptive information with the XML representation of EXPRESS schemas and/or data governed by EXPRESS schemas in an XML document. Declarations are also specified that enable inter-document references to be represented. The early bindings specified in clauses 8 and 9 modify the `schema_instance` XML parameter entity (see 6.4.1); this allows the document descriptive elements defined in this clause to be used with the early bindings.

The XML Namespace URI for the document descriptive elements defined in this clause is "http://www.iso.org/10303-28/header/".

6.1 The iso_10303_28 element

The `iso_10303_28` element contains all other elements defined in this part of ISO 10303. The element type declaration is:

```
<!ELEMENT iso_10303_28
 (iso_10303_28_header?,
  (express_schema | express_data)+)>
<!ATTLIST iso_10303_28
```



```
        representation_category NMTOKENS #REQUIRED
        version CDATA #FIXED "PDTS">
```

The `representation_category` XML attribute identifies the representation(s), as specified in 5.1.3, to which the `express_data` in the document conforms. The tokens shall be one or more of the following: SCHEMA_DECL, SCHEMA_TEXT, LB, ETEB, OSEB or ARCH. The Mixed representation category shall be specified as a combination of two or more of these tokens.

The `version` XML attribute identifies the edition of this part of ISO 10303 to which the XML document conforms.

There is no significance to the ordering of the `express_schema` and `express_data` within the content of this element in an XML document.

The content elements of the `iso_10303_28` element and their XML attributes are described in the following subclauses.

6.2 Document header information

This subclause specifies information that is applicable to the entire XML document.

6.2.1 The iso_10303_28_header element

The `iso_10303_28_header` contains information that is applicable to the entire XML document.

The element type declaration is:

```
<!ELEMENT iso_10303_28_header
 (document_name, purpose?, time_stamp?, author?,
originating_organization?, authorization?, originating_system?,
preprocessor_version?, documentation?)>
```

6.2.2 The document_name element

The `document_name` element provides a human readable identifier for the XML document.

The element type declaration is:

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

6.2.3 The time_stamp element

The `time_stamp` element specifies the date and time when the XML document was created. The contents of the element shall conform to the extended format for the complete calendar date as specified in 4.2.1.1 of ISO 8601. The time of day may be included by appending the extended format for the time of the day as specified either in 4.3.1.1 or in 4.3.3 of ISO 8601 to the complete calendar date. The date and time shall be separated by the capital letter "T" as specified in 4.4.1 of ISO 8601.

The alternate formats of 4.3.1.1 and 4.3.3 permit the optional inclusion of a time zone specifier that is required if the time is not Coodinated Universal Time.

The element type declaration is:

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

6.2.4 The author element

The `author` element identifies the person or group of persons who created the XML document. This part of ISO 10303 places no further requirements on the content of the author element.

The element type declaration is:

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

6.2.5 The originating_organization element

The `originating_organization` element identifies the organization that created, or is responsible for, the XML document, and possibly an associated contact in that organization.

The element type declaration is:

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

6.2.6 The authorization element

The `authorization` element specifies the release authorization for the XML document and the signatory, where appropriate.

The element type declaration is:

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

NOTE - This may be distinct from the authorizations for various information units contained within the document.

6.2.7 The originating_system element

The `originating_system` element identifies the software system that created or captured the information contained in the XML document, including platform and version identifiers.

The element type declaration is:

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


6.2.8 The preprocessor_version element

The `preprocessor_version` element identifies the software system that created the XML document, including platform and version identifiers.

The element type declaration is:

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

NOTE - The `preprocessor_version` identifies the system that was used to produce the XML document. It may well be distinct from the software system that created or captured the original information.

6.2.9 The purpose element

If present, the `purpose` element shall contain a description of the reason the XML document was created to aid understanding by receiving parties.

The element type declaration is:

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

6.2.10 The documentation element

The `documentation` element provides a description of the contents of the element containing it.

The element type declaration is:

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

6.3 EXPRESS schema descriptive information

This subclause specifies information that is applicable to the EXPRESS schemas identified or represented in the XML document.

6.3.1 The express_schema element

The `express_schema` element shall contain one EXPRESS schema or external reference to an EXPRESS schema. The EXPRESS schema may be represented using the markup declaration set specified in Annex C or as text when contained within the XML document. The `schema_decl` XML parameter entity may be used to include or omit the `schema_decl` element from the content of the `express_schema` content model. If present, the `express_schema_description` XML attribute shall contain a human readable name.

One of the two following `schema_decl` XML parameter entity declarations shall appear in the markup declaration set prior to the `express_schema` element type declaration:

```
<!ENTITY % schema_decl "schema_decl |">

<!ENTITY % schema_decl "">
```

The element type declaration is:

```
<!ELEMENT express_schema
(%schema_decl; schema_text | external_refid)>
<!ATTLIST express_schema
      id ID #IMPLIED
      express_schema_description CDATA #IMPLIED
      express_schema_identifier CDATA #IMPLIED
      express_schema_version CDATA #IMPLIED >
```

6.3.2 The schema_text element

If present, the `schema_text` element shall contain a text representation of an EXPRESS schema as specified in ISO 10303-11, using whatever character encoding is specified in the XML processing instruction(s). The content shall be one of the following;

— a valid XML CDATA section as specified in XML 1.0 (see XML 1.0 section 2.7);

— a valid XML text string in which the character "<" is replaced by "<" and the character "&" is replaced by "&".

If the character sequence "]]>" appears in the EXPRESS schema, the valid XML text string form shall be used to represent the schema.

The element type declaration is:

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

EXAMPLE - The EXPRESS schema:

```
SCHEMA hello_world; END_SCHEMA;
```

would appear as:

```
<schema_text><![CDATA[SCHEMA hello_world; END_SCHEMA;]]>
</schema_text>
```

an XML CDATA section.

6.3.3 The schema_decl element

If present, the `schema_decl` element shall contain an EXPRESS schema represented according to the markup declaration set defined in Annex C.

6.4 Data descriptive information

This subclause specifies information associated with data that corresponds to an EXPRESS schema that is represented according to this part of ISO 10303.

6.4.1 The express_data element

The `express_data` element shall contain a single `schema_instance`. The `express_data` element may be named using the `name` XML attribute and may contain associated header information in a `data_section_header`. If present, the `name` XML attribute shall contain a human readable name. In an XML document containing data represented using more than one data representation binding (see 10.3), the `representation_category` XML attribute shall specify the specific binding used in the `express_data` element.

The XML parameter entity `schema_instance` is used to specify a particle of the content model of the `express_data` element. The value of the `schema_instance` XML parameter entity in a given DTD shall be a choice of content particles. The choice shall include all element types used in the XML document to represent schema instances.

EXAMPLE - The following XML parameter entity declaration allows ETEB data sets based on a schema named "cars", late bound data sets, OSEB data sets, and in an XML document.

```
<!ENTITY % schema_instance "(Cars-schema | schema_instance | uos )">
```

The markup declarations are:

```
<!ELEMENT express_data
 (data_section_header?, %schema_instance;)>
<!ATTLIST express_data
    id ID #REQUIRED
    name CDATA #IMPLIED
    representation_category NMTOKEN #IMPLIED>
```

6.4.2 The data_section_header element

If present, the `data_section_header` shall contain descriptive information applicable to the `express_data` element to that contains it.

The element type declaration is:

```
<!ELEMENT data_section_header
 (documentation?)>
```


6.5 Inter-XML document reference information

References between XML documents conforming to this part of ISO 10303 shall be represented using XML Linking Language (XLink). These references are represented as XLink "simple" links that support outbound links to remote resources.

NOTE - A remote resource might be a single XML element, an entire XML document or a portion of an XML document such as a schema instance.

6.5.1 References to remote XML resources

The `external_refid` element is a valid XML Linking Language (XLink) simple link. This element, called the linking element, allows for two types of references to remote resources, a reference to a single element in a remote resource in place of an XML IDREF or a reference to a group of elements in a remote resource to be included in the local XML document. The use of this linking element is described in the following two subclauses.

A markup declaration set conforming to this part of ISO 10303 shall contain the following markup declarations. If necessary, the XLink namespace may be added to the `external_refid` markup declaration or element.

```
<!ELEMENT external_refid EMPTY>
<!ATTLIST external_refid
  id ID #REQUIRED
  xlink:type CDATA #FIXED 'simple'
  xlink:href CDATA #REQUIRED
  xlink:arcrole CDATA #REQUIRED
  xlink:title CDATA #IMPLIED
  xlink:role CDATA #IMPLIED
  xlink:show CDATA #IMPLIED
  xlink:actuate CDATA #IMPLIED
>
```

6.5.2 Referencing a remote XML element

When an element in a local XML document refers to a single element in a remote XML document that reference is represented using the `external_refid` linking element. The local element for which a reference to a remote element is required shall contain a reference to the `id` XML attribute of the `external_refid` element. The XML attribute values for the `external_refid` element shall be as follows. The XML IDREF reference to the linking element and the link itself shall be resolved with the result that the local element refers to the remote element via the XML attribute in the local element that referred to the linking element.

The `xlink:arcrole` XML attribute value shall be the URI "urn:10303-28:6.5.2".

The `xlink:href` XML attribute value shall be a URI and be a reference to the single element in the remote resource to which the local element is to refer via the linking element.

If present, the `xlink:title` XML attribute value shall be a human readable description of the remote element.

If present, the `xlink:role` attribute value shall be a URI that refers to a remote resource describing a property of the remote. If the remote element is represented using one of the markup declaration sets specified in this part of ISO 10303, the value of the `xlink:role` attribute shall be "urn:10303-28:5.1.3:<category>" where "<category>" is replaced with the name one of representation categories "SCHEMA", " LB", "ETEB", " OSEB" or "ARCH" as specified in 5.1.3.

If present, the `xlink:show` and `xlink:actuate` values shall be one of the allowed values as specified in XLink.

EXAMPLE - In the following ETEB example, an element type named `Garden` refers to a remote element type named `Greenhouse` by a child element type named `Greenhouse-ref`.

In one ETEB XML document:

```
<iso_10303_28 representation_category="ETEB">
...
<Garden id="igard01">
  ...
  <Garden.has_greenhouse>
    <Greenhouse-ref refid="iextref01"/>
  </Garden.has_greenhouse>
</Garden>
<external_refid id="iextref01" xlink:type="simple"
    xlink:title="sample reference to greenhouse"
    xlink:role="urn:10303-28:5.1.3:ETEB"
    xlink:arcrole="urn:10303-28:6.5.2"
    xlink:href="http://docname.xml#xpointer(id('igh01'))"/>
...
</iso_10303_28>
```

In a second ETEB XML document named 'docname.xml':

```
<iso_10303_28 representation_category="ETEB">
...
<Greenhouse id="igh01">
  ...
</Greenhouse>
...
</iso_10303_28>
```

6.5.3 Including components in documents

Including components of one XML document conforming to this part of ISO 10303 in another XML document that conforms to this part of ISO 10303 is accomplished by using the `external_refid` linking element. The XML attribute values for the `external_refid` element shall be as follows.

The `xlink:arcrole` XML attribute value shall be the URI "urn:10303-28:6.5.3".

The `xlink:href` XML attribute value shall be a URI and be a reference to either a single element in the remote resource or to a group of elements in the remote resource. If the reference is to a single element, that element and all of its children are included in the local XML document. If the reference is to a group of elements, the entire group is included in the local XML document.

If present, the `xlink:title` XML attribute value shall be a human readable description of the remote element.

If present, the `xlink:role` attribute value shall be a URI that refers to a remote resource describing a property of that resource. If the remote elements are represented using one or more of the markup declaration sets specified in this part of ISO 10303, the value of the `xlink:role` attribute shall be "urn:10303-28:5.1.3:<category>"where "<category>" is replaced with the name of one or more of the representation categories "SCHEMA", "LB", "ETEB", "OSEB" or "ARCH" as specified in 5.1.3.

If present, the `xlink:show` and `xlink:actuate` values shall be one of the allowed values as specified in XLink.

EXAMPLE - In the following example, one XML document contains data and a reference to a schema represented as text in another XML document.

In one mixed XML document referencing a schema and containing late bound data:

```
<iso_10303_28 representation_category="SCHEMA_TEXT LB">
<express_schema>
<external_refid id="id_extref01"
    xlink:type="simple"
    xlink:title="included schema text"
    xlink:role="urn:10303-28:5.1.3:SCHEMA"
    xlink:arcrole="urn:10303-28:6.5.3"
xlink:href="http://dataname.xml#xpointer(/iso_10303_28/express_schema
/schema_text))"/>
</express_schema>
<express_data id="lb001" representation_category="LB">
...
</express_data>
</iso_10303_28>
```

In a second XML document named ‘dataname.xml’:

```
<iso_10303_28 representation_category="SCHEMA_TEXT">
<express_schema id='id_schema1'>
<schema_text>SCHEMA hello_world; (* remark &amp;amp; explanation
*)END_SCHEMA;</schema_text>
</express_schema>
</iso_10303_28>
```


7 Late bound markup declarations for EXPRESS-defined data

This clause specifies a late bound method of representing data corresponding to an EXPRESS schema. This late binding allows data governed by an arbitrary EXPRESS schema to be represented using the same markup declaration set.

This late bound markup declaration set acts as part of the base architecture for the definition of client markup declaration sets that can then be processed according to the late bound markup declaration set. This late bound markup declaration set includes options that facilitate the specification of client markup declaration sets.

Annex B provides the complete markup declaration set for the late bound representation of data. This clause specifies how that markup declaration set shall be used to represent data corresponding to an EXPRESS schema.

The late binding markup declaration set (see Annex B) and the EXPRESS language markup declaration set (see Annex C) share several elements. The following elements are common to those markup declaration sets:

— `binary_literal`;

— `boolean_literal`;

— `embedded_remark`;

— `enumeration_ref`;

— `false`;

— `indeterminate`;

— `integer_literal`;

— `logical_literal`;

— `real_literal`;

— `schema_decl`;

— `string_literal`;

— `tail_remark`;

— `true`;

— `unknown`.

All XML attributes in the late binding markup declaration set that correspond to EXPRESS identifiers shall have a value that is the same as the EXPRESS identifier all in lower case.

7.1 Representation of a schema instance

A schema instance is an information set, grouped for some purpose, that is governed by a single EXPRESS schema. The content of the `schema_instance` element constitutes one schema instance. The EXPRESS schema need not be present in the same XML document.

The `express_schema_identifier` XML attribute shall specify a unique identifier for the EXPRESS schema. For EXPRESS schemas defined in other parts of ISO 10303 the information object identifier shall be used as the identifier.

NOTE 1 - The `express_schema_identifier` can be a URI.

The `express_schema_name` XML attribute shall specify the name of the EXPRESS schema that governs the schema instance. This is an EXPRESS schema_id (see ISO 10303-11:9.3) and, when the EXPRESS schema is represented in the XML document using the markup declarations in Annex C, it corresponds with the content of the `schema_id` element in the `schema_decl` element representing the EXPRESS schema.

If present, the `express_schema_version` XML attribute shall specify the version of the EXPRESS schema referenced by the `express_schema_name` XML attribute.

If present, the `xml:lang` XML attribute shall specify the primary natural language in which the content of the schema instance is written (see XML 1.0 section 2.12).

The `complete_population` XML attribute indicates whether or not a complete population is included in the `schema_instance`. A complete population is a set of data that can be validated against the EXPRESS schema without additional data. A partial population is one that is not intended by the sender to be validated against the EXPRESS schema without having access to the data that completes the population.

NOTE 2 - If a complete population is not included, readers need to know where the rest of the population is, by prior agreement with the sending party.

In the case that the EXPRESS schema for the schema instance is included in the document or referenced from the document, the `refid` XML attribute shall reference the `express_schema` representing the EXPRESS schema.

EXAMPLE - This example shows a schema_instance element with a reference to an identified schema on the Web and containing two example representations of EXPRESS entity instances.

```
<schema_instance
express_schema_identifier="http://www.someplace/myexpress.exp"
express_schema_name="mr_jones_garden" express_schema_version="6">
  <entity_instance express_entity_name="ent1" id="id1">
  </entity_instance>
```



```
    <entity_instance_as_group id="id2">
      <partial_entity_instance express_entity_name="ent2">
      </partial_entity_instance>
    </entity_instance_as_group>
  </schema_instance>
```

7.2 Representation of an EXPRESS entity instance

The value of each EXPRESS entity instance shall be represented by one of:

— an `entity_instance` element;

— an `entity_instance_as_group` element.

NOTE 1 – An EXPRESS simple entity instance is an EXPRESS entity instance that is completely described by a *single* EXPRESS entity data type. An EXPRESS complex entity instance is an instance whose description involves more than one EXPRESS entity data type, even when only one of them contains explicit attributes. An EXPRESS simple entity instance can be an instance of a supertype, as long as it is not an instance of any subtype, but an instance of a subtype is always an EXPRESS complex entity instance.

NOTE 2 - The selection of mapping depends on the EXPRESS entity instance rather than the EXPRESS entity data type. It is possible for different instances of the same EXPRESS entity data type to use different mappings.

An EXPRESS simple entity instance shall be represented by one of the following:

— an `entity_instance` element containing no `partial_entity_instance` elements;

— an `entity_instance_as_group` element containing exactly one `partial_entity_instance`.

An EXPRESS complex entity instance shall be represented by one of the following:

— an `entity_instance` element containing at least one `partial_entity_instance` element;

— an `entity_instance_as_group` containing more than one `partial_entity_instance`.

Each EXPRESS partial complex entity value corresponding to a single EXPRESS entity data type participating in an EXPRESS complex entity instance shall be represented by one of the following:

— an `entity_instance` element;

— one `partial_entity_instance` element.

The EXPRESS attributes and `partial_entity_instance` elements may appear in any order and `partial_entity_instance` elements may be nested in any way allowed by the markup declaration set.

7.2.1 Representation using the entity_instance element

The value of the `express_entity_name` XML attribute shall be the name of the EXPRESS entity data type that is the type of the EXPRESS entity instance in the schema where the EXPRESS entity data type is declared. This definition shall apply to all XML attributes named `express_entity_name`.

For EXPRESS entity data types that are interfaced into the governing schema, the value of the `express_schema_name` XML attribute shall be the name of the schema in which the EXPRESS entity data type is declared. Otherwise, this XML attribute shall be omitted.

In the case that that the EXPRESS entity instance is an EXPRESS constant, the value of the `express_constant_name` XML attribute shall be the name of the EXPRESS constant which defines the EXPRESS entity instance in the name scope of the governing EXPRESS schema. The `express_constant_name` XML attribute shall be omitted in all other cases.

The content of the `entity_instance` element shall be the values of the EXPRESS attributes of the entity instance, represented as specified in 7.3, and any `partial_entity_instance` elements that are part of the complex entity instance.

EXAMPLE - An example `entity_instance` element follows.

```
<entity_instance express_entity_name="real_value_range" id="rvr2"
express_schema_name="support_items">
<attribute_instance express_attribute_name="minimum_value">
<real_literal>-5.0</real_literal>
</attribute_instance>
<attribute_instance express_attribute_name="maximum_value">
<real_literal>20.0</real_literal>
</attribute_instance>
<partial_entity_instance express_entity_name="temperature_range"
id="tr2">
</partial_entity_instance>
</entity_instance>
```

7.2.2 Representation using entity_instance_as_group

The value of the `express_entity_name` XML attribute shall be as specified in 7.2.1 for the XML attribute of the same name in an `entity_instance` element.

The value of the `express_constant_name` XML attribute shall be as specified as in 7.2.1 for the XML attribute of the same name in an `entity_instance` element.

The content of the `entity_instance_as_group` element shall be the `partial_entity_instance` elements that are part of the complex entity instance. The manner in

which the `partial_entity_instance` elements are nested is not specified in this part of ISO 10303.

EXAMPLE - An example `entity_instance_as_group` element follows.

```
<entity_instance_as_group id="po1">
<partial_entity_instance express_entity_name="pond">
<attribute_instance express_attribute_name="maintained_by">
<entity_instance_ref refid="wts1"/>
</attribute_instance>
</partial_entity_instance>
<partial_entity_instance express_entity_name="fish_pond" id="fp1">
<attribute_instance express_attribute_name="fish">
<set_literal>
<type_literal express_type_name="fish_type">
<enumeration_ref>goldfish</enumeration_ref>
</type_literal>
</set_literal>
</attribute_instance>
</partial_entity_instance>
</entity_instance_as_group>
```

7.3 Representation of EXPRESS attributes

Explicit attributes of an EXPRESS entity data type shall be represented in the XML document and EXPRESS derived and inverse attributes may be represented in the XML document. Specific requirements, however, apply to EXPRESS OPTIONAL explicit attributes (see 7.3.2), EXPRESS explicit attributes whose values are EXPRESS entity instances (see 7.3), and all redeclarations of EXPRESS explicit attributes (see 7.3.5, 7.3.6 and 7.3.7).

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

EXPRESS attributes are represented using the `attribute_instance` and `inherited_attribute_instance` elements, and the requirements below apply to both these elements.

The `inherited_attribute_instance` element may be used to represent an EXPRESS attribute that is inherited by an EXPRESS entity data type from one of its supertypes provided that the supertype is not explicitly represented by an `entity_instance` or `partial_entity_instance` element.

In the case where two EXPRESS attributes of the same name are inherited by an EXPRESS entity data type, the `inherited_attribute_instance` element shall not be used to represent either EXPRESS attribute.

Each EXPRESS attribute value may be represented only once, either by `attribute_instance` or `inherited_attribute_instance`, within the context of an `entity_instance` or an `entity_instance_as_group`.

EXAMPLE 1 - An example `attribute_instance` element follows.

```
<attribute_instance express_attribute_name="minimum_value">
<real_literal>-5.0</real_literal>
</attribute_instance>
```

EXAMPLE 2 - An example `inherited_attribute_instance` element follows.

For the EXPRESS fragment:

```
ENTITY person;
 age: INTEGER;
END_ENTITY;

ENTITY employed_person
 SUBTYPE OF (person);
 salary: INTEGER;
END_ENTITY;
```

data could be represented as:

```
<entity_instance express_entity_name="employed_person" id="xx">
 <attribute_instance express_attribute_name="salary">
  <integer_literal>50000</integer_literal>
 </attribute_instance>
 <inherited_attribute_instance express_attribute_name="age">
  <integer_literal>50</integer_literal>
 </inherited_attribute_instance>
</entity_instance>
```

7.3.1 Representation of EXPRESS mandatory explicit attributes

Each EXPRESS mandatory explicit attribute appearing in the corresponding EXPRESS entity data type shall be represented within an `entity_instance` element and a `partial_entity_instance` element by the `attribute_instance` or `inherited_attribute_instance` element.

The `express_attribute_name` XML attribute of the `attribute_instance` and `inherited_attribute_instance` element shall be the name of the EXPRESS attribute, in lower case. The content of the `attribute_instance` or `inherited_attribute_instance` element shall depend on the data type of the EXPRESS attribute, as specified in 7.4.

The `express_attribute_type` XML attribute of the `attribute_instance` or `inherited_attribute_instance` element shall have a value of "explicit" for explicit attributes. This is the default value defined in the markup declaration and therefore the value need not be specified in the XML document.

7.3.2 Representation of EXPRESS OPTIONAL explicit attributes

An EXPRESS explicit attribute that is declared to be OPTIONAL is not required to have a value in a given EXPRESS entity instance. When a value for the EXPRESS attribute is present in the EXPRESS entity instance being represented, it shall be represented as specified in 7.3.1. When no value for the

EXPRESS attribute is present in the EXPRESS entity instance, the corresponding element is omitted from the content of the element representing the EXPRESS entity instance.

The `express_attribute_type` XML attribute of the `attribute_instance` or `inherited_attribute_instance` element shall have a value as defined for mandatory explicit attributes.

7.3.3 Representation of EXPRESS INVERSE attributes

EXPRESS inverse attributes may be represented in the XML document.

The `express_attribute_type` XML attribute of the `attribute_instance` or `inherited_attribute_instance` element shall have a value of "inverse" for EXPRESS inverse attributes.

7.3.4 Representation of DERIVEd attributes

The derived attributes of an EXPRESS entity data type may be represented in the XML document. When an EXPRESS derived attribute in a subtype redeclares an explicit attribute from a supertype, however, the representation shall be as described in 7.3.5.

The `express_attribute_type` XML attribute of the `attribute_instance` or `inherited_attribute_instance` element shall have a value of "derived" for derived attributes.

EXAMPLE - An example `attribute_instance` element representing an EXPRESS derived attribute follows.

```
<attribute_instance express_attribute_name="circumference"
express_attribute_type="derived">
<real_literal>-5.0</real_literal>
</attribute_instance>
```

7.3.5 Representation of redeclared attributes

If an EXPRESS subtype entity data type redeclares an attribute of one of its supertypes the attribute value shall be represented as an attribute of the supertype, as specified in 7.3.1 or 7.3.2, using the data type of the attribute declared in the supertype to determine the value representation. The redeclared attribute shall be ignored, that is, it shall not be considered an attribute of the subtype for representation purposes.

NOTE - In this case, redeclaration may affect the possible values of the attribute, but it does not affect the representation of those values. In particular, a supertype attribute with data type REAL redeclared in a subtype as data type INTEGER is represented as a REAL value in the supertype.

7.3.6 Representation of explicit attributes redeclared as DERIVEd

If an EXPRESS subtype entity data type redeclares an EXPRESS attribute of one of its supertypes as a DERIVEd attribute, and if the (original) attribute is declared in the supertype as a mandatory

explicit attribute, the value of the attribute shall be represented by an `attribute_instance` or `inherited_attribute_instance` element in the content of the EXPRESS entity instance element representing the supertype.

7.3.7 Attributes redeclared as INVERSE

If an EXPRESS subtype entity data type redeclares an EXPRESS attribute of its supertype as an INVERSE attribute, there is no effect on the representation. The redeclared EXPRESS attribute is not represented in any way.

7.4 Representation of EXPRESS attribute values

7.4.1 Representation of primitive data types

For any literal value which represents the value of an EXPRESS constant, the XML attribute `express_constant_name` may be specified on that primitive. The value shall be the name of the EXPRESS constant. This XML attribute shall only be present when the value is defined as a constant in the EXPRESS schema.

EXAMPLE - An example `attribute_instance` element representing an EXPRESS attribute with a value that is an EXPRESS constant follows.

```
<attribute_instance express_attribute_name="length">
<real_literal
express_constant_name="default_length">5.0</real_literal>
</attribute_instance>
```

7.4.1.1 Boolean and logical data types

A value of EXPRESS boolean data type shall be represented in an XML document using the `boolean_literal` element, and values of the EXPRESS logical data type shall be represented in an XML document using the `logical_literal` element.

The value TRUE shall be represented using the element `true`; the value FALSE shall be represented using the element `false`; and the value UNKNOWN shall be represented using the element `unknown`.

EXAMPLE - An example `attribute_instance` element representing an EXPRESS attribute with a boolean value of FALSE follows.

```
<attribute_instance express_attribute_name="visible">
<boolean_literal><false/></boolean_literal>
</attribute_instance>
```


7.4.1.2 Integer data type

A value of EXPRESS integer data type shall be represented in the XML document using the `integer_literal` element.

The content data shall have 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 signifies that the value is negative.

EXAMPLE - An example attribute_instance element representing an EXPRESS attribute with an integer value follows.

```
<attribute_instance express_attribute_name="count">
<integer_literal>5</integer_literal>
</attribute_instance>
```

7.4.1.3 Number and real data types

A value of EXPRESS number or real data types shall be represented in an XML document using the `real_literal` element.

The content data shall have one of the Numeric Representation forms (NR1, NR2, NR3) specified in ISO 6093.

NOTE - 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.

EXAMPLE - An example `attribute_instance` element representing an EXPRESS attribute with a real value follows.

```
<attribute_instance express_attribute_name="length">
  <real_literal>5.0</real_literal>
</attribute_instance>
```

7.4.1.4 String data type

A value of EXPRESS string data type shall be represented in the XML document using the `string_literal` element.

The content data shall be the characters constituting the value of the string. The content data may contain the control codes RETURN (1D) and NEW LINE (0A) but these control codes shall not be considered as part of the value of the string. No other control codes (00-1F) may appear except those escape sequences that are interpreted as character-code selection sequences as specified in ISO 6429.

EXAMPLE - An example `attribute_instance` element representing an EXPRESS attribute with a string value follows.

```
<attribute_instance express_attribute_name="unit">
<string_literal>kg/m**3</string_literal>
</attribute_instance>
```

7.4.1.5 Binary data type

A value of EXPRESS binary data type shall be represented in the XML document by the `binary_literal element`.

The binary data may be included within the XML document in which case there need not be any value specified for the `external_binary_literal` XML attribute. In this case, the content data shall be an encoding of the binary value. If the binary value is the empty sequence (length = 0), the content data shall be empty. If the binary value is not the empty sequence, it shall be cast as a sequence of octets, wherein the first bit of the binary value shall be the highest order bit of the first octet, with successive bits of the binary value appearing in positions of descending order; the nineth bit of the binary value shall be the highest-order bit of the second octet, and so on; and all empty positions in the last octet shall be occupied by zeros. This sequence of octets shall be encoded into printable character codes in one of the two following ways and the content data shall be the result of the encoding:

— using the IETF RFC 2045 Section 6.8 Base64 Content-Transfer-Encoding (“base64”);

— where each binary octet is encoded as a character tuple, consisting of the two hexadecimal digits {[0-9a-fA-F]} representing the octet code (“hex”).

The binary data may be specified in an external file, and in this case a value shall be specified for the `external_binary_literal` XML attribute, and the corresponding XML entity shall be defined in the XML document.

In the case where the binary data is provided in an external file and is also encoded in the XML document, the `external_binary_literal` XML attribute shall be specified and the PCDATA shall also be provided.

The `notation` XML attribute value shall identify the encoding that was used for the content data.

The `empty_bits` XML attribute value shall be a single digit giving the number of empty positions, if any, in the last octet of the sequence of octets that was encoded into the content data.

NOTE - If the BINARY value is itself an “octet string”, then the length of that octet string can be determined from the encoded value as:

– number of octets = encoding-length / 2, if the notation is “hex”, and

– number of octets = 3*encoding-length / 4, if the notation is “base64” and encoding-length modulo 4 = 0, or

– number of octets = 3*encoding-length / 4 + 1, if the notation is “base64” and encoding-length modulo 4 = 2, or

– number of octets = 3*encoding-length / 4 + 2, if the notation is “base64” and encoding-length modulo 4 = 3.

If the BINARY value is a “bit string”, then the length of that bit string can be determined from the encoded value as: number of bits = 8 * (number of octets) – empty_bits.

7.4.2 Representation of aggregate data types

7.4.2.1 List

Values of the EXPRESS list data type shall be represented in the XML document by a `list_literal` element.

The content of the `list_literal` shall be a sequence of XML elements, each of which is the representation of one member of the list value. The values of the list members shall appear in corresponding order from first to last. If the list value is empty (has no value members), the content of the `list_literal` shall be empty. Each list member value shall be represented according to the data type of that value.

NOTE - If, in a particular EXPRESS entity instance, no value is provided for an OPTIONAL attribute whose data type is the list data type, the `attribute_instance` is omitted from the content of the representation of the EXPRESS entity instance. It is not represented by an empty list.

EXAMPLE - An example `attribute_instance` representing an EXPRESS attribute with a value that is a list of references to EXPRESS entity instances follows.

```
<attribute_instance express_attribute_name="flowering_order">
  <list_literal>
    <entity_instance_ref refid="op2"/>
    <entity_instance_ref refid="op1"/>
  </list_literal>
</attribute_instance>
```

7.4.2.2 Array

Values of the EXPRESS array data type shall be represented in the XML document by an `array_literal` element.

The content of the `array_literal` shall be a sequence of XML elements, each of which is the representation of one member of the array value. The values of the array members shall appear in ascending order of their corresponding index values. Each array member value shall be represented according to its data type.

If an OPTIONAL array member has no value, it shall be represented by the `unset` element.

If the value of an EXPRESS attribute is a multidimensional array, the attribute shall be represented as an array of arrays, nested as deeply as there are dimensions. In constructing such arrays, the inner-most array, the array containing only instances of the element type, shall correspond to the right-most ARRAY specifier in the EXPRESS data type. The ordering of the members within the representation shall be such that all the members of the inner-most array are represented for each member of the next outer array. This order means that the right-most index in each such array shall vary first.

EXAMPLE - An example `attribute_instance` element representing an EXPRESS attribute with a value that is of type `ARRAY[1:3] OF ARRAY[4:4] OF OPTIONAL plant` follows.

```
<attribute_instance express_attribute_name="planting_plan">
<array_literal>
<array_literal>
<entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
</array_literal>
<array_literal>
<entity_instance_ref refid="op2"/>
<entity_instance_ref refid="op2"/>
<entity_instance_ref refid="op2"/>
<entity_instance_ref refid="op2"/>
</array_literal>
<array_literal>
<unset/>
<unset/>
<unset/>
<unset/>
</array_literal>
</array_literal>
</attribute_instance>
```

7.4.2.3 Set

Values of the EXPRESS set data type shall be represented in the XML document by a `set_literal` element.

The content of the `set_literal` shall be a sequence of XML elements, each of which is the representation of one member of the set value. The values of the set members may be presented in any order, but each distinct value shall be presented only once. If the set value is empty (has no value members), the content of the `set_literal` shall be empty. Each set member value shall be represented according to the data type of that value.

NOTE - If, in a particular EXPRESS entity instance, no value is provided for an OPTIONAL attribute whose data type is the set data type, the `attribute_instance` is omitted from the content of the representation of the EXPRESS entity instance. It is not represented by an empty set.

EXAMPLE - An example `attribute_instance` element representing an EXPRESS attribute with a value that is a set follows.


```
<attribute_instance express_attribute_name="has_beds">
<set_literal>
<entity_instance_ref refid="bed1"/>
<entity_instance_ref refid="bed2"/>
<entity_instance_ref refid="bed3"/>
<entity_instance_ref refid="bed4"/>
<entity_instance_ref refid="bed5"/>
</set_literal>
</attribute_instance>
```

7.4.2.4 Bag

Values of the EXPRESS bag data type shall be represented in the XML document by a `bag_literal` element.

The content of the `bag_literal` shall be a sequence of XML elements, each of which is the representation of one member of the bag value. The values of the bag members may be presented in any order. Each distinct value shall occur in the content exactly as often as it occurs in the bag value. If the bag value is empty (has no value members), the content of the `bag_literal` shall be empty. Each bag member value shall be represented according to the data type of that value.

NOTE - If, in a particular EXPRESS entity instance, no value is provided for an OPTIONAL attribute whose data type is a bag data type, the `attribute_instance` is omitted from the content of the representation of the EXPRESS entity instance. It is not represented by an empty `bag_literal`.

EXAMPLE - An example `attribute_instance` element representing an EXPRESS attribute with a value that is a bag follows.

```
<attribute_instance express_attribute_name="nails">
<bag_literal>
<entity_instance_ref refid="nail1"/>
<entity_instance_ref refid="nail2"/>
<entity_instance_ref refid="nail2"/>
<entity_instance_ref refid="nail1"/>
<entity_instance_ref refid="nail4"/>
</bag_literal>
</attribute_instance>
```

7.4.3 Representation of defined data types

Defined data types shall be represented by the `type_literal` element.

The value of the `express_type_name` XML attribute shall be the name of the defined data type the element represents, in the name scope of the EXPRESS schema which governs the schema instance, or if there is no such name then it shall be the name of the data type as defined in the EXPRESS schema where the defined data type is declared.

NOTE -When an alias is specified in the interface specification for the defined data type, the value of the alias will appear as the value for this XML attribute.

The value of the express_schema_name XML attribute shall be the name of the EXPRESS schema to which the data type belongs, and it shall be included only when there is no name for the data type in the name scope of the governing EXPRESS schema.

7.4.3.1 Representation of non-constructed defined data types

For the purposes of the late binding, the term *non-constructed defined data type* is used to mean a type defined by an EXPRESS type declaration in which the immediate underlying type is neither an ENUMERATION nor a SELECT type.

A value of a non-constructed defined type shall be represented in the XML document as a type_literal. The content of the type_literal shall be the representation of the value of the non-constructed defined type, according to the underlying type of the value.

NOTE 1 - When one non-constructed defined data type is defined in terms of another, the content of the type_literal will be another type_literal.

NOTE 2 - When a non-constructed defined data type is defined in terms of a (previously declared) select type, the content of the type_literal will be the representation of whatever type in the select list is actually present in the value.

EXAMPLE - An example attribute_instance element representing an EXPRESS attribute with a value that is of an EXPRESS non-constructed defined type follows.

```
<attribute_instance express_attribute_name="water_volume">
<type_literal express_type_name="positive_integer">
<integer_literal>300</integer_literal>
</type_literal>
</attribute_instance>
```

7.4.3.2 Representation of enumeration types

Values of an EXPRESS ENUMERATION data type shall be represented in the XML document as a type_literal whose content is an enumeration_ref. The content of the enumeration_ref shall be one of the enumerated values in the EXPRESS declaration of the enumeration type, spelled exactly as in the EXPRESS schema. The value of the express_type_name XML attribute shall be the name of the enumeration type as declared in the EXPRESS schema. The value of the enumeration_domain XML attribute, if present, shall consist of the enumeration items.

EXAMPLE - An example attribute_instance representing an EXPRESS attribute with a value that is an EXPRESS enumeration item follows.

```
<attribute_instance express_attribute_name="colour">
<type_literal express_type_name="flower_colour">
<enumeration_ref>red</enumeration_ref>
</type_literal>
</attribute_instance>
```


7.4.3.3 Representation of select types

Values of an EXPRESS select data type shall be represented in the XML document as a `type_literal`. The content of the `type_literal` shall be as specified in this subclause.

An EXPRESS select data type defines a list of data types, called the "select list", whose values are valid instances of the select data type. The late binding treats the EXPRESS named data types in a select list differently depending on which one of the following they are: an EXPRESS entity data type, an enumerated data type, a select data type, or a non-constructed defined data type.

If the value is an instance of an EXPRESS entity data type in the select list, the content shall be an `entity_instance_ref`, `entity_instance` or `entity_instance_as_group`.

If the value is an instance of an enumeration data type in the select list, the content shall be a `type_literal` as specified in 7.4.3.2.

If the value is an instance of a non-constructed defined data type in the select list, the content shall be a `type_literal` as specified in 7.4.3.1.

If the value is an instance of a (nested) select data type in the select list, the content shall be a `type_literal` representing the nested select data type as specified in 7.4.3.3.

NOTE 1 - According to ISO 10303-11:1994, an instance of a subtype of an EXPRESS entity data type is an instance of the EXPRESS entity data type. So "an instance of an EXPRESS entity data type in the select list" includes instances of subtypes of those EXPRESS entity data types.

NOTE 2 - If the EXPRESS entity data types in the select list are not mutually exclusive, then a value of the select data type may instantiate more than one EXPRESS entity data type in the select list.

NOTE 3 - If the value is not an EXPRESS entity instance, it is an instance of exactly one simple defined type or enumeration data type. The value may, however, be a valid instance of several (nested) select data types and thereby instantiate more than one type in the original select list.

EXAMPLE - Given the following EXPRESS declarations

```
ENTITY fish_pond;
fish_colours : SET OF observed_fish_colour;
END_ENTITY;

TYPE observed_fish_colour = SELECT (fish_colour,
multi_coloured_fish_colour);
END_TYPE;

TYPE multi_coloured_fish_colour = SET [2:4] OF fish_colour;
END_TYPE;

TYPE fish_colour = ENUMERATION OF (yellow, black, silver, blue);
END_TYPE;
```

An EXPRESS attribute with a value that is a set of **observered_fish_colors** would be represented as shown below.


```
<attribute_instance express_attribute_name="fish_colours">
  <set_literal>
    <type_literal express_type_name="observed_fish_colour">
      <type_literal express_type_name="fish_colour">
        <enumeration_ref>yellow</enumeration_ref>
      </type_literal>
    </type_literal>
  </set_literal>
</attribute_instance>
```

7.4.4 Representation of EXPRESS attribute values that are EXPRESS entity instance references

When the value of an EXPRESS attribute is itself an EXPRESS entity instance reference, then the EXPRESS entity instance shall be represented in one of the following ways:

— directly nested within the element representing the EXPRESS attribute;

— elsewhere in the XML document;

— external to the XML document.

The elements `entity_instance` or `entity_instance_as_group` shall be used to represent an EXPRESS entity instance directly within the `attribute_instance` according to 7.2.

The element `entity_instance_ref` shall be used to reference an EXPRESS entity instance in the XML document or to reference an `external_refid` element which will in turn point to an EXPRESS entity instance in a remote XML document. The value of the XML attribute `refid` within `entity_instance_ref` shall be equal to the value of the `id` XML attribute for the EXPRESS entity instance representation element if it is in the same XML document or shall be equal to the value of the id XML attribute for an `external_refid` element referring to an element in a remote XML document as specified in 6.5.2. There is no requirement with respect to the order of occurrence of the EXPRESS attribute representation and the EXPRESS entity instance representation element in the XML document.

EXAMPLE - Given the following EXPRESS declarations

```
ENTITY pond;
   maintained_at : temperature_range;
END_ENTITY;

ENTITY temperature_range;
   minimum_value  : REAL;
   maximum_value  : REAL;
END_ENTITY;
```

An example `schema_instance` element would be as shown below.

```
<schema_instance express_schema_name="example">
```



```
<entity_instance express_entity_name="temperature_range" id="tr1">
<attribute_instance express_attribute_name="minimum_value">
<real_literal>-5.0</real_literal>
</attribute_instance>
<attribute_instance express_attribute_name="maximum_value">
<real_literal>25.0</real_literal>
</attribute_instance>
</entity_instance>

<entity_instance express_entity_name="pond" id="po1">
<attribute_instance
express_attribute_name="maintained_at"><entity_instance_ref
refid="tr1"/>
</attribute_instance>
</entity_instance>

</schema_instance>
```

An alternative representation would be to nest the `entity_instance` representing the `temperature_range` as shown below.

```
<schema_instance express_schema_name="example">
<entity_instance express_entity_name="pond" id="po1">
<attribute_instance express_attribute_name="maintained_at">
<entity_instance express_entity_name="temperature_range" id="tr1">
<attribute_instance express_attribute_name="minimum_value">
<real_literal>-5.0</real_literal>
</attribute_instance>
<attribute_instance express_attribute_name="maximum_value">
<real_literal>25.0</real_literal>
</attribute_instance>
</entity_instance>
</attribute_instance>
</entity_instance>
</schema_instance>
```

8 EXPRESS-typed Early Binding (ETEB)

This clause specifies the EXPRESS-typed Early Binding or ETEB. The ETEB is a mapping from an EXPRESS schema, referred to in this clause as the context schema, to an early-bound XML markup declaration set that maintains a very high degree of the typing present in the EXPRESS schema. The resulting markup declaration set will be architecturally compatible with the late bound markup declaration set defined in Annex B.

NOTE - "maintains a high degree of the typing" means that the data types declared in the EXPRESS schema (e.g., entity data types, defined data types) are explicitly maintained to the greatest degree possible in the XML declaration set produced by this specification.

8.1 Fundamental concepts and assumptions

8.1.1 Preconditions

The creation of an ETEB markup declaration set from an EXPRESS schema assumes that the schema meets satisfies the following precondition:

— Syntactically correct EXPRESS schema.

8.1.2 EXPRESS entity referencing

In order to represent a reference to an instance of an EXPRESS entity data type, the ETEB uses an empty XML element that serves as the reference from one element (i.e., an XML representation of an instance of an EXPRESS entity data type) to another. This empty element is named by appending the EXPRESS entity data type identifier with the characters "-ref"; this new element is referred to as an entity data type reference.

NOTE 1 - A hyphen was chosen as a delimiter to prevent possible name conflicts with EXPRESS identifiers. Since hyphens are not legal characters in EXPRESS identifiers, but are legal in XML, there is no possibility that the name created for the entity data type reference element type declaration will clash with an EXPRESS identifier.

NOTE 2 - All references between EXPRESS entity instances within the EXPRESS-typed early binding use the -ref elements rather than use containment to capture such relationships.

EXAMPLE - The following is an example of how a reference to an EXPRESS entity instance is represented in this early-bound XML mapping.

For the EXPRESS declarations:

```
ENTITY greenhouse;
END_ENTITY;
ENTITY garden;
 has_greenhouse : greenhouse;
END_ENTITY;
```

The XML declarations are:

```
<!ELEMENT Greenhouse EMPTY>
<!ATTLIST Greenhouse
    express_entity_name NMTOKEN #FIXED "greenhouse"
    late-bound-element NMTOKEN #FIXED "entity_instance"
    id ID #REQUIRED>

<!ELEMENT Greenhouse-ref EMPTY>
<!ATTLIST Greenhouse-ref
    refid IDREF #REQUIRED
    reftype CDATA #FIXED "refid (Greenhouse | external_refid)"
    late-bound-name CDATA #FIXED "reftype #DEFAULT"
    late-bound-element NMTOKEN #FIXED "entity_instance_ref">

<!ELEMENT Garden (Garden.has_greenhouse)>
```



```
...

<!ELEMENT Garden.has_greenhouse (Greenhouse-ref)>
...
```

8.1.3 Multiple supertypes

The ETEB specifies two ways to represent subtype/supertype graphs depending on whether or not the graph contains an EXPRESS entity data type with multiple supertypes. If no multiple supertypes are present, the representation takes advantage of the nested structure of XML and places subtypes within the supertype "container". This approach enables some of the subtype constraints to be represented in the structure of the markup declaration set.

If multiple supertypes are present, then the markup declarations corresponding to the EXPRESS entity data types in the subtype/supertype graph are flattened out in a single level decomposition. This approach offers maximum flexibility in creating complex instances, but places the burden of enforcing the subtype constraints on processors.

8.1.4 Unmapped EXPRESS concepts

This binding does not address the following EXPRESS features:

— functions;

— global rules;

— procedures;

— local domain rule (WHERE rules);

— inverse attributes;

— redeclared attributes.

In addition, some features of aggregate data types are considered as constraints and are not represented in the markup declarations.

8.1.5 Late binding architectural form

The ETEB is specified as a client of the late binding declarations specified in clauses 6 and 7 as architectural forms. Element type declarations specified in the ETEB are derived from corresponding and more-generic late binding declarations; thus, a client ETEB document can be processed as if it were a late bound document. See 10.3.2.2 for a complete specification of the architectural form processing instructions and their use in a document.

8.2 XML markup declaration set

The following subclauses specify the declaration or feature of a declaration that must be present in the markup declaration set for each feature of the subject EXPRESS schema.

8.2.1 Element naming

ETEB element type names corresponding to EXPRESS identifiers shall consist of lower case characters for letters, with the exception that the first character of the name shall be an upper case character.

If the name of the EXPRESS entity data type or defined data type begins with the characters “xml” regardless of case, these characters in the names shall be replaced by the character string “X-m-l”.

NOTE - XML does not permit element type names that begins with the characters “xml”; this requirement addresses that limitations. EXPRESS attribute names are not modified by this requirement because EXPRESS attribute names are not directly mapped to XML element type names.

8.2.2 Schema independent declarations

The declarations specified in clause 6 shall be included in the XML markup declaration set specified in this clause with the following exceptions; the element type declarations:

— `real_literal`;

— `integer_literal`;

— `logical_literal`;

— `boolean_literal`;

— `string_literal`;

— `binary_literal`; and

— `enumeration_ref`;

shall be removed from the markup declaration set and replaced with the following declarations:

```
<!ELEMENT real  (#PCDATA)>
<!ATTLIST real
    precision CDATA  #IMPLIED
    late-bound-element NMTOKEN #FIXED "real_literal">

<!ELEMENT integer  (#PCDATA)>
<!ATTLIST integer
    late-bound-element NMTOKEN #FIXED "integer_literal">

<!ELEMENT logical (false | unknown | true)>
<!ATTLIST logical
```



```
        late-bound-element NMTOKEN #FIXED "logical_literal">

    <!ELEMENT boolean (false | true)>
    <!ATTLIST boolean
        late-bound-element NMTOKEN #FIXED "boolean_literal">

    <!ELEMENT string  (#PCDATA)>
    <!ATTLIST string  width CDATA  #IMPLIED
        late-bound-element NMTOKEN #FIXED "string_literal">

    <!ELEMENT binary  (#PCDATA)>
    <!ATTLIST binary
        late-bound-element NMTOKEN #FIXED "binary_literal"
        external_binary_literal ENTITY #IMPLIED
        notation  (hex | base64) #IMPLIED
        empty_bits CDATA #REQUIRED >

    <!ELEMENT enumeration-item (#PCDATA)>
    <!ATTLIST enumeration-item
        late-bound-element NMTOKEN #FIXED "enumeration_ref">
```

The markup element type declarations that are not in this list shall be modified as follows:

— add an XML attribute called “late-bound-element” to the attribute list of each element, with a type of NMTOKEN that is #FIXED; the value of the attribute shall be the name of the declared element.

NOTE - The addition of this attribute to all the document description elements used in this binding is necessary to support architectural processing in the absence of validating architectural engines.

8.2.3 EXPRESS schema declaration

The ETEB is applied to a EXPRESS schema that provides the context for the binding, called the *context schema* of the binding. The markup declaration set shall contain an element type declaration that corresponds to the context schema declaration. The name of the element type declaration shall be the EXPRESS schema identifier (i.e., the EXPRESS schema name) represented as specified in 8.2.1 with the string "-schema" appended.

NOTE - The appended string is necessary to disambiguate possible clashes of EXPRESS schema name and EXPRESS entity data type or defined type names.

The content model and attribute list of the element type declaration shall be constructed as specified in the following subclauses.

8.2.3.1 Content model

The content model shall be a repeating (zero or more, "*") choice content particle containing the following element identifiers (i.e., names):

— element types corresponding to EXPRESS entity data types in the schema that are not subtypes;

— synthetic element types (see 8.2.7.2);

— the `external_refid` element type; and

— element types corresponding to EXPRESS entity data types implicitly or explicitly interfaced via schema interface specifications; if the interfaced EXPRESS entity data type is aliased, then the identifier of the alias shall be included in the content model in place of the original identifier. See 8.2.9 for the rules governing interfaced EXPRESS entity data type declarations.

EXAMPLE - For the EXPRESS schema:

```
SCHEMA mr_jones_garden;
ENTITY bed; END_ENTITY;
ENTITY garden; END_ENTITY;
END_SCHEMA;
```

the corresponding XML element type declaration is

```
<!ELEMENT Mr_jones_garden-schema (Bed | Garden | external_refid)*>
```

8.2.3.2 XML attribute list

The markup declaration set shall contain an ATTLIST declaration for the EXPRESS schema element type declaration containing the following XML attributes called:

— "id", with an XML type of ID, and is #REQUIRED;

— "express_schema_name", with an XML type of CDATA, and is #IMPLIED;

— "express_schema_description", with an XML type of CDATA, and is #IMPLIED;

— "express_schema_identifier", with an XML type of CDATA, and is #IMPLIED;

— "late-bound-element", with an XML type of NMTOKEN, is #FIXED, and has a value of "schema_instance";

— "refid", with an XML type of IDREF, and is #IMPLIED;

— "reftype", with an XML type of CDATA, is #FIXED, and has a value of "refid (express_schema | external_refid)".

NOTE - The refid attribute is included to reference the EXPRESS schema declaration if it is included in the XML document. The `express_schema_identifier` attribute may be used to allow for multiple versions of EXPRESS schemas.

The value of the `express_schema_name` XML attribute, if supplied, shall be the name of the context schema.

If present, the value of the `express_schema_identifier` XML attribute shall be an identifier that uniquely identifies the EXPRESS schema. For schemas specified in other parts of ISO 10303, this value should be the information object identifier for that schema as specified in the part.

EXAMPLE - For the **application_context_schema**, the ATTLIST declaration would be:

```
<!ATTLIST Application_context_schema-schema
    id ID #REQUIRED
    express_schema_name CDATA #IMPLIED
    express_schema_description CDATA #IMPLIED
    express_schema_identifier CDATA #IMPLIED
    refid IDREF #IMPLIED
    reftype CDATA #FIXED "refid (express_schema | external_refid)"
    late-bound-element NMTOKEN #FIXED "schema_instance">
```

8.2.4 Defined type declarations

For each EXPRESS defined type declaration, the markup declaration set shall contain an element type declaration corresponding to the EXPRESS defined type. For defined type declarations in the context schema, the name of the element shall be the same as the defined type identifier (i.e., EXPRESS defined type name). In the case of interfaced defined types, the name of the element shall be the same as the defined type identifier modified as specified in 8.2.9. The content model and attribute list of the element type declaration shall be constructed as specified in the following subclauses.

8.2.4.1 Content model

The content model of the element type declaration representing a defined type depends on the underlying type, as specified in the following subclauses.

NOTE - The underlying type of an EXPRESS defined type cannot be an entity data type.

8.2.4.1.1 Simple underlying types

The content model of an element type declaration representing a defined type with a simple data type as its underlying type consists of a single particle. This particle is the identifier of the simple type element type declaration given in 8.2.2 corresponding to the underlying type.

EXAMPLE - For the EXPRESS declaration:

```
TYPE plant_name = STRING;
END_TYPE;
```

the corresponding XML element type declaration is

```
<!ELEMENT Plant_name (string)>
```


8.2.4.1.2 Aggregate underlying types

The content model of an element type declaration representing a defined type with an aggregate data type as its underlying type consists of a single particle. This particle is the identifier of the aggregate data type element type declaration as specified in 8.2.8 corresponding to the underlying type.

EXAMPLE - For the EXPRESS declaration:

```
TYPE names = SET [1:?] OF STRING;
END_TYPE;
```

the corresponding XML element type declaration is

```
<!ELEMENT names (set-of-string)>
```

8.2.4.1.3 Defined type underlying types

The content model of an element type declaration representing a defined type with another defined type as its underlying type consists of a single particle. This particle is the identifier of the EXPRESS defined type element type declaration given in 8.2.4 corresponding to the underlying type.

EXAMPLE - For the EXPRESS declarations:

```
TYPE label = STRING; END_TYPE;
TYPE position = label; END_TYPE;
```

the XML element type declaration for the second defined type is

```
<!ELEMENT Position (Label)>
```

8.2.4.1.4 Enumeration types

The content model of an element type declaration representing a defined type that is an enumeration shall be the particle `enumeration-item`. The XML attribute list representing this kind of defined type declaration defines the enumeration values; see 8.2.4.2.1.

EXAMPLE - For the EXPRESS declarations:

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

the XML element type declaration for the enumeration type is:

```
<!ELEMENT Flower_colour (enumeration-item)>
```


8.2.4.1.5 Select types

The content model of an element type declaration representing a defined type that is a select type shall be a list of particles separated by a choice operator ("|") where each particle corresponds to an item in the select list, as follows:

— if the item is an EXPRESS entity data type, the particle shall be the identifier of the EXPRESS entity data type reference element for the EXPRESS entity data type element, as specified in 8.2.5;

— if the item is a defined type, the particle shall be the identifier of the defined type element, as specified in 8.2.4.

EXAMPLE - For the EXPRESS declarations:

```
TYPE efficiency = SELECT (efficiency_measure, efficiency_range);
END_TYPE;
TYPE efficiency_measure = SELECT (percentage, fractional_value);
END_TYPE;
ENTITY efficiency_range;
 minimum_value : efficiency_measure;
 maximum_value : efficiency_measure;
END_ENTITY;
```

the corresponding XML element type declaration is:

```
<!ELEMENT Efficiency (Efficiency_measure | Efficiency_range-ref)>
```

8.2.4.2 XML attribute list

The markup declaration set shall contain an ATTLIST declaration for the element type declaration representing the defined type containing the following XML attributes called:

— "express_type_name", with an XML type of CDATA, is #FIXED, and has a value of the unqualified defined type name;

— "late-bound-element", with an XML type of NMTOKEN, #FIXED, and has a value of "type_literal".

EXAMPLE - For the EXPRESS declarations:

```
TYPE plant_name = STRING; END_TYPE;
```

the XML element type declaration for the defined type is:

```
<!ELEMENT Plant_name (string)>
<!ATTLIST Plant_name
    express_type_name NMTOKEN #FIXED "plant_name"
    late-bound-element NMTOKEN #FIXED "type_literal">
```


Other XML attributes shall also be present in the ATTLIST for the underlying types as specified in the following subclauses.

8.2.4.2.1 Enumeration types

The ATTLIST for element type declarations corresponding to enumeration types shall contain the following XML attributes called:

— "value-space", which is of XML type of NMTOKEN, is #FIXED, and has a value constructed from the enumeration item names. The value shall be a single string consisting of the enumeration item names separated by spaces such that each item is a distinct token in the string. All letters in enumeration item names shall be lower case characters;

— "late-bound-name" which is of XML type CDATA, is #FIXED, and has a value of "value-space enumeration_domain"

EXAMPLE - For the EXPRESS declarations:

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

the XML element type declaration for the TYPE is

```
<!ELEMENT Flower_colour (enumeration-item)>
<!ATTLIST Flower_colour
    express_type_name NMTOKEN #FIXED "flower_colour"
    late-bound-element NMTOKEN #FIXED "type_literal"
    value-space NMTOKEN #FIXED "red yellow white"
    late-bound-name CDATA #FIXED "value-space enumeration_domain">
```

8.2.5 EXPRESS entity data type declarations

For each EXPRESS entity data type declaration, the markup declaration set shall contain an element type declaration corresponding to the EXPRESS entity data type. The name of the element shall correspond to the EXPRESS entity data type identifier (i.e., the EXPRESS entity data type name) as specified in 8.2.1 or, in the case of interfaced EXPRESS entity data types, as specified in 8.2.9. The content model and attribute list for the element type declaration shall be constructed as specified in the following subclauses.

8.2.5.1 Content model

The content model of the element representing an EXPRESS entity data type shall consist of one particle for each explicit attribute of the EXPRESS entity data type declaration. The name of the particle shall be the name of the element type declaration corresponding to the EXPRESS explicit attribute as specified in 8.2.6. The first particle in the content model shall correspond to the first EXPRESS attribute of the entity data type declaration. The order of the particles in the content model shall be the same as the order of the corresponding attributes in the EXPRESS entity data type declaration; commas shall separate the particles.

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

If the EXPRESS entity data type has no explicit attributes and no subtypes, the content model shall be specified as EMPTY.

If the EXPRESS entity data type has subtypes and the subtype/supertype graph of which it is a part does not have any EXPRESS entity data types with multiple supertypes, then the content model shall contain an additional particle following the attribute particles. This particle shall be constructed as specified in 8.2.7. If the EXPRESS entity data type is not an ABSTRACT supertype, then the zero-or-one XML operator, "?", shall be appended to this particle.

EXAMPLE - For the EXPRESS declarations:

```
ENTITY plant;
 colour : flower_colour;
 latin_name : plant_name;
END_ENTITY;

ENTITY aquatic_plant  SUBTYPE OF (plant); END_ENTITY;
```

the corresponding XML element type declarations are:

```
<!ELEMENT Plant
  (Plant.colour, Plant.latin_name, Plant-subtypes?)>
<!ELEMENT Plant-subtypes (Aquatic_plant)+>
```

8.2.5.2 XML attribute list

The markup declaration set shall contain an ATTLIST declaration for the element type declaration representing an EXPRESS entity data type containing the following XML attributes called:

— "id", with an XML type of ID, and is #REQUIRED;

— "express_entity_name", with an XML type of CDATA, is #FIXED, and has a value equal to the unqualified entity name;

— "late-bound-element", with an XML type of NMTOKEN, is #FIXED; if the EXPRESS entity data type is not a subtype, then this XML attribute has a value of "entity_instance"; if the EXPRESS entity data type is a subtype, then the attribute has a value of "partial_entity_instance".

If the EXPRESS entity data type contains uniqueness rules, then the ATTLIST shall contain one XML attribute for each uniqueness rule declared as follows:

— the name of the XML attribute shall be "unique-1", "unique-2", etc, where the numeric portion of the name corresponds to the sequential order of the rule in the EXPRESS declaration;

— the type is CDATA and is #FIXED;

— the value of the attribute shall be the name of the element corresponding to the EXPRESS attribute that is unique, or the names of elements corresponding to the EXPRESS attributes that are jointly unique; the ordering of the names in the value shall correspond to the ordering of the EXPRESS attribute names in the UNIQUE rule declaration.

NOTE - EXPRESS uniqueness rule label is purposely omitted from the ETEB markup declaration set.

EXAMPLE - For the EXPRESS declaration:

```
ENTITY car;
    serial_no : INTEGER;
    vin : STRING;
 UNIQUE
    UR1:serial_no;
    vin;
END_ENTITY;

ENTITY person;
    owns : car;
    name : STRING;
    ssn : INTEGER;
 UNIQUE
    UR1: name, ssn;
END_ENTITY;
```

the corresponding XML element type declarations are:

```
<!ELEMENT Car (Car.serial_no, Car.vin)>
<!ATTLIST Car
    id ID #REQUIRED
    express_entity_name NMTOKEN #FIXED "car"
    late-bound-element NMTOKEN #FIXED "entity_instance"
    unique-1 CDATA #FIXED "Car.serial_no"
    unique-2 CDATA #FIXED "Car.vin">

<!ELEMENT Person (Person.owns, Person.name, Person.ssn)>
<!ATTLIST Person
    id ID #REQUIRED
    express_entity_name NMTOKEN #FIXED "person"
    late-bound-element NMTOKEN #FIXED "entity_instance"
    unique-1 CDATA #FIXED "Person.name Person.ssn">
```

8.2.5.3 The entity data type reference element

For each EXPRESS entity data type used as an EXPRESS attribute data type, the markup declaration set shall contain an entity data type reference element type declaration. The name of this element shall be the name corresponding to the EXPRESS entity data type identifier (i.e., the EXPRESS entity name) as specified in 8.2.1 with the string "-ref" appended; if the EXPRESS entity data type is an interfaced EXPRESS entity data type, then the name shall correspond to the interfaced EXPRESS entity data type identifier as specified in 8.2.9 with the string "-ref" appended.

The content model of this element type declaration shall be EMPTY.

The markup declaration set shall contain an ATTLIST declaration for the entity data type reference element type declaration containing the following XML attributes called:

— "refid, with an XML type IDREF which is #REQUIRED;

— "reftype", with an XML type of CDATA #FIXED and a value of "refid (*xxx* | {*yyy* | *zzz* | … |} external_refid)" where xxx is the name of the element being pointed to and yyy, zzz, etc., are the names of the elements representing the subtypes of the EXPRESS entity data type *xxx* (if any);

NOTE - Since a reference to an instance of a subtype of an entity is *also* a reference to an instance of each of its supertypes, the "reftype" XML attribute includes the type names of all subtypes of a supertype, regardless of whether they are a direct or indirect subtype.

— "late-bound-name", with an XML type of CDATA #FIXED, and a value of "reftype #DEFAULT";

— "late-bound-element", with an XML type of NMTOKEN #FIXED, and a value of "entity_instance_ref".

EXAMPLE - For the EXPRESS declaration:

```
ENTITY bed; END_ENTITY;
```

the corresponding XML element type declarations are:

```
<!ELEMENT Bed EMPTY>
<!ATTLIST Bed
    id ID #REQUIRED
    express_entity_name NMTOKEN #FIXED "bed"
    late-bound-element NMTOKEN #FIXED "entity_instance">
<!ELEMENT Bed-ref EMPTY>
<!ATTLIST Bed-ref
    refid IDREF #REQUIRED
    reftype CDATA #FIXED "refid (Bed | external_refid)"
    late-bound-name CDATA #FIXED "reftype #DEFAULT"
    late-bound-element NMTOKEN #FIXED "entity_instance_ref">
```

8.2.6 EXPRESS explicit attribute declarations

For each EXPRESS explicit attribute defined in an EXPRESS entity data type declaration, the markup declaration set shall contain an element type declaration corresponding to the EXPRESS attribute. The name of the element shall be a concatenation of the name of the element corresponding to the EXPRESS entity data type, the PERIOD character ".", and the EXPRESS attribute name in lower case, in that order.

NOTE - If the entity data type is interfaced into the context schema, then the name of the element type declaration used to construct the attribute element type name will be the entity data type identifier as modified per 8.2.9.

The content model and attribute list of the element type declaration representing the EXPRESS explicit attribute shall be constructed as specified in the following subclauses.

8.2.6.1 Content model

The content model of the element type declaration representing an EXPRESS explicit attribute shall consist of a single content particle. The name of the particle depends on the data type of the EXPRESS attribute as specified in the following subclauses.

8.2.6.1.1 Simple data types

If the data type of the EXPRESS explicit attribute is a simple type, the name of the particle shall be the element type declaration name corresponding to the simple type as specified 8.2.2.

EXAMPLE - For the EXPRESS declaration:

```
ENTITY real_value_range;
 minimum_value : REAL;
 maximum_value : REAL;
END_ENTITY;
```

the corresponding XML element type declarations are:

```
<!ELEMENT Real_value_range (Real_value_range.minimum_value,
Real_value_range.maximum_value)>
<!ATTLIST Real_value_range
     id ID #REQUIRED
     express_entity_name NMTOKEN #FIXED "real_value_range"
     late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Real_value_range.maximum_value (real)>
<!ATTLIST Real_value_range.maximum_value
      express_attribute_name NMTOKEN #FIXED "maximum_value"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Real_value_range.minimum_value (real)>
<!ATTLIST Real_value_range.minimum_value
      express_attribute_name NMTOKEN #FIXED "minimum_value"
      late-bound-element NMTOKEN #FIXED "attribute_instance">
```

8.2.6.1.2 Defined, enumeration, and select data types

If the data type of the EXPRESS explicit attribute is a defined type, enumeration type, or select type, the name of the particle shall be the name of the element type declaration representing the EXPRESS defined type declaration as specified in 8.2.4.

EXAMPLE - For the EXPRESS Declarations:

```
TYPE plant_name = STRING; END_TYPE;
ENTITY plant;
```



```
 latin_name : plant_name;
END_ENTITY;
```

the corresponding XML element type declarations are:

```
<!ELEMENT Plant_name (string)>
<!ATTLIST Plant_name
    express_type_name NMTOKEN #FIXED "plant_name"
    late-bound-element NMTOKEN #FIXED "type_literal">

<!ELEMENT Plant (Plant.latin_name)>
<!ATTLIST Plant
    id ID #REQUIRED
    express_entity_name NMTOKEN #FIXED "plant"
    late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Plant.latin_name (Plant_name)>
<!ATTLIST Plant.latin_name
      express_attribute_name NMTOKEN #FIXED "latin_name"
      late-bound-element NMTOKEN #FIXED "attribute_instance">
```

8.2.6.1.3 Entity data types

If the data type of the EXPRESS explicit attribute is an EXPRESS entity data type, the name of the particle shall be the name of the EXPRESS entity data type reference element corresponding to the referenced EXPRESS entity data type.

EXAMPLE - For the EXPRESS declaration:

```
ENTITY garden;
 has_greenhouse : greenhouse;
END_ENTITY;

ENTITY greenhouse;
 enforced_temperature_range : temperature_range;
END_ENTITY;
```

the corresponding XML element type declarations are:

```
<!ELEMENT Garden (Garden.has_greenhouse)>
<!ATTLIST Garden
    id ID #REQUIRED
    express_entity_name NMTOKEN #FIXED "garden"
    late-bound-element NMTOKEN #FIXED "entity_instance">


<!ELEMENT Garden.has_greenhouse (Greenhouse-ref)>
<!ATTLIST Garden.has_greenhouse
      express_attribute_name NMTOKEN #FIXED "has_greenhouse"
      late-bound-element NMTOKEN #FIXED "attribute_instance">
```



```
<!ELEMENT Greenhouse (Greenhouse.enforced_temperature_range)>
<!ATTLIST Greenhouse
    id ID #REQUIRED
    express_entity_name NMTOKEN #FIXED "greenhouse"
    late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Greenhouse-ref EMPTY>
<!ATTLIST Greenhouse-ref
    refid IDREF #REQUIRED
    late-bound-name CDATA #FIXED "reftype #DEFAULT"
    reftype CDATA #FIXED "refid (Greenhouse | external_refid)"
    late-bound-element NMTOKEN #FIXED "entity_instance_ref">
```

8.2.6.1.4 Aggregate data types

If the data type of the EXPRESS explicit attribute is an aggregate data type, the name of the particle shall be the name of the element type declaration representing the aggregate data type as specified in 8.2.8

EXAMPLE - For the EXPRESS declaration:

```
ENTITY plant;
 latin_name : plant_name;
 english_names : SET [1 : ?] OF plant_name;
END_ENTITY;
```

the corresponding XML element type declarations are:

```
<!ELEMENT Plant (Plant.latin_name, Plant.english_names)>
<!ATTLIST Plant
    id ID #REQUIRED
    express_entity_name NMTOKEN #FIXED "plant"
    late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Plant.latin_name (Plant_name)>
<!ATTLIST Plant.latin_name
      express_attribute_name NMTOKEN #FIXED "latin_name"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Plant.english_names (set-of-Plant_name)>
<!ATTLIST Plant.english_names
      express_attribute_name NMTOKEN #FIXED "english_names"
      late-bound-element NMTOKEN #FIXED "attribute_instance">
```

8.2.6.2 XML attribute list

The markup declaration set shall contain an ATTLIST declaration for the EXPRESS attribute element type declaration containing the following XML attributes called:

— "express_attribute_name", with an XML type of CDATA, is #FIXED, and have a value equal to the unqualified name of the EXPRESS explicit attribute;

— "late-bound-element", with an XML type of NMTOKEN, is #FIXED, and has a value of "attribute_instance".

8.2.7 Subtypes and Supertypes

The structure of the declarations in the markup declaration set that represent EXPRESS subtype/supertype graphs depend on whether or not the subtype/supertype graph contains EXPRESS entity data types with multiple supertypes. See 8.1.3 for an explanation of the bifurcation.

8.2.7.1 Subtypes

If an EXPRESS entity data type is a supertype and the subtype/supertype graph of which it is a part contains no EXPRESS entity data types with multiple supertypes, the markup declaration set shall have the following characteristics:

— it shall contain an element type declaration representing the collection of subtypes of the EXPRESS entity data type and the name of this element shall be the name of the element corresponding to the supertype EXPRESS entity data type, with the string "-subtypes" appended;

— the content model of the "-subtypes" element type declaration shall consist of the names of the elements corresponding to the immediate subtypes of the EXPRESS supertype entity data type separated by the choice operator, "|". The choice particle shall have a cardinality operator of one-or-more, "+";

— the last particle in the content model of the element type declaration corresponding to the EXPRESS supertype entity data type (as specified in 8.2.5.1) shall be the name of the “-subtypes” element type declaration.

EXAMPLE - For the EXPRESS declaration:

```
ENTITY vehicle;
    vin : STRING;
END_ENTITY;

ENTITY car SUBTYPE OF (vehicle);
    make  : STRING;
    car_model : STRING;
END_ENTITY;

ENTITY truck SUBTYPE OF (vehicle);
    capacity : INTEGER;
END_ENTITY;
```

the corresponding XML element type declarations are:

```
<!ELEMENT Vehicle (Vehicle.vin, Vehicle-subtypes?)>
<!ATTLIST Vehicle
    id ID #REQUIRED
```



```
    express_entity_name NMTOKEN #FIXED "vehicle"
    late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Car (Car.make, Car.car_model)>
<!ATTLIST Car
    id ID #IMPLIED
    express_entity_name NMTOKEN #FIXED "car"
    late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Truck (Truck.capacity)>
<!ATTLIST Truck
    id ID #IMPLIED
    express_entity_name NMTOKEN #FIXED "truck"
    late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Vehicle-subtypes ((Car | Truck)+)>
```

NOTE 1 - The *xxx*-subtypes element is a structural wrapper that does not affect and is not affected by the architectural mapping from/to the late binding. Therefore, there is no late bound element that corresponds to this declaration.

NOTE 2 - There is no ATTLIST declaration for the *xxx*-subtypes element.

NOTE 3 - The enforcement of subtype constraints is part of the constraint-checking behaviour of processors and is not addressed in the markup declaration set.

8.2.7.2 Multiple supertypes

If an EXPRESS entity data type is a supertype and the subtype/supertype graph contains EXPRESS entity data types with multiple supertypes, then markup declaration set shall contain a single element type declaration that corresponds to the *entire subtype/supertype graph*. This element type is called a *synthetic element type* with respect to the EXPRESS schema because it does not correspond to any single EXPRESS entity data type declaration.

NOTE 1 - The requirements specified in 8.2.7.1 do not apply to the EXPRESS entity data types in this subtype/supertype graph. The requirements specified in 8.2.5 and 8.2.6 do apply.

NOTE 2 - The requirements specified in this subclause result in a single element type declaration that represents all complex entity data types specified by the entire, *complete* subtype/supertype graph. Each individual EXPRESS entity data type in the graph is still declared as an element, but the only place that the subtype instances of this element can appear is within the context of this large "synthetic" type. The supertypes can be instantiated on their own in the EXPRESS schema element.

The name of the element type declaration representing the complete subtype/supertype graph shall be constructed from the names of the independently instantiable EXPRESS entity data types in the subtype/supertype graph (i.e., those EXPRESS entity data types that are not subtypes). The names shall be concatenated to form a single token where the names occur alphabetically in the string and

use "CamelCase"1 for differentiation of the name components. This name shall be prefixed with the characters “syn-”.

NOTE 3 – The prefix “syn-” disambiguates element type names in situations where a diamond-shape inheritance tree occurs. In these situations, both the synthetic element type declaration and the root entity element type declaration would have the same name if the prefix were not applied.

The content model of the element type declaration representing the complete subtype/supertype graph shall consist of the names of EXPRESS entity data types that are part of the subtype/supertype graph separated by the choice operator, "|". The content model shall have a cardinality operator of one-or-more, "+".

The markup declaration set shall contain an XML attribute list declaration for the element type declaration with the following XML attributes called:

— "id", with a type of ID, which is #REQUIRED.

— "late-bound-element", with a type of NMTOKEN, which is #FIXED, and has a value of "entity_instance_as_group".

EXAMPLE - For the EXPRESS declaration:

```
ENTITY pond
 ABSTRACT SUPERTYPE OF (fish_pond ANDOR ornamental_pond);
 water_volume: positive_integer;
END_ENTITY;

ENTITY livestock_container  ABSTRACT SUPERTYPE OF (fish_pond);
 livestock_count : OPTIONAL INTEGER;
END_ENTITY;

ENTITY fish_pond  SUBTYPE OF (pond, livestock_container);
END_ENTITY;

ENTITY ornamental_pond  SUBTYPE OF (pond);
END_ENTITY;
```

The complex entity data types of this graph are pond&ornamental_pond, pond&ornamental_pond&fish_pond&livestock_container, and pond&fish_pond&livestock_container.

The XML element type declarations corresponding to the EXPRESS above are:

```
<!ELEMENT Pond (Pond.water_volume)>
<!ATTLIST Pond
   id ID #IMPLIED
   express_entity_name NMTOKEN #FIXED "pond"
   late-bound-element NMTOKEN #FIXED "partial_entity_instance">
```

1 "CamelCase" eliminates spaces between words that comprise the term and capitalizes the first character of each word.


```
<!ELEMENT Livestock_container (Livestock_container.livestock_count?)>
<!ATTLIST Livestock_container
   id ID #IMPLIED
   express_entity_name NMTOKEN #FIXED "livestock_container"
   late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Fish_pond EMPTY >
<!ATTLIST Fish_pond
   id ID #IMPLIED
   express_entity_name NMTOKEN #FIXED "fish_pond"
   late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Ornamental_pond EMPTY>
<!ATTLIST Ornamental_pond
   id ID #IMPLIED
   express_entity_name NMTOKEN #FIXED "ornamental_pond"
   late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT syn-Livestock_containerPond
 (Fish_pond | Livestock_container | Ornamental_pond | Pond)+>
<!ATTLIST syn-Livestock_containerPond
  id ID #REQUIRED
  late-bound-element NMTOKEN #FIXED "entity_instance_as_group">
```

8.2.8 Aggregate data types

For each aggregate data type, the markup declaration set shall contain an element type declaration representing the aggregate data type. The name of the element shall be created by concatenating the following three strings:

— the name of the kind of the aggregate data type: either “set”, “bag”, “list” or “array”;

— the string "-of-";

— the name of the base type of the aggregate data type: either the name of the named type, the keyword of the simple type or if the base type is another aggregate data type, then this subclause, shall be "invoked again" and the name of the base type is the name of the resulting element type declaration.

NOTE - The recursive behaviour of nested aggregates will result in a separate element type declaration for each level of nesting.

8.2.8.1 Content model

The content model of the element type declaration representing a set, list, or bag shall contain a single particle. The name of this particle depends on the base type of the aggregate. If the base type is:

— a simple type, the name shall be that of the corresponding simple type element type declaration specified in 8.2.4;

— a defined, enumeration, or select type, the name shall be that of the corresponding element type declaration specified in 8.2.4;

— an EXPRESS entity data type, the name shall be that of the corresponding entity data type reference element type declaration specified in 8.2.5.3;

— an aggregate data type, the name shall be that of the corresponding aggregate element type declaration specified in 8.2.8.

If the aggregate type is a set, bag, or list, then the particle is appended with a zero-or-more cardinality operator, "*".

The content model of the element type declaration representing an array shall contain two particles separated by a choice operator, "|". The name of the first particle depends on the base type of the aggregate and shall be named in the same manner as that of a set, list, or bag. The second particle shall be `unset`. The choice particle shall be appended with a one-or-more cardinality operator, "+".

NOTE - The inclusion of the `unset` element in the content model of array element type declarations is necessary to account for OPTIONAL members in the array. The enforcement of mandatory array members, therefore, is left to processors (as is ensuring an appropriate number of members in an array) since the element type declaration assumes optionality.

8.2.8.2 Attribute list

The markup declaration set shall contain an ATTLIST declaration for the aggregate type element type declaration. The ATTLIST declaration shall contain an XML attribute named "late-bound-element", with an XML type of NMTOKEN, which is #FIXED. The value of this XML attribute depends upon the kind of aggregate type. For arrays the value shall be "array_literal", for bags the value shall be "bag_literal", for lists the value shall be "list_literal" and for sets the value shall be "set_literal".

EXAMPLE 1 - For the EXPRESS declaration:

```
ENTITY garden;
 has_beds : SET [5 : 5] OF bed;
END_ENTITY;

ENTITY bed; END_ENTITY;
```

the corresponding XML element type declarations are:

```
<!ELEMENT Garden (Garden.has_beds)>
<!ATTLIST Garden
    id ID #REQUIRED
    express_entity_name NMTOKEN #FIXED "garden"
    late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Garden.has_beds (set-of-Bed)>
<!ATTLIST Garden.has_beds
      express_attribute_name NMTOKEN #FIXED "has_beds"
      late-bound-element NMTOKEN #FIXED "attribute_instance">
```



```
<!ELEMENT Bed EMPTY>
<!ATTLIST Bed
      id ID #REQUIRED
      express_entity_name NMTOKEN #FIXED "bed"
      late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Bed-ref EMPTY>
<!ATTLIST Bed-ref
      refid IDREF #REQUIRED
      late-bound-element NMTOKEN #FIXED "entity_instance_ref"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      reftype CDATA #FIXED "refid (Bed | external_refid)">

<!ELEMENT set-of-Bed (Bed-ref*)>
<!ATTLIST set-of-Bed
      late-bound-element NMTOKEN #FIXED "set_literal">
```

EXAMPLE 2 - For the EXPRESS declaration:

```
TYPE matrix = ARRAY [1:3] OF ARRAY[1:4] OF REAL; END_TYPE;
```

the corresponding XML element type declaration is

```
<!ELEMENT Matrix (array-of-array-of-real)>
<!ATTLIST Matrix
    express_entity_name NMTOKEN #FIXED "matrix"
    late-bound-element NMTOKEN #FIXED "type_literal">

<!ELEMENT array-of-array-of-real ((array-of-real | unset)+)>
<!ATTLIST array-of-array-of-real
    late-bound-element NMTOKEN #FIXED "array_literal">

<!ELEMENT array-of-real ((real | unset)+)>
<!ATTLIST array-of-real
    late-bound-element NMTOKEN #FIXED "array_literal">
```

8.2.9 Interface specifications

The EXPRESS USE FROM and REFERENCE FROM declarations either explicitly or implicitly interface EXPRESS entity data types and defined types declared in other schemas (the *interfaced schemas*) into the schema in which the USE FROM/REFERENCE FROM declaration appears (the *interfacing schema*). The markup declaration set shall contain element type declarations corresponding to the interfaced EXPRESS entity data types and defined types. The element type declarations shall be named and structured according to the rules for EXPRESS entity data types and defined types specified in 8.2.4 through 8.2.8 modified according to the rules specified in this subclause.

The name of the element type declaration corresponding to each interfaced EXPRESS entity data type or defined type shall adhere to one of the following:

— If the EXPRESS entity data type or defined type is explicitly interfaced and not aliased, the name of the corresponding element type declaration shall correspond to the EXPRESS entity data type or defined type name;

— If the EXPRESS entity data type or defined type is explicitly interfaced and aliased, the name of the element type declaration shall correspond to the alias name;

— If the EXPRESS entity data type or defined type is implicitly interfaced and the name of the EXPRESS entity data type or defined type does not clash with names in the interfacing schema or names interfaced from other schemas, the name of the corresponding element type declaration shall correspond to the EXPRESS entity data type or defined type name.

— If the EXPRESS entity data type or defined type is implicitly interfaced and the name of the EXPRESS entity data type or defined type clashes with a name in the interfacing schema or name interfaced from other schemas, the name of the corresponding element type declaration shall be a string formed by concatenating the following strings in order: the name corresponding to the source EXPRESS schema, the string "-schema.", and the name corresponding to the interfaced EXPRESS entity data type or defined type.

The name of the element type declaration corresponding to an interfaced EXPRESS entity data type that is not a subtype shall appear as a particle in the content model of the schema element.

If an interfaced EXPRESS entity data type or defined type is aliased, then the alias shall be used in place of the original name to meet all the markup declaration set requirements for EXPRESS named types as specified in 8.2.4 through 8.2.8.

The XML attribute list declarations for the element type declarations representing the EXPRESS entity data type or defined type shall have the following XML attributes called:

— "express_schema_name", with an XML type of NMTOKEN, is #FIXED, and has a value equal to the source schema name;

— "import-method", with an XML type of CDATA, is #FIXED and has a value of "explicit" if the EXPRESS entity data type is explicitly interfaced or "implicit" is the EXPRESS entity data type is implicitly interfaced;

— If the EXPRESS entity data type or defined type is aliased: "aliased-from", with an XML type of NMTOKEN, is #FIXED, and have a value equal to the name of the EXPRESS entity data type or defined type declaration in the source EXPRESS schema.

NOTE - Explicitly interfaced EXPRESS entity data types and defined types are (1) the EXPRESS entity data types or defined types named in a USE FROM/REFERENCE FROM statement; or (2) all the EXRPESS entity data types and defined types in the interfaced schema if no EXPRESS entity data types or defined types are explicitly identified in the USE FROM/REFERENCE FROM statement.

Interfaced constants shall follow the requirements for constants specified in 8.2.10 in addition the requirements specified in this subclause.

EXAMPLE - For the EXPRESS declaration:


```
SCHEMA mr_jones_garden;
REFERENCE FROM mr_smiths_garden (garden AS mr_smiths_garden);
USE FROM mr_smiths_garden (plant AS mr_smiths_plant);
END_SCHEMA;
```

where

```
SCHEMA mr_smiths_garden;

ENTITY bed;
 description : STRING;
END_ENTITY;

ENTITY flower_plant SUBTYPE OF (plant);
 flower_colour : STRING;
END_ENTITY;

ENTITY garden;
 has_bed : bed;
END_ENTITY;

ENTITY plant SUPERTYPE OF (flower_plant ANDOR vegetable_plant);
 name : STRING;
END_ENTITY;

ENTITY vegetable_plant SUBTYPE OF (plant);
 vegetable_type : STRING;
END_ENTITY;

END_SCHEMA;
```

the corresponding XML element type declarations are

```
<!ELEMENT Mr_smiths_garden (Mr_smiths_garden.has_bed)>
<!ATTLIST Mr_smiths_garden
      id ID #REQUIRED
      express_entity_name NMTOKEN #FIXED "garden"
      express_schema_name NMTOKEN #FIXED "mr_smiths_garden"
      late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Mr_smiths_garden-ref EMPTY>
<!ATTLIST Mr_smiths_garden-ref
      refid IDREF #REQUIRED
      late-bound-element NMTOKEN #FIXED "entity_instance_ref"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      reftype CDATA #FIXED "refid (Mr_smiths_garden |
external_refid)">

<!ELEMENT Mr_smiths_garden.has_bed
   (Mr_smiths_garden-schema.bed-ref)>
<!ATTLIST Mr_smiths_garden.has_bed
      express_attribute_name NMTOKEN #FIXED "has_bed"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Mr_smiths_garden-schema.bed
```



```
 (Mr_smiths_garden-schema.bed.description)>
<!ATTLIST Mr_smiths_garden-schema.bed
      id ID #REQUIRED
      express_entity_name NMTOKEN #FIXED "bed"
      express_schema_name NMTOKEN #FIXED "mr_smiths_garden"
      late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Mr_smiths_garden-schema.bed-ref EMPTY>
<!ATTLIST Mr_smiths_garden-schema.bed-ref
      refid IDREF #REQUIRED
      late-bound-element NMTOKEN #FIXED "entity_instance_ref"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      reftype CDATA #FIXED "refid (Mr_smiths_garden-schema.bed |
external_refid)">

<!ELEMENT Mr_smiths_garden-schema.bed.description (string)>
<!ATTLIST Mr_smiths_garden-schema.bed.description
      express_attribute_name NMTOKEN #FIXED "description"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Mr_smiths_plant (Mr_smiths_plant.name)>
<!ATTLIST Mr_smiths_plant
      id ID #REQUIRED
      express_entity_name NMTOKEN #FIXED "plant"
      express_schema_name NMTOKEN #FIXED "mr_smiths_garden"
      late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Mr_smiths_plant.name (string)>
<!ATTLIST Mr_smiths_plant.name
      express_attribute_name NMTOKEN #FIXED "name"
      late-bound-element NMTOKEN #FIXED "attribute_instance">
```

8.2.10 Constants

For each constant declaration in an EXPRESS schema, the markup declaration set shall contain an XML attribute called "express_constant_name". The attribute shall be declared in the ATTLIST of the element type declaration corresponding to the base type of the constant. The XML type of the XML attribute shall be CDATA, and shall be #IMPLIED. The XML attribute has no default or specified value in the markup declaration set. Rather, the value of the XML attribute shall be the constant name and appear in an instance of the element in a document.

NOTE - Constants in an EXPRESS schema are pre-declared members of a data population governed by the schema. The manifestation of a constant in a document is an element that is almost identical to other elements of the same element type. The only difference is the existence of an additional XML attribute that provides the name of the constant, thereby designating the element as representing a constant.

The existence of a constant cannot be enforced by the markup declaration set. Rather, the requirement is enforced by processors that validate the document against the EXPRESS schema.

EXAMPLE - For the EXPRESS declarations:

```
CONSTANT
```



```
    beverly_hills : INTEGER := 90210;
    unit_vector   : LIST [1:3] OF REAL := [1.0, 1.0, 1.0];
    origin        : point := point(0.0, 0.0, 0.0);
END_CONSTANT;
```

the corresponding XML element type declarations are:

```
<!ELEMENT integer  (#PCDATA)>
<!ATTLIST integer
    late-bound-element NMTOKEN #FIXED "integer_literal"
    express_constant_name NMTOKEN #IMPLIED>

<!ELEMENT list-of-real (real+)>
<!ATTLIST list-of-real
    late-bound-element NMTOKEN #FIXED "list_literal"
    express_constant_name NMTOKEN #IMPLIED>

<!ELEMENT Point (Point.x, Point.y, Point.z)>
<!ATTLIST Point
    id ID #REQUIRED
    express_entity_name NMTOKEN #FIXED "point"
    late-bound-element NMTOKEN #FIXED "entity_instance"
    express_constant_name CDATA #IMPLIED>
```

the corresponding XML element instances are:

```
<integer express_constant_name='beverly_hills'>90210</integer>

<list-of-real express_constant_name='unit_vector'>
    <real>1.0</real>
    <real>1.0</real>
    <real>1.0</real>
</list-of-real>

<Point id='p1' express_constant_name='origin'>
    <Point.x><real>0.0</real></Point.x>
    <Point.y><real>0.0</real></Point.y>
    <Point.z><real>0.0</real></Point.z>
</Point>
```

The `integer` and `list-of-real` element instances presented in this example will appear in an XML document within the context of a higher-level element's use, i.e., embedded as an attribute value, for example. They will not appear as immediate child elements of the schema element.

8.2.11 Derived attributes

This subclause shall not be applied in the case that an EXPRESS attribute is redeclared to be an EXPRESS derived attribute. In all other cases, for each EXPRESS derived attribute in an EXPRESS entity data type , the markup declaration set shall contain an element type declaration corresponding to the EXPRESS derived attribute. The element type declaration shall be constructed in the same manner as an explicit EXPRESS attribute (see 8.2.6) with the following exceptions, additions, and modifications:

— In the element type declaration representing the EXPRESS entity data type, the content particle(s) that correspond to element type declaration(s) representing the EXPRESS derived attributes shall appear following any EXPRESS explicit attribute content particles and preceding the *xxx*-subtype particle if present; the particles shall appear in the order in which the EXPRESS derived attributes appear in the EXPRESS entity data type declaration;

— The content particle(s) shall be appended with the optional cardinality indicator, “?”;

— Add the XML attribute called "derived" to the ATTLIST for the element type declaration representing the EXPRESS derived attribute, with an XML type of CDATA, #FIXED, and a value equal to "true"

Specification of the use, validity of or value of an ETEB element representing an EXPRESS derived attribute is outside the scope of this part of ISO 10303.

8.2.12 Declaration set optimizations

The markup declaration set may be optimized by any of the following modifications:

— Remove unused *xxx*-ref element type declarations;

— Remove the XML ID-typed attribute for elements representing EXPRESS entity data types that are not referenceable;

— Remove elements representing unused simple types;

— Replace all element type names, or parts of element type names, corresponding to EXPRESS identifiers with short names.

8.3 Data encoding

8.3.1 Representation of EXPRESS entity instances

For each EXPRESS simple entity instance in the schema instance, the element representing the schema instance (see 8.2.3) shall contain an element of the type corresponding to the EXPRESS entity data type of the instance as specified in 8.2.5. The element shall have a local identifier that is the value of the `id` attribute of the element.

For each EXPRESS complex entity instance, if the subtype/supertype graph does not contain an EXPRESS entity data type with multiple supertypes, the schema element shall contain an element governed by the declarations derived from the EXPRESS entity data types in the graph according to 8.2.5 and 8.2.7.1. If the subtype/supertype graph does contain multiple supertypes, the schema element shall contain elements governed by the declarations derived from the EXPRESS entity data types in the graph according to 8.2.5 and 8.2.7.2

8.3.2 Representation of EXPRESS explicit attributes

For each EXPRESS attribute value of an EXPRESS entity instance corresponding to an EXPRESS explicit attribute of the EXPRESS entity data type, the element representing the EXPRESS entity instance shall contain an element of the type corresponding to the EXPRESS attribute, as specified in 8.2.6. The content of the element representing the EXPRESS attribute shall depend on the data type of the EXPRESS attribute, as specified in 8.3.4.

If the EXPRESS attribute is declared as OPTIONAL and there is no value in the EXPRESS entity instance, then the element corresponding to the EXPRESS attribute shall not appear in the representation of the EXPRESS entity instance.

8.3.3 Representation of DERIVEd attributes

If the EXPRESS entity data type governing an EXPRESS entity instance has EXPRESS derived attributes, an element corresponding to the EXPRESS derived attribute value may be inserted into the element corresponding to the EXPRESS entity instance as specified in 8.2.11

8.3.4 Representation of EXPRESS attribute values

8.3.4.1 BOOLEAN and LOGICAL data types

An EXPRESS attribute value of an EXPRESS entity instance that is of the BOOLEAN data type shall be represented using a `boolean` element as specified in 8.2.2. An EXPRESS attribute value that is of the LOGICAL data type shall be represented using a `logical` element as specified in 8.2.2.

The value TRUE shall be represented using the element `true`; the value FALSE shall be represented using the element `false`; and the value UNKNOWN shall be represented using the element `unknown`.

8.3.4.2 INTEGER data type

An EXPRESS attribute value of an EXPRESS entity instance that is of the INTEGER data type shall be represented using an `integer` element as specified in 8.2.2.

The content of the `integer` element shall represent the value in the form specified as Numeric Representation 1 (NR1) in ISO 6093: 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.

EXAMPLE - The following are valid content for an element that represents an EXPRESS attribute of data type INTEGER:

```
-1
0
12678967543233
```


8.3.4.3 REAL and NUMBER data types

An EXPRESS attribute value of an EXPRESS entity instance that is of the REAL or NUMBER data type shall be represented using a `real` element as specified in 8.2.2.

The content of the real element shall represent the value in one of the Numeric Representation forms (NR1, NR2, NR3) specified in ISO 6093.

NOTE - 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.

EXAMPLE - The following are valid content for an element that represents an EXPRESS REAL or EXPRESS NUMBER:

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

8.3.4.4 STRING data type

An EXPRESS attribute value of an EXPRESS entity instance that is of the STRING data type shall be represented using a `string` element as specified in 8.2.2.

The content the string element shall be comprised of exactly the characters constituting the value of the string, except that the control codes RETURN (1D) and NEW LINE (0A) may occur anywhere in the content data and shall have no significance. No other control codes (00-1F) shall appear in the content except those escape sequences that are interpreted as character-code selection sequences as specified in ISO 6429.

8.3.4.5 BINARY data type

An EXPRESS attribute value of an EXPRESS entity instance that is of the BINARY data type shall be represented using a `binary` element as specified in 8.2.2. The binary data may be included within the XML document as content data. If the binary value is the empty sequence (length = 0), the content data shall be empty. Otherwise, the binary value shall be encoded as either “hex” or “base64” in the manner specified in 7.4.1.5 and the content data shall be the result of that encoding.

The binary data may be specified in an external file, and in this case a value shall be specified for the `express_binary_literal` XML attribute, and the corresponding XML entity shall be defined in the document.

In the case where the binary data is provided in an external file and is also encoded in the document, the `express_binary_literal` XML attribute shall be specified and the PCDATA shall also be provided.

The `notation` XML attribute value shall identify the encoding that was used for the content data.

The `empty_bits` XML attribute value shall be a single digit giving the number of empty positions, if any, in the last octet of the sequence of octets that was encoded into the content data.

NOTE - If the BINARY value is itself an “octet string”, then the length of that octet string can be determined from the encoded value as:

– number of octets = encoding-length / 2, if the notation is “hex”, and

– number of octets = 3*encoding-length / 4, if the notation is “base64” and encoding-length modulo 4 = 0, or

– number of octets = 3*encoding-length / 4 + 1, if the notation is “base64” and encoding-length modulo 4 = 2, or

– number of octets = 3*encoding-length / 4 + 2, if the notation is “base64” and encoding-length modulo 4 = 3.

– If the BINARY value is a “bit string”, then the length of that bit string can be determined from the encoded value as: number of bits = 8 * (number of octets) – empty_bits.

8.3.4.6 EXPRESS entity data type

An EXPRESS attribute value of a EXPRESS entity instance that is a reference to an EXPRESS entity instance shall be represented using an entity data type reference element as specified in 8.2.5.3. The `refid` XML attribute of this element shall have a value, and the value shall be the value of the `id` XML attribute of the element representing the EXPRESS entity instance in the same XML document or of an `external_refid` that refers to an element representing the EXPRESS entity instance in a remote XML document.

8.3.4.7 Aggregate data types

An EXPRESS attribute value of an EXPRESS entity instance that is of an aggregate data type shall be represented by an element governed by the declaration specified in 8.2.8.

The content of the aggregate data type element shall be a sequence of elements, each of which is the representation of one member of the aggregate type. The sequence of elements shall be presented in order from first element to the last. If the EXPRESS aggregate instance has no members, the element shall have no content. Each member of the aggregate shall be represented according to the data type of that value as specified in 8.3.

OPTIONAL values in an ARRAY aggregate that are omitted shall be represented by a reference to the `unset` element.

8.3.4.8 ENUMERATION data type

An EXPRESS attribute value that is of an ENUMERATION data type shall be represented by the corresponding enumeration type element specified in 8.2.4. The content of this element shall be an

`enumeration-item` element. The content of the `enumeration-item` element shall be the enumeration id corresponding to the value, as specified in 8.2.4.

8.3.4.9 Representation of select types

An EXPRESS attribute value of an EXPRESS entity instance that is a SELECT data type shall be represented as specified in 8.2.4. The content of this element shall be element corresponding to the data type of the attribute value as specified in 8.2.4.1.5.

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.

Annex A
(normative)

Information object registration

To provide for unambiguous identification of an information object in an open system, the object identifier

{ iso standard 10303 part(28) version (1)}

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

A.1 Information identification

To provide for unambiguous identification of the late bound architectural Document Type Declaration an open information system, the object identifier

{ iso standard 10303 part(28) version(1) object(1) 10303_28_Architectural_DTD(1) }

is assigned to the DTD as defined in Annex D. The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

Annex B
(normative)

The XML markup declaration set for late bound data

B.1 Introduction

This annex specifies a markup declaration set used to represent data for which there is a specification in the EXPRESS language (ISO 10303-11). These declarations provide for the representation of data in a late bound form. This markup declaration set can be used to represent data described by any EXPRESS schema.

B.2 Common elements

The following elements are common to the Late Binding markup declaration set and the markup declaration set for the EXPRESS language specified in Annex C.

```
<!NOTATION base64 PUBLIC "base-64 encoded data">
<!NOTATION hex PUBLIC "hexadecimal digits">
<!NOTATION uri SYSTEM "http://www.w3.org/REC/uri.xml">

<!ELEMENT binary_literal
 (#PCDATA)>
<!ATTLIST binary_literal
      express-production CDATA #FIXED "binary_literal"
      express_constant_name NMTOKEN #IMPLIED
      external_binary_literal ENTITY #IMPLIED
      notation  (hex | base64) #IMPLIED
      empty_bits CDATA #REQUIRED>

<!ELEMENT boolean_literal
 (
  (embedded_remark | tail_remark)?,
  (false | true))>
<!ATTLIST boolean_literal
      express-production CDATA #FIXED "#NONE"
      express_constant_name NMTOKEN #IMPLIED >

<!ELEMENT embedded_remark
 (#PCDATA | embedded_remark)*>
<!ATTLIST embedded_remark
      express-production CDATA #FIXED "embedded_remark">

<!ELEMENT enumeration_ref
 (#PCDATA)>
<!ATTLIST enumeration_ref
      express-production CDATA #FIXED "enumeration_ref"
      refid IDREF #IMPLIED
      reftype CDATA #FIXED "refid (enumeration_id)">
```



```
<!ELEMENT false EMPTY>
<!ATTLIST false
      express-production CDATA #FIXED "FALSE">

<!ELEMENT integer_literal
 (#PCDATA)>
<!ATTLIST integer_literal
      express-production CDATA #FIXED "integer_literal"
      express_constant_name NMTOKEN #IMPLIED >

<!ELEMENT logical_literal
 (
  (embedded_remark | tail_remark)?,
  (false | true | unknown))>
<!ATTLIST logical_literal
      express-production CDATA #FIXED "logical_literal"
      express_constant_name NMTOKEN #IMPLIED >

<!ELEMENT real_literal
 (#PCDATA)>
<!ATTLIST real_literal
      express-production CDATA #FIXED "real_literal"
      express_constant_name NMTOKEN #IMPLIED >

<!ELEMENT string_literal
 (#PCDATA)>
<!ATTLIST string_literal
      express-production CDATA #FIXED "string_literal"
      express_constant_name NMTOKEN #IMPLIED >

<!ELEMENT tail_remark
 (#PCDATA)*>
<!ATTLIST tail_remark
      express-production CDATA #FIXED "tail_remark">

<!ELEMENT true EMPTY>
<!ATTLIST true
      express-production CDATA #FIXED "TRUE">

<!ELEMENT unknown EMPTY>
<!ATTLIST unknown
      express-production CDATA #FIXED "UNKNOWN">

<!ELEMENT unset EMPTY>
<!ATTLIST unset
      express-production CDATA #FIXED "?">
```

B.3 Late binding markup declarations

The following are the late binding-specific markup declarations.

```
<!ENTITY % schema_instance "schema_instance">
<!ELEMENT array_literal
```



```
 (binary_literal | boolean_literal | integer_literal | logical_literal |
real_literal | string_literal | bag_literal | list_literal | set_literal |
array_literal | type_literal | entity_instance | entity_instance_as_group |
entity_instance_ref | unset)*>
<!ATTLIST array_literal
      express_constant_name NMTOKEN #IMPLIED >

<!ELEMENT attribute_instance
 (binary_literal | boolean_literal | integer_literal | logical_literal |
real_literal | string_literal | bag_literal | list_literal | set_literal |
array_literal | type_literal | entity_instance | entity_instance_as_group |
entity_instance_ref)>
<!ATTLIST attribute_instance
      express_attribute_name NMTOKEN #REQUIRED
      express_attribute_type  (explicit | inverse | derived) "explicit">

<!ELEMENT bag_literal
 (binary_literal* | boolean_literal* | integer_literal* | logical_literal*
| real_literal* | string_literal* | bag_literal* | list_literal* |
set_literal* | array_literal* | type_literal* |
  (entity_instance | entity_instance_as_group | entity_instance_ref)*)>
<!ATTLIST bag_literal
      express_constant_name NMTOKEN #IMPLIED >

<!ELEMENT entity_instance
 (
  (inherited_attribute_instance | attribute_instance)*,
partial_entity_instance*)>
<!ATTLIST entity_instance
      express_constant_name NMTOKEN #IMPLIED
      express_entity_name NMTOKEN #REQUIRED
      express_schema_name NMTOKEN #IMPLIED
      id ID #REQUIRED >

<!ELEMENT entity_instance_as_group
 (partial_entity_instance+)>
<!ATTLIST entity_instance_as_group
      express_constant_name NMTOKEN #IMPLIED
      id ID #REQUIRED >

<!ELEMENT entity_instance_ref EMPTY>
<!ATTLIST entity_instance_ref
      refid IDREF #REQUIRED
      reftype CDATA #FIXED "refid (entity_instance |
entity_instance_as_group | partial_entity_instance | external_refid)">


<!ELEMENT inherited_attribute_instance
 (binary_literal | boolean_literal | integer_literal | logical_literal |
real_literal | string_literal | bag_literal | list_literal | set_literal |
array_literal | type_literal | entity_instance | entity_instance_as_group |
entity_instance_ref)>
<!ATTLIST inherited_attribute_instance
      express_attribute_name NMTOKEN #REQUIRED
      express_attribute_type  (explicit | inverse | derived) "explicit">
```



```
<!ELEMENT list_literal
 (binary_literal* | boolean_literal* | integer_literal* | logical_literal*
| real_literal* | string_literal* | bag_literal* | list_literal* |
set_literal* | array_literal* | type_literal* |
  (entity_instance | entity_instance_as_group | entity_instance_ref)*)>
<!ATTLIST list_literal
      express_constant_name NMTOKEN #IMPLIED >

<!ELEMENT partial_entity_instance
 (
  (inherited_attribute_instance | attribute_instance)*,
partial_entity_instance*)>
<!ATTLIST partial_entity_instance
      express_entity_name NMTOKEN #REQUIRED
      express_schema_name NMTOKEN #IMPLIED
      id ID #IMPLIED >

<!ELEMENT schema_instance
 (entity_instance | entity_instance_as_group | external_refid)*>
<!ATTLIST schema_instance
      complete_population  (yes | no) "yes"
      express_schema_identifier CDATA #IMPLIED
      express_schema_name NMTOKEN #REQUIRED
      express_schema_version CDATA #IMPLIED
      id ID #IMPLIED
      xml:lang CDATA #IMPLIED
      refid IDREF #IMPLIED
      reftype CDATA #FIXED "refid (express_schema | external_refid)">

<!ELEMENT set_literal
 (binary_literal* | boolean_literal* | integer_literal* | logical_literal*
| real_literal* | string_literal* | bag_literal* | list_literal* |
set_literal* | array_literal* | type_literal* |
  (entity_instance | entity_instance_as_group | entity_instance_ref)*)>
<!ATTLIST set_literal
      express_constant_name NMTOKEN #IMPLIED >

<!ELEMENT type_literal
 (binary_literal | boolean_literal | integer_literal | logical_literal |
real_literal | string_literal | bag_literal | list_literal | set_literal |
array_literal | type_literal | entity_instance | entity_instance_as_group |
entity_instance_ref | enumeration_ref)>
<!ATTLIST type_literal
      express_constant_name NMTOKEN #IMPLIED
      express_schema_name NMTOKEN #IMPLIED
   express_type_name NMTOKEN #REQUIRED
   enumeration_domain NMTOKENS #IMPLIED >
```


Annex C
(normative)

Representing EXPRESS schemas

C.1 Introduction

This annex provides the XML declarations for elements used to represent EXPRESS language constructs. It uses some elements from the markup declaration set for late bound data specified in B.2.

The values of all XML attributes in the EXPRESS language markup declaration set that refer to EXPRESS identifiers shall be specified in lower case.

In this markup declaration set, element declarations that correspond directly to constructs in the EXPRESS language contain the `express-production` XML attribute. The values of that XML attribute are taken from ISO 10303-11 Annex A and are the lexical elements of the EXPRESS language syntax. The value of the `express-production` XML attribute signifies the EXPRESS construct upon which the XML element type declaration is based. This is applicable to those element type declarations that are specified in B.2 as well.

C.2 Representation of EXPRESS remarks in the XML representation EXPRESS schemas

EXPRESS remarks that appear in an EXPRESS schema shall be represented in XML according to the following rules:

— Normal rules of XML shall be used for quoting characters that are not allowed in PCDATA.

— An embedded remark shall be included as an `embedded_remark` as the first item in the tag immediately following the remark. The start delimiters "(*" and end delimiters, "*)" shall be removed.

EXAMPLE 1 – An example remark follows.

```
(* The following entity is an example of
how embedded remarks are handled *)
ENTITY example;
capacity: INTEGER;
END_ENTITY;
```

would become:

```
<entity_decl>
  <embedded_remark>The following entity is an example of
how embedded remarks are handled </embedded_remark>
 <entity_id>example</entity_id>
<explicit_attr_block>
   <explicit_attr>
```



```
      <attribute_id>capacity</attribute_id>
      <base_type>
        <integer/>            </base_type>
    </explicit_attr>
  </explicit_attr_block>
</entity_decl>
```

— Embedded remarks which are enclosed within embedded remarks shall be embedded in the XML and the start delimiter "(*" and end delimiter "*)" shall be removed.

EXAMPLE 2 – A nested embedded remark example follows.

```
(* This is a nested embedded remark,
(* this remark being nested *)
within this one *)
```

would become:

```
<embedded_remark> This is a nested embedded remark,
<embedded_remark> this remark being nested </embedded_remark>
within this one </embedded_remark>
```

— Tail remarks shall be represented using `<tail_remark>` at the start of the item immediately before the tail remark if this is able to accommodate remarks, and if not then the item immediately before this. The start delimiters "--" shall be removed.

EXAMPLE 3 – A tail remark example follows.

```
ENTITY example;
capacity: INTEGER; -- decided against using REAL here.
END_ENTITY;
```

would become:

```
<entity_decl>
  <entity_id>example</entity_id>
<explicit_attr_block>
    <explicit_attr>
      <attribute_id>capacity</attribute_id>
      <base_type>
        <tail_remark>decided against using REAL here.</tail_remark>
        <integer/>
      </base_type>
    </explicit_attr>
  </explicit_attr_block>
</entity_decl>
```

Where two separate remarks are to be included as a single XML remark, these shall be embedded within a single `embedded_remark` and some information loss may occur.

EXAMPLE 4 – The placement of a remark is lost in the following example.

```
  (* The following entity is an example of
```



```
how embedded remarks are handled *)
ENTITY example; -- could not think of a better name for this entity.
capacity: INTEGER;
END_ENTITY;
```

would become:

```
<entity_decl>
  <embedded_remark><embedded_remark>The following entity is an
example of how embedded remarks are handled
</embedded_remark><embedded_remark>could not think of a better name
for this entity.</embedded_remark></embedded_remark>
 <entity_id>example</entity_id>
<explicit_attr_block>
   <explicit_attr>
    <attribute_id>capacity</attribute_id>
    <base_type>
     <integer/>
 </base_type>
  </explicit_attr>
 </explicit_attr_block>
</entity_decl>
```

NOTE - Where particular conventions have been used in the original EXPRESS schema for remarks, these may be used to produce a more sensible result in the XML if appropriate. For example the convention of adding a remark after each END_ENTITY to indicate the EXPRESS entity name just ended would be added to the next EXPRESS entity declaration according to the rules above. However, it may be more sensible to ignore such a remark or to replace it with an XML comment.

C.3 EXPRESS language markup declarations

The following markup declarations, when combined with then markup declarations specified in B.2, provide for the representation of EXPRESS schemas in an XML document.

```
<!ELEMENT abs EMPTY>
<!ATTLIST abs
      express-production CDATA #FIXED "ABS">

<!ELEMENT abstract_supertype
 (
  (embedded_remark | tail_remark)?,
  (entity_ref | supertype_one_of | supertype_and_or | supertype_and)?)>
<!ATTLIST abstract_supertype
      express-production CDATA #FIXED "abstract_supertype_declaration">

<!ELEMENT acos EMPTY>
<!ATTLIST acos
      express-production CDATA #FIXED "ACOS">

<!ELEMENT add EMPTY>
<!ATTLIST add
      express-production CDATA #FIXED "+">
```



```
<!ELEMENT aggregate_initializer
 (
  (embedded_remark | tail_remark)?, element_list)>
<!ATTLIST aggregate_initializer
      express-production CDATA #FIXED "aggregate_initializer">

<!ELEMENT aggregate_source
 (
  (embedded_remark | tail_remark)?,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression))>
<!ATTLIST aggregate_source
      express-production CDATA #FIXED "aggregate_source">

<!ELEMENT aggregate_type
 (
  (embedded_remark | tail_remark)?,
  (aggregate_type | general_array_type | general_bag_type |
general_list_type | general_set_type | generic_type | entity_ref | type_ref
| binary | boolean | integer | logical | number | real | string),
  (type_label_id | type_label_ref)?)>
<!ATTLIST aggregate_type
      express-production CDATA #FIXED "aggregate_type">

<!ELEMENT algorithm_head
 (
  (embedded_remark | tail_remark)?, declaration_block?, constant_block?,
local_variable_block?)>
<!ATTLIST algorithm_head
      express-production CDATA #FIXED "algorithm_head">

<!ELEMENT alias_stmt
 (
  (embedded_remark | tail_remark)?, variable_id,
  (parameter_ref | variable_ref), qualifier?, statement_block)>
<!ATTLIST alias_stmt
      express-production CDATA #FIXED "alias_stmt">

<!ELEMENT and EMPTY>
<!ATTLIST and
      express-production CDATA #FIXED "AND">

<!ELEMENT applies_to_entities
 (
  (embedded_remark | tail_remark)?, entity_ref+)>
<!ATTLIST applies_to_entities
      express-production CDATA #FIXED "rule_head">

<!ELEMENT array_type
 (
```



```
  (embedded_remark | tail_remark)?, index_spec, base_type, optional?,
unique?)>
<!ATTLIST array_type
      express-production CDATA #FIXED "array_type">

<!ELEMENT asin EMPTY>
<!ATTLIST asin
      express-production CDATA #FIXED "ASIN">

<!ELEMENT assignment_stmt
 (
  (embedded_remark | tail_remark)?,
  (parameter_ref | variable_ref), qualifier?,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression | relation_expression))>
<!ATTLIST assignment_stmt
      express-production CDATA #FIXED "assignment_stmt">

<!ELEMENT atan EMPTY>
<!ATTLIST atan
      express-production CDATA #FIXED "ATAN">

<!ELEMENT attribute_id
 (#PCDATA)>
<!ATTLIST attribute_id
      express-production CDATA #FIXED "attribute_id"
      id ID #IMPLIED >

<!ELEMENT attribute_ref
 (#PCDATA)>
<!ATTLIST attribute_ref
      express-production CDATA #FIXED "attribute_ref"
      refid IDREF #IMPLIED
      reftype CDATA #FIXED "refid (attribute_id)">

<!ELEMENT bag_type
 (
  (embedded_remark | tail_remark)?, bound_spec?, base_type)>
<!ATTLIST bag_type
      express-production CDATA #FIXED "bag_type">

<!ELEMENT base_type
 (
  (embedded_remark | tail_remark)?,
  (array_type | bag_type | list_type | set_type | binary | boolean |
integer | logical | number | real | string | entity_ref | type_ref))>
<!ATTLIST base_type
      express-production CDATA #FIXED "base_type">

<!ELEMENT binary
 (
```



```
  (embedded_remark | tail_remark)?, width_spec?)>
<!ATTLIST binary
      express-production CDATA #FIXED "binary_type">

<!ELEMENT blength EMPTY>
<!ATTLIST blength
      express-production CDATA #FIXED "BLENGTH">

<!ELEMENT boolean EMPTY>
<!ATTLIST boolean
      express-production CDATA #FIXED "boolean_type">

<!ELEMENT bound_spec
 (
  (embedded_remark | tail_remark)?, lower_bound, upper_bound)>
<!ATTLIST bound_spec
      express-production CDATA #FIXED "bound_spec">

<!ELEMENT case_action
 (
  (embedded_remark | tail_remark)?, case_label,
  (alias_stmt | assignment_stmt | case_stmt | statement_block | escape_stmt
| if_stmt | null_stmt | procedure_call_stmt | repeat_stmt | return_stmt |
skip_stmt))>
<!ATTLIST case_action
      express-production CDATA #FIXED "case_action">

<!ELEMENT case_label
 (
  (embedded_remark | tail_remark)?,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression | relation_expression)+)>
<!ATTLIST case_label
      express-production CDATA #FIXED "case_label">

<!ELEMENT case_stmt
 (
  (embedded_remark | tail_remark)?,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression | relation_expression),
case_action*, otherwise?)>
<!ATTLIST case_stmt
      express-production CDATA #FIXED "case_stmt">

<!ELEMENT complex_entity_constructor EMPTY>
<!ATTLIST complex_entity_constructor
      express-production CDATA #FIXED "||">
```



```
<!ELEMENT constant_block
 (
  (embedded_remark | tail_remark)?, constant_decl*)>
<!ATTLIST constant_block
      express-production CDATA #FIXED "constant_decl">

<!ELEMENT constant_decl
 (
  (embedded_remark | tail_remark)?, constant_id, base_type,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression | relation_expression))>
<!ATTLIST constant_decl
      express-production CDATA #FIXED "constant_body">

<!ELEMENT constant_id
 (#PCDATA)>
<!ATTLIST constant_id
      express-production CDATA #FIXED "constant_id"
      id ID #IMPLIED >

<!ELEMENT constant_import
 (
  (embedded_remark | tail_remark)?, constant_id, constant_ref)>
<!ATTLIST constant_import
      express-production CDATA #FIXED "resource_or_rename">

<!ELEMENT constant_ref
 (#PCDATA)>
<!ATTLIST constant_ref
      express-production CDATA #FIXED "constant_ref"
      refid IDREF #IMPLIED
      reftype CDATA #FIXED "refid (constant_id)">

<!ELEMENT const_e EMPTY>
<!ATTLIST const_e
      express-production CDATA #FIXED "CONST_E">

<!ELEMENT cos EMPTY>
<!ATTLIST cos
      express-production CDATA #FIXED "COS">

<!ELEMENT declaration_block
 (
  (embedded_remark | tail_remark)?,
  (entity_decl | function_decl | procedure_decl | type_decl)*)>
<!ATTLIST declaration_block
      express-production CDATA #FIXED "declaration">

<!ELEMENT derived_attr
 (
```



```
  (embedded_remark | tail_remark)?,
  (attribute_id | qualified_attribute), base_type,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression | relation_expression))>
<!ATTLIST derived_attr
      express-production CDATA #FIXED "derived_attr">

<!ELEMENT derive_clause
 (
  (embedded_remark | tail_remark)?, derived_attr+)>
<!ATTLIST derive_clause
      express-production CDATA #FIXED "derive_clause">

<!ELEMENT domain_rule
 (
  (embedded_remark | tail_remark)?, label?, logical_expression)>
<!ATTLIST domain_rule
      express-production CDATA #FIXED "domain_rule">

<!ELEMENT element_item
 (
  (embedded_remark | tail_remark)?,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression | relation_expression),
repetition?)>
<!ATTLIST element_item
      express-production CDATA #FIXED "element">

<!ELEMENT element_list
 (
  (embedded_remark | tail_remark)?, element_item*)>
<!ATTLIST element_list
      express-production CDATA #FIXED "element">

<!ELEMENT entity_constructor
 (
  (embedded_remark | tail_remark)?, entity_ref,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression | relation_expression)*)>
<!ATTLIST entity_constructor
      express-production CDATA #FIXED "entity_constructor">

<!ELEMENT entity_decl
```



```
 (
  (embedded_remark | tail_remark)?, entity_id,
  (abstract_supertype | supertype_of)?, subtype_of?, explicit_attr_block?,
derive_clause?, inverse_clause?, unique_clause?, where_clause?)>
<!ATTLIST entity_decl
      express-production CDATA #FIXED "entity_decl">

<!ELEMENT entity_id
 (#PCDATA)>
<!ATTLIST entity_id
      express-production CDATA #FIXED "entity_id"
      id ID #IMPLIED >

<!ELEMENT entity_import
 (
  (embedded_remark | tail_remark)?, entity_id, entity_ref)>
<!ATTLIST entity_import
      express-production CDATA #FIXED "resource_or_rename">

<!ELEMENT entity_ref
 (#PCDATA)>
<!ATTLIST entity_ref
      express-production CDATA #FIXED "entity_ref"
      refid IDREF #IMPLIED
      reftype CDATA #FIXED "refid (entity_id)">

<!ELEMENT enumeration
 (
  (embedded_remark | tail_remark)?, enumeration_id+)>
<!ATTLIST enumeration
      express-production CDATA #FIXED "enumeration_type">

<!ELEMENT enumeration_id
 (#PCDATA)>
<!ATTLIST enumeration_id
      express-production CDATA #FIXED "enumeration_id"
      id ID #IMPLIED >

<!ELEMENT enumeration_reference
 (
  (embedded_remark | tail_remark)?, type_ref?, enumeration_ref)>
<!ATTLIST enumeration_reference
      express-production CDATA #FIXED "enumeration_reference">

<!ELEMENT equal EMPTY>
<!ATTLIST equal
      express-production CDATA #FIXED "=">

<!ELEMENT escape_stmt EMPTY>
<!ATTLIST escape_stmt
      express-production CDATA #FIXED "escape_stmt">

<!ELEMENT exists EMPTY>
<!ATTLIST exists
      express-production CDATA #FIXED "EXISTS">
```



```
<!ELEMENT exp EMPTY>
<!ATTLIST exp
      express-production CDATA #FIXED "EXP">

<!ELEMENT explicit_attr
 (
  (embedded_remark | tail_remark)?,
  (attribute_id | qualified_attribute), optional?, base_type)>
<!ATTLIST explicit_attr
      express-production CDATA #FIXED "explicit_attr">

<!ELEMENT explicit_attr_block
 (
  (embedded_remark | tail_remark)?, explicit_attr+)>
<!ATTLIST explicit_attr_block
      express-production CDATA #FIXED "explicit_attr_block">

<!ELEMENT factor
 (raise_to_power,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op),
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op))>
<!ATTLIST factor
      express-production CDATA #FIXED "factor">

<!ELEMENT fixed EMPTY>
<!ATTLIST fixed
      express-production CDATA #FIXED "FIXED">

<!ELEMENT formal_parameter
 (
  (embedded_remark | tail_remark)?, parameter_id,
  (aggregate_type | general_array_type | general_bag_type |
general_list_type | general_set_type | generic_type | entity_ref | type_ref
| binary | boolean | integer | logical | number | real | string))>
<!ATTLIST formal_parameter
      express-production CDATA #FIXED "formal_parameter">

<!ELEMENT formal_parameter_block
 (
  (embedded_remark | tail_remark)?, formal_parameter*)>
<!ATTLIST formal_parameter_block
      express-production CDATA #FIXED "formal_parameter">

<!ELEMENT format EMPTY>
```



```
<!ATTLIST format
      express-production CDATA #FIXED "FORMAT">

<!ELEMENT function_call
 (
  (embedded_remark | tail_remark)?,
  (abs | acos | asin | atan | blength | cos | exists | exp | format |
hibound | hiindex | length | lobound | loindex | log | log2 | log10 | nvl |
odd | rolesof | sin | sizeof | sqrt | tan | typeof | usedin | value |
value_in | value_unique | function_ref),
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression | relation_expression)*)>
<!ATTLIST function_call
      express-production CDATA #FIXED "function_call">

<!ELEMENT function_decl
 (
  (embedded_remark | tail_remark)?, function_id, formal_parameter_block?,
function_return_type, algorithm_head?, statement_block)>
<!ATTLIST function_decl
      express-production CDATA #FIXED "function_decl">

<!ELEMENT function_id
 (#PCDATA)>
<!ATTLIST function_id
      express-production CDATA #FIXED "function_id"
      id ID #IMPLIED >

<!ELEMENT function_import
 (
  (embedded_remark | tail_remark)?, function_id, function_ref)>
<!ATTLIST function_import
      express-production CDATA #FIXED "resource_or_rename">

<!ELEMENT function_ref
 (#PCDATA)>
<!ATTLIST function_ref
      express-production CDATA #FIXED "function_ref"
      refid IDREF #IMPLIED
      reftype CDATA #FIXED "refid (function_id)">

<!ELEMENT function_return_type
 (aggregate_type | general_array_type | general_bag_type |
general_list_type | general_set_type | generic_type | entity_ref | type_ref
| binary | boolean | integer | logical | number | real | string)>
<!ATTLIST function_return_type
      express-production CDATA #FIXED "parameter_type">

<!ELEMENT general_array_type
 (
  (embedded_remark | tail_remark)?,
```



```
  (aggregate_type | general_array_type | general_bag_type |
general_list_type | general_set_type | generic_type | entity_ref | type_ref
| binary | boolean | integer | logical | number | real | string),
bound_spec?, optional?, unique?)>
<!ATTLIST general_array_type
      express-production CDATA #FIXED "general_array_type">

<!ELEMENT general_bag_type
 (
  (embedded_remark | tail_remark)?,
  (aggregate_type | general_array_type | general_bag_type |
general_list_type | general_set_type | generic_type | entity_ref | type_ref
| binary | boolean | integer | logical | number | real | string),
bound_spec?)>
<!ATTLIST general_bag_type
      express-production CDATA #FIXED "general_bag_type">

<!ELEMENT general_list_type
 (
  (embedded_remark | tail_remark)?,
  (aggregate_type | general_array_type | general_bag_type |
general_list_type | general_set_type | generic_type | entity_ref | type_ref
| binary | boolean | integer | logical | number | real | string),
bound_spec?, unique?)>
<!ATTLIST general_list_type
      express-production CDATA #FIXED "general_list_type">

<!ELEMENT general_set_type
 (
  (embedded_remark | tail_remark)?,
  (aggregate_type | general_array_type | general_bag_type |
general_list_type | general_set_type | generic_type | entity_ref | type_ref
| binary | boolean | integer | logical | number | real | string),
bound_spec?)>
<!ATTLIST general_set_type
      express-production CDATA #FIXED "general_set_type">

<!ELEMENT generic_type
 (
  (embedded_remark | tail_remark)?,
  (type_label_id | type_label_ref)?)>
<!ATTLIST generic_type
      express-production CDATA #FIXED "generic_type">

<!ELEMENT greater_than EMPTY>
<!ATTLIST greater_than
      express-production CDATA #FIXED "&gt;">

<!ELEMENT greater_than_or_equal EMPTY>
<!ATTLIST greater_than_or_equal
      express-production CDATA #FIXED "&gt;=">

<!ELEMENT hibound EMPTY>
<!ATTLIST hibound
      express-production CDATA #FIXED "HIBOUND">
```



```
<!ELEMENT high_index
 (
  (embedded_remark | tail_remark)?,
  (integer_literal | numeric_expression))>
<!ATTLIST high_index
      express-production CDATA #FIXED "index_2">

<!ELEMENT hiindex EMPTY>
<!ATTLIST hiindex
      express-production CDATA #FIXED "HIINDEX">

<!ELEMENT if_stmt
 (
  (embedded_remark | tail_remark)?, logical_expression, statement_block,
statement_block?)>
<!ATTLIST if_stmt
      express-production CDATA #FIXED "if_stmt">

<!ELEMENT import_all EMPTY>
<!ATTLIST import_all
      express-production CDATA #FIXED "#NONE">

<!ELEMENT in EMPTY>
<!ATTLIST in
      express-production CDATA #FIXED "IN">

<!ELEMENT increment
 (
  (embedded_remark | tail_remark)?,
  (integer_literal | numeric_expression))>
<!ATTLIST increment
      express-production CDATA #FIXED "increment">

<!ELEMENT increment_control
 (
  (embedded_remark | tail_remark)?, variable_id, lower_bound, upper_bound,
increment?)>
<!ATTLIST increment_control
      express-production CDATA #FIXED "increment_control">

<!ELEMENT index_qualifier
 (
  (embedded_remark | tail_remark)?, low_index, high_index?)>
<!ATTLIST index_qualifier
      express-production CDATA #FIXED "index_qualifier">

<!ELEMENT index_spec
 (
  (embedded_remark | tail_remark)?, low_index, high_index?)>
<!ATTLIST index_spec
      express-production CDATA #FIXED "bound_spec">

<!ELEMENT insert EMPTY>
<!ATTLIST insert
```



```
      express-production CDATA #FIXED "INSERT">

<!ELEMENT instance_equal EMPTY>
<!ATTLIST instance_equal
      express-production CDATA #FIXED ":=:">

<!ELEMENT instance_not_equal EMPTY>
<!ATTLIST instance_not_equal
      express-production CDATA #FIXED ":&lt;&gt;:">

<!ELEMENT integer EMPTY>
<!ATTLIST integer
      express-production CDATA #FIXED "integer_type">

<!ELEMENT integer_divide EMPTY>
<!ATTLIST integer_divide
      express-production CDATA #FIXED "DIV">

<!ELEMENT interface_specification_block
 (
  (embedded_remark | tail_remark)?,
  (reference_from | use_from)+)>
<!ATTLIST interface_specification_block
      express-production CDATA #FIXED "interface_specification">

<!ELEMENT interval
 (
  (embedded_remark | tail_remark)?,
  (interval_low_inclusive | interval_low_exclusive), interval_item,
  (interval_high_inclusive | interval_high_exclusive))>
<!ATTLIST interval
      express-production CDATA #FIXED "interval">

<!ELEMENT interval_high_exclusive
 (
  (embedded_remark | tail_remark)?,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression))>
<!ATTLIST interval_high_exclusive
      express-production CDATA #FIXED "interval_high">

<!ELEMENT interval_high_inclusive
 (
  (embedded_remark | tail_remark)?,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression))>
<!ATTLIST interval_high_inclusive
```



```
      express-production CDATA #FIXED "interval_high">

<!ELEMENT interval_item
 (
  (embedded_remark | tail_remark)?,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression))>
<!ATTLIST interval_item
      express-production CDATA #FIXED "interval_item">

<!ELEMENT interval_low_exclusive
 (
  (embedded_remark | tail_remark)?,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression))>
<!ATTLIST interval_low_exclusive
      express-production CDATA #FIXED "interval_low">

<!ELEMENT interval_low_inclusive
 (
  (embedded_remark | tail_remark)?,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression))>
<!ATTLIST interval_low_inclusive
      express-production CDATA #FIXED "interval_low">

<!ELEMENT inverse_attr
 (
  (embedded_remark | tail_remark)?,
  (attribute_id | qualified_attribute), entity_ref, attribute_ref,
  (inverse_set | inverse_bag)?)>
<!ATTLIST inverse_attr
      express-production CDATA #FIXED "inverse_attr">

<!ELEMENT inverse_bag
 (
  (embedded_remark | tail_remark)?, bound_spec?)>
<!ATTLIST inverse_bag
      express-production CDATA #FIXED "BAG">

<!ELEMENT inverse_clause
 (
  (embedded_remark | tail_remark)?, inverse_attr+)>
```



```
<!ATTLIST inverse_clause
      express-production CDATA #FIXED "inverse_clause">

<!ELEMENT inverse_set
 (
  (embedded_remark | tail_remark)?, bound_spec?)>
<!ATTLIST inverse_set
      express-production CDATA #FIXED "SET">

<!ELEMENT label
 (#PCDATA)>
<!ATTLIST label
      express-production CDATA #FIXED "label">

<!ELEMENT length EMPTY>
<!ATTLIST length
      express-production CDATA #FIXED "LENGTH">

<!ELEMENT less_than EMPTY>
<!ATTLIST less_than
      express-production CDATA #FIXED "&lt;">

<!ELEMENT less_than_or_equal EMPTY>
<!ATTLIST less_than_or_equal
      express-production CDATA #FIXED "&lt;=">

<!ELEMENT like EMPTY>
<!ATTLIST like
      express-production CDATA #FIXED "LIKE">

<!ELEMENT list_type
 (
  (embedded_remark | tail_remark)?, bound_spec?, base_type, unique?)>
<!ATTLIST list_type
      express-production CDATA #FIXED "list_type">

<!ELEMENT lobound EMPTY>
<!ATTLIST lobound
      express-production CDATA #FIXED "LOBOUND">

<!ELEMENT local_variable_block
 (
  (embedded_remark | tail_remark)?, local_variable_decl*)>
<!ATTLIST local_variable_block
      express-production CDATA #FIXED "local_decl">

<!ELEMENT local_variable_decl
 (
  (embedded_remark | tail_remark)?, variable_id,
  (aggregate_type | general_array_type | general_bag_type |
general_list_type | general_set_type | generic_type | entity_ref | type_ref
| binary | boolean | integer | logical | number | real | string),
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
```



```
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression | relation_expression)?)>
<!ATTLIST local_variable_decl
      express-production CDATA #FIXED "local_variable">

<!ELEMENT log EMPTY>
<!ATTLIST log
      express-production CDATA #FIXED "LOG">

<!ELEMENT log10 EMPTY>
<!ATTLIST log10
      express-production CDATA #FIXED "LOG10">

<!ELEMENT log2 EMPTY>
<!ATTLIST log2
      express-production CDATA #FIXED "LOG2">

<!ELEMENT logical EMPTY>
<!ATTLIST logical
      express-production CDATA #FIXED "logical_type">

<!ELEMENT logical_expression
 (
  (embedded_remark | tail_remark)?,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression | relation_expression))>
<!ATTLIST logical_expression
      express-production CDATA #FIXED "logical_expression">

<!ELEMENT loindex EMPTY>
<!ATTLIST loindex
      express-production CDATA #FIXED "LOINDEX">

<!ELEMENT lower_bound
 (
  (embedded_remark | tail_remark)?,
  (integer_literal | numeric_expression))>
<!ATTLIST lower_bound
      express-production CDATA #FIXED "bound_1">

<!ELEMENT low_index
 (
  (embedded_remark | tail_remark)?,
  (integer_literal | numeric_expression))>
<!ATTLIST low_index
      express-production CDATA #FIXED "index_1">

<!ELEMENT mod EMPTY>
<!ATTLIST mod
      express-production CDATA #FIXED "MOD">
```



```
<!ELEMENT multiply EMPTY>
<!ATTLIST multiply
      express-production CDATA #FIXED "*">

<!ELEMENT negate EMPTY>
<!ATTLIST negate
      express-production CDATA #FIXED "-">

<!ELEMENT not EMPTY>
<!ATTLIST not
      express-production CDATA #FIXED "NOT">

<!ELEMENT not_equal EMPTY>
<!ATTLIST not_equal
      express-production CDATA #FIXED "&lt;&gt;">

<!ELEMENT null_stmt EMPTY>
<!ATTLIST null_stmt
      express-production CDATA #FIXED "null_stmt">

<!ELEMENT number EMPTY>
<!ATTLIST number
      express-production CDATA #FIXED "number_type">

<!ELEMENT numeric_expression
 (
  (embedded_remark | tail_remark)?,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression))>
<!ATTLIST numeric_expression
      express-production CDATA #FIXED "numeric_expression">

<!ELEMENT nvl EMPTY>
<!ATTLIST nvl
      express-production CDATA #FIXED "NVL">

<!ELEMENT odd EMPTY>
<!ATTLIST odd
      express-production CDATA #FIXED "ODD">

<!ELEMENT optional EMPTY>
<!ATTLIST optional
      express-production CDATA #FIXED "OPTIONAL">

<!ELEMENT or EMPTY>
<!ATTLIST or
      express-production CDATA #FIXED "OR">

<!ELEMENT otherwise
 (
```



```
  (embedded_remark | tail_remark)?,
  (alias_stmt | assignment_stmt | case_stmt | statement_block | escape_stmt
| if_stmt | null_stmt | procedure_call_stmt | repeat_stmt | return_stmt |
skip_stmt))>
<!ATTLIST otherwise
      express-production CDATA #FIXED "OTHERWISE">

<!ELEMENT parameter_id
 (#PCDATA)>
<!ATTLIST parameter_id
      express-production CDATA #FIXED "parameter_id"
      id ID #IMPLIED >

<!ELEMENT parameter_ref
 (#PCDATA)>
<!ATTLIST parameter_ref
      express-production CDATA #FIXED "parameter_ref"
      refid IDREF #IMPLIED
      reftype CDATA #FIXED "refid (parameter_id)">

<!ELEMENT parenthetic_expression
 (
  (embedded_remark | tail_remark)?,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression | relation_expression))>
<!ATTLIST parenthetic_expression
      express-production CDATA #FIXED "()">

<!ELEMENT pi EMPTY>
<!ATTLIST pi
      express-production CDATA #FIXED "PI">

<!ELEMENT plus EMPTY>
<!ATTLIST plus
      express-production CDATA #FIXED "+">

<!ELEMENT population
 (
  (embedded_remark | tail_remark)?, entity_ref)>
<!ATTLIST population
      express-production CDATA #FIXED "population">

<!ELEMENT precision_spec
 (
  (embedded_remark | tail_remark)?,
  (integer_literal | numeric_expression))>
<!ATTLIST precision_spec
      express-production CDATA #FIXED "precision_spec">

<!ELEMENT procedure_call_stmt
 (
```



```
  (embedded_remark | tail_remark)?,
  (
   (insert | remove | procedure_ref),
   (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression | relation_expression)*))>
<!ATTLIST procedure_call_stmt
      express-production CDATA #FIXED "procedure_call_stmt">

<!ELEMENT procedure_decl
 (
  (embedded_remark | tail_remark)?, procedure_id,
procedure_formal_parameter_block?, algorithm_head?, statement_block?)>
<!ATTLIST procedure_decl
      express-production CDATA #FIXED "procedure_decl">

<!ELEMENT procedure_formal_parameter_block
 (
  (embedded_remark | tail_remark)?,
  (formal_parameter | var_formal_parameter)*)>
<!ATTLIST procedure_formal_parameter_block
      express-production CDATA #FIXED "formal_parameter">

<!ELEMENT procedure_id
 (#PCDATA)>
<!ATTLIST procedure_id
      express-production CDATA #FIXED "procedure_id"
      id ID #IMPLIED >

<!ELEMENT procedure_import
 (
  (embedded_remark | tail_remark)?, procedure_id, procedure_ref)>
<!ATTLIST procedure_import
      express-production CDATA #FIXED "resource_or_rename">

<!ELEMENT procedure_ref
 (#PCDATA)>
<!ATTLIST procedure_ref
      express-production CDATA #FIXED "procedure_ref"
      refid IDREF #IMPLIED
      reftype CDATA #FIXED "refid (procedure_id)">

<!ELEMENT qualified_attribute
 (
  (embedded_remark | tail_remark)?, entity_ref, attribute_ref)>
<!ATTLIST qualified_attribute
      express-production CDATA #FIXED "qualified_attribute">

<!ELEMENT qualified_factor
 (
  (embedded_remark | tail_remark)?,
```



```
  (attribute_ref | const_e | pi | self | unset | constant_ref |
function_call | parameter_ref | variable_ref | population), qualifier)>
<!ATTLIST qualified_factor
      express-production CDATA #FIXED "qualifiable_factor">

<!ELEMENT qualifier
 (
  (embedded_remark | tail_remark)?,
  (attribute_ref | entity_ref | index_qualifier)*)>
<!ATTLIST qualifier
      express-production CDATA #FIXED "qualifier">

<!ELEMENT query
 (
  (embedded_remark | tail_remark)?, variable_id,
  (aggregate_source, logical_expression))>
<!ATTLIST query
      express-production CDATA #FIXED "query_expression">

<!ELEMENT raise_to_power EMPTY>
<!ATTLIST raise_to_power
      express-production CDATA #FIXED "**">

<!ELEMENT real
 (
  (embedded_remark | tail_remark)?, precision_spec?)>
<!ATTLIST real
      express-production CDATA #FIXED "real_type">

<!ELEMENT real_divide EMPTY>
<!ATTLIST real_divide
      express-production CDATA #FIXED "/">

<!ELEMENT reference_from
 (
  (embedded_remark | tail_remark)?, schema_ref,
  (import_all |
   (constant_import | entity_import | function_import | procedure_import |
type_import)+))>
<!ATTLIST reference_from
      express-production CDATA #FIXED "reference_clause">

<!ELEMENT relation_expression
 (
  (embedded_remark | tail_remark)?,
  (less_than | greater_than | less_than_or_equal | greater_than_or_equal |
not_equal | equal | instance_not_equal | instance_equal | in | like),
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression),
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
```



```
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression))>
<!ATTLIST relation_expression
      express-production CDATA #FIXED "expression">

<!ELEMENT remove EMPTY>
<!ATTLIST remove
      express-production CDATA #FIXED "REMOVE">

<!ELEMENT repeat_control
 (
  (embedded_remark | tail_remark)?, increment_control, while?, until?)>
<!ATTLIST repeat_control
      express-production CDATA #FIXED "repeat_control">

<!ELEMENT repeat_stmt
 (
  (embedded_remark | tail_remark)?, repeat_control, statement_block)>
<!ATTLIST repeat_stmt
      express-production CDATA #FIXED "repeat_stmt">

<!ELEMENT repetition
 (
  (embedded_remark | tail_remark)?,
  (integer_literal | numeric_expression))>
<!ATTLIST repetition
      express-production CDATA #FIXED "repetition">

<!ELEMENT return_stmt
 (
  (embedded_remark | tail_remark)?,
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term | simple_expression | relation_expression)?)>
<!ATTLIST return_stmt
      express-production CDATA #FIXED "return_stmt">

<!ELEMENT rolesof EMPTY>
<!ATTLIST rolesof
      express-production CDATA #FIXED "ROLESOF">

<!ELEMENT rule_decl
 (
  (embedded_remark | tail_remark)?, rule_id, applies_to_entities,
algorithm_head?, statement_block?, where_clause)>
<!ATTLIST rule_decl
      express-production CDATA #FIXED "rule_decl">

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



```
<!ATTLIST rule_id
      express-production CDATA #FIXED "rule_id">

<!ELEMENT schema_decl
 (
  (embedded_remark | tail_remark)?, schema_id,
interface_specification_block?, constant_block?,
  (entity_decl | function_decl | procedure_decl | type_decl | rule_decl)*)>
<!ATTLIST schema_decl
      express-production CDATA #FIXED "schema_decl">

<!ELEMENT schema_id
 (#PCDATA)>
<!ATTLIST schema_id
      express-production CDATA #FIXED "schema_id"
       id ID #IMPLIED >

<!ELEMENT schema_ref
 (#PCDATA)>
<!ATTLIST schema_ref
      express-production CDATA #FIXED "schema_ref"
      refid IDREF #IMPLIED
      reftype CDATA #FIXED "refid (schema_id | external_refid)">

<!ELEMENT select
 (
  (embedded_remark | tail_remark)?,
  (entity_ref | type_ref)+)>
<!ATTLIST select
      express-production CDATA #FIXED "select_type">

<!ELEMENT self EMPTY>
<!ATTLIST self
      express-production CDATA #FIXED "SELF">

<!ELEMENT set_type
 (
  (embedded_remark | tail_remark)?, bound_spec?, base_type)>
<!ATTLIST set_type
      express-production CDATA #FIXED "set_type">

<!ELEMENT simple_expression
 (
  (embedded_remark | tail_remark)?,
  (add | subtract | or | xor),
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term),
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
```



```
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term))>
<!ATTLIST simple_expression
      express-production CDATA #FIXED "simple_expression">

<!ELEMENT sin EMPTY>
<!ATTLIST sin
      express-production CDATA #FIXED "SIN">

<!ELEMENT sizeof EMPTY>
<!ATTLIST sizeof
      express-production CDATA #FIXED "SIZEOF">

<!ELEMENT skip_stmt EMPTY>
<!ATTLIST skip_stmt
      express-production CDATA #FIXED "SKIP">

<!ELEMENT sqrt EMPTY>
<!ATTLIST sqrt
      express-production CDATA #FIXED "SQRT">

<!ELEMENT statement_block
 (
  (embedded_remark | tail_remark)?,
  (alias_stmt | assignment_stmt | case_stmt | statement_block | escape_stmt
| if_stmt | null_stmt | procedure_call_stmt | repeat_stmt | return_stmt |
skip_stmt)+)>
<!ATTLIST statement_block
      express-production CDATA #FIXED "stmt">

<!ELEMENT string
 (
  (embedded_remark | tail_remark)?, width_spec?)>
<!ATTLIST string
      express-production CDATA #FIXED "string_type">

<!ELEMENT subtract EMPTY>
<!ATTLIST subtract
      express-production CDATA #FIXED "-">

<!ELEMENT subtype_of
 (
  (embedded_remark | tail_remark)?, entity_ref+)>
<!ATTLIST subtype_of
      express-production CDATA #FIXED "subtype_declaration">

<!ELEMENT supertype_and
 (
  (embedded_remark | tail_remark)?,
  (entity_ref | supertype_one_of | supertype_and_or | supertype_and)+)>
<!ATTLIST supertype_and
      express-production CDATA #FIXED "supertype_rule">

<!ELEMENT supertype_and_or
 (
```



```
  (embedded_remark | tail_remark)?,
  (entity_ref | supertype_one_of | supertype_and_or | supertype_and)+)>
<!ATTLIST supertype_and_or
      express-production CDATA #FIXED "supertype_rule">

<!ELEMENT supertype_of
 (
  (embedded_remark | tail_remark)?,
  (entity_ref | supertype_one_of | supertype_and_or | supertype_and))>
<!ATTLIST supertype_of
      express-production CDATA #FIXED "supertype_rule">

<!ELEMENT supertype_one_of
 (
  (embedded_remark | tail_remark)?,
  (entity_ref | supertype_one_of | supertype_and_or | supertype_and)+)>
<!ATTLIST supertype_one_of
      express-production CDATA #FIXED "one_of">

<!ELEMENT tan EMPTY>
<!ATTLIST tan
      express-production CDATA #FIXED "TAN">

<!ELEMENT term
 (
  (multiply | real_divide | integer_divide | mod | and |
complex_entity_constructor),
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term),
  (aggregate_initializer | entity_constructor | enumeration_reference |
interval | query | binary_literal | boolean_literal | integer_literal |
logical_literal | real_literal | string_literal | attribute_ref | const_e |
pi | self | unset | constant_ref | function_call | parameter_ref |
variable_ref | population | qualified_factor | parenthetic_expression |
unary_op | factor | term))>
<!ATTLIST term
      express-production CDATA #FIXED "term">

<!ELEMENT typeof EMPTY>
<!ATTLIST typeof
      express-production CDATA #FIXED "TYPEOF">

<!ELEMENT type_decl
 (
  (embedded_remark | tail_remark)?, type_id, underlying_type,
where_clause?)>
<!ATTLIST type_decl
      express-production CDATA #FIXED "type_decl">

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



```
<!ATTLIST type_id
      express-production CDATA #FIXED "type_id"
      id ID #IMPLIED >

<!ELEMENT type_import
 (
  (embedded_remark | tail_remark)?, type_id, type_ref)>
<!ATTLIST type_import
      express-production CDATA #FIXED "resource_or_rename">

<!ELEMENT type_label_id
 (#PCDATA)>
<!ATTLIST type_label_id
      express-production CDATA #FIXED "type_label_id"
      id ID #IMPLIED >

<!ELEMENT type_label_ref
 (#PCDATA)>
<!ATTLIST type_label_ref
      express-production CDATA #FIXED "type_label_ref"
      refid IDREF #IMPLIED
      reftype CDATA #FIXED "refid (type_label_id)">

<!ELEMENT type_ref
 (#PCDATA)>
<!ATTLIST type_ref
      express-production CDATA #FIXED "type_ref"
      refid IDREF #IMPLIED
      reftype CDATA #FIXED "refid (type_id)">

<!ELEMENT unary_op
 (
  (embedded_remark | tail_remark)?,
  (plus | negate | not),
  (binary_literal | boolean_literal | integer_literal | logical_literal |
real_literal | string_literal | attribute_ref | const_e | pi | self | unset
| constant_ref | function_call | parameter_ref | variable_ref | population
| qualified_factor | parenthetic_expression))>
<!ATTLIST unary_op
      express-production CDATA #FIXED "unary_op">

<!ELEMENT underlying_type
 (
  (embedded_remark | tail_remark)?,
  (enumeration | select | array_type | bag_type | list_type | set_type |
binary | boolean | integer | logical | number | real | string | type_ref))>
<!ATTLIST underlying_type
      express-production CDATA #FIXED "underlying_type">

<!ELEMENT unique EMPTY>
<!ATTLIST unique
      express-production CDATA #FIXED "UNIQUE">

<!ELEMENT unique_clause
 (
```



```
  (embedded_remark | tail_remark)?, unique_rule+)>
<!ATTLIST unique_clause
      express-production CDATA #FIXED "unique_clause">

<!ELEMENT unique_rule
 (
  (embedded_remark | tail_remark)?, label?,
  (attribute_ref | qualified_attribute)+)>
<!ATTLIST unique_rule
      express-production CDATA #FIXED "unique_rule">

<!ELEMENT until
 (
  (embedded_remark | tail_remark)?, logical_expression)>
<!ATTLIST until
      express-production CDATA #FIXED "until_control">

<!ELEMENT upper_bound
 (
  (embedded_remark | tail_remark)?,
  (unset | integer_literal | numeric_expression))>
<!ATTLIST upper_bound
      express-production CDATA #FIXED "bound_2">

<!ELEMENT usedin EMPTY>
<!ATTLIST usedin
      express-production CDATA #FIXED "USEDIN">

<!ELEMENT use_from
 (
  (embedded_remark | tail_remark)?, schema_ref,
  (import_all |
   (entity_import | type_import)+))>
<!ATTLIST use_from
      express-production CDATA #FIXED "use_clause">

<!ELEMENT value EMPTY>
<!ATTLIST value
      express-production CDATA #FIXED "VALUE">

<!ELEMENT value_in EMPTY>
<!ATTLIST value_in
      express-production CDATA #FIXED "VALUE_IN">

<!ELEMENT value_unique EMPTY>
<!ATTLIST value_unique
      express-production CDATA #FIXED "VALUE_UNIQUE">

<!ELEMENT variable_id
 (#PCDATA)>
<!ATTLIST variable_id
      express-production CDATA #FIXED "variable_id"
      id ID #IMPLIED >

<!ELEMENT variable_ref
```



```
 (#PCDATA)>
<!ATTLIST variable_ref
      express-production CDATA #FIXED "variable_ref"
      refid IDREF #IMPLIED
      reftype CDATA #FIXED "refid (variable_id)">

<!ELEMENT var_formal_parameter
 (
  (embedded_remark | tail_remark)?, parameter_id,
  (aggregate_type | general_array_type | general_bag_type |
general_list_type | general_set_type | generic_type | entity_ref | type_ref
| binary | boolean | integer | logical | number | real | string))>
<!ATTLIST var_formal_parameter
      express-production CDATA #FIXED "formal_parameter">

<!ELEMENT where_clause
 (
  (embedded_remark | tail_remark)?, domain_rule+)>
<!ATTLIST where_clause
      express-production CDATA #FIXED "where_clause">

<!ELEMENT while
 (
  (embedded_remark | tail_remark)?, logical_expression)>
<!ATTLIST while
      express-production CDATA #FIXED "while_control">

<!ELEMENT width_spec
 (
  (embedded_remark | tail_remark)?,
  (integer_literal | numeric_expression), fixed?)>
<!ATTLIST width_spec
      express-production CDATA #FIXED "width_spec">

<!ELEMENT xor EMPTY>
<!ATTLIST xor
      express-production CDATA #FIXED "XOR">
```


Annex D
(normative)

Base architecture document type definition

This annex specifies the DTD used for XML documents that architecturally conform to this part of ISO 10303.

The set of markup declarations specified in clause 6, Annex B and Annex C comprise the DTD. The name of the document type declaration for the architectural DTD is `iso_10303_28`. Its SYSTEM identifier is "http://www.mel.nist.gov/step/parts/part28e1/cd/iso_10303_28_base_arch.dtd". Its PUBLIC identifier is "ISO 10303-28:2000//DTD 10303_28_Architectural_DTD//EN".

This DTD is available in computer-interpretable form and can be found at the following URL:

http://www.mel.nist.gov/step/parts/part28e1/cd/iso_10303_28_base_arch.dtd

If there is difficulty accessing these sites contact ISO Central Secretariat or contact the ISO TC 184/SC4 Secretariat directly at: sc4sec@cme.nist.gov.

Annex E
(normative)

Universal Resource Names for EXPRESS schemas specified in ISO standards

This annex specifies the construction of a URN to be used in XML Namespace declarations in this part of ISO 10303 for the identification of an EXPRESS schema that has been registered as an International Standard.

When it is necessary to associate a URN with an EXPRESS schema that has been registered as an International Standard, the URN shall have the form:

urn:iso10303-28:*p28_binding*/iso_standard-*isonum*/[part-*pn*/]version-*vn*/*schema_name*

where:

p28_binding represents the type of XML binding as specified in this part of ISO 10303 and has one of the following values: lb , eteb, or oseb;

isonum shall be the ISO standard number under which the EXPRESS schema is registered;

pn is the part number of the part of the ISO standard under which the EXPRESS schema is registered if the ISO standard is broken down into parts;.

vn is the version number of the standard or part EXPRESS schema under which the EXPRESS schema is registered;

schema_name is the EXPRESS schema name with the first letter uppercase and the remaining ones lowercase.

EXAMPLE - The following example illustrates what the XML namespace for EXPRESS schema AP203 (ISO 10303-203) would be in an OSEB binding representation.

```
<uos xmlns:a203="urn:iso10303-28:oseb/iso_standard-10303/part-
203/version-1/Config_control_design_schema">
```


Annex F
(informative)

Computer interpretable listings

This annex references a listing of the markup declaration sets specified in this part of ISO 10303. These listings are available in computer-interpretable form and can be found at the following URL:

http://www.mel.nist.gov/step/parts/part28e1/cd/

If there is difficulty accessing these sites contact ISO Central Secretariat or contact the ISO TC 184/SC4 Secretariat directly at: sc4sec@cme.nist.gov.

NOTE - The information provided in computer-interpretable form at the above URLs is informative. The information that is contained in the body of this part of ISO 10303 is normative.

Annex G
(informative)

Technical Discussions

G.1 Stricter content models for subtypes

G.1.1 Introduction

For subtype/supertype graphs with no instances of multiple supertypes, the ETEB handles EXPRESS complex entity data types that result from inter-subtype constraints in a very simple manner. (See 8.2.7.1.) The content model for the *xxx*-subtypes element is a simple choice of zero or more of the immediate subtypes of the entity, as illustrated in the following example:

EXAMPLE - For the EXPRESS declaration:

```
ENTITY vehicle
  SUPERTYPE OF (ONEOF(car, truck));
    vin : STRING;
END_ENTITY;

ENTITY car SUBTYPE OF (vehicle);
    make  : STRING;
    car_model : STRING
END_ENTITY;

ENTITY truck SUBTYPE OF (vehicle);
    capacity : INTEGER;
END_ENTITY;
```

the corresponding XML element type declarations are:

```
<!ELEMENT vehicle (vehicle.vin, vehicle-subtypes?)>
<!ATTLIST vehicle
    id ID #REQUIRED
    express_entity_name NMTOKEN #FIXED "vehicle"
    late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT car (car.make, car.car_model)>
<!ATTLIST car
    id ID #IMPLIED
    express_entity_name NMTOKEN #FIXED "car"
    late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT truck (truck.capacity)>
<!ATTLIST truck
    id ID #IMPLIED
    express_entity_name NMTOKEN #FIXED "truck"
    late-bound-element NMTOKEN #FIXED “partial_entity_instance">

<!ELEMENT vehicle-subtypes ((car | truck)+)>
```


The supertype constraints are not maintained in the XML declaration; all immediate subtypes of **vehicle** would be included in the choice list of `vehicle-subtypes`. This approach assumes that the constraints will be enforced by the processor, including the constraint that a subtype partial complex entity instance can only appear once within `vehicle-subtypes` (the cardinality operator "+" permits duplication of elements in the list).

It is possible to encode the inter-subtype constraints directly into the XML markup declaration set. Doing so is a two-step process:

— determine the set of complex entity data types according to ISO 10303-11 Annex B and create a content model containing a choice of the complex entity data types;

— factor the content model to make it deterministic.

These steps are explained in more detail below. If applied to an application of this part of ISO 10303, these requirements would replace those specified in 8.2.7.

These steps are also applicable to subtype/supertype graphs that contain multiple supertypes. The only difference is that the evaluation of the complex entity data type using ISO 10303-11 Annex B is performed on the entire subtype/supertype graph rather than on the immediate subtypes of a supertype.

XML documents that conform to the stricter markup declaration set will also conform to the markup declaration set resulting from the requirements specified in clause 8 because the former is merely a specialization of the latter.

G.1.2 Complex instance evaluation

The following explanation describes how to create a stricter content model for subtypes that structurally-encodes inter-subtype constraints.

If an entity is a supertype and the subtype/supertype graph of which it is a part contains no entities with multiple supertypes, then the following declarations will appear in the markup declaration set per clause 8:

— an element type declaration corresponding to the subtype instances of the entity; the name of this element is the name of the supertype entity appended with the string "-subtypes";

— the last particle in the content model of the supertype entity element type declaration (as specified in 8.2.5.1) is the name of the subtypes element type declaration.

To create a stricter content model, the specification of the structure of the *xxx*-subtypes content model in 8.2.5.1 can be replaced by the content model constructed according to the following algorithm. Any XML that conforms to the resulting content model can be processed under the more general constraints specified for the *xxx*-subtypes content model specified in 8.2.5.1.

The content model of the stricter *xxx*-subtypes element type declaration is constructed as follows:

(a) For the subgraph consisting of the entity and its immediate subtypes, evaluate the set of complex entity data types according to annex B of ISO 10303-11;

i. Remove from this set the complex entity data corresponding to the supertype entity in isolation (if present);

ii. Remove the supertype entity partial complex entity data from each member of the remaining set. (Thus, the remaining partial complex instances in the set are the possible subtype combinations allowed for the supertype);

(b) If the set is empty, then the *xxx*-subtype particle is not added to the content model of the supertype entity element type declaration. If the set is not empty, the content model of the *xxx*-subtypes element type declaration is a parenthesized list constructed of particles separated by the choice operator "|" where a particle is created for each member of the set in (a) as follows:

i. If the member is a single entity data type, the particle is the entity data type identifier.

ii. If the member is comprised of two or more entity data types, the particle is a comma-separated, parenthesized list containing the identifiers of the entity data types.

(c) Factor the content model of the *xxx*-subtypes element type declaration created in (b) according to the factoring algorithm presented in G.1.3.

EXAMPLE - For the EXPRESS declaration:

```
ENTITY vehicle;
    vin : STRING;
END_ENTITY;

ENTITY car SUBTYPE OF (vehicle);
    make  : STRING;
    car_model : STRING
END_ENTITY;

ENTITY truck SUBTYPE OF (vehicle);
    capacity : INTEGER;
END_ENTITY;
```

the corresponding XML element type declarations are:

```
<!ELEMENT vehicle (vehicle.vin, vehicle-subtypes?)>
<!ATTLIST vehicle
    id ID #REQUIRED
    express-name CDATA #FIXED "vehicle"
    late-bound-element NMTOKEN #FIXED "entity_instance"
    late-bound-rename CDATA #FIXED "id object_id express-name
         express_entity_name">

<!ELEMENT car (car.make, car.car_model)>
<!ATTLIST car
    id ID #REQUIRED
```



```
        express-name CDATA #FIXED "car"
        late-bound-element NMTOKEN #FIXED "partial_entity_instance"
        late-bound-rename CDATA #FIXED "id object_id express-name
             express_entity_name">

    <!ELEMENT truck (truck.capacity)>
    <!ATTLIST truck
        id ID #REQUIRED
        express-name CDATA #FIXED "truck"
        late-bound-element NMTOKEN #FIXED "partial_entity_instance"
        late-bound-rename CDATA #FIXED "id object_id express-name
             express_entity_name">
```

before factoring (steps (a) and (b)):

```
    <!ELEMENT vehicle-subtypes (car | truck | (car, truck))>
```

after factoring (step (c)):

```
    <!ELEMENT vehicle-subtypes (car, truck?) | truck))>
```

If the subtype/supertype graph does contain an entity declaration multiple supertypes, the same basic approach/procedure is used with the following exceptions:

(a) The content model constructed is that of the synthetic element type created for the complex entity data type as specified in 8.2.7.2.

(b) The evaluation of the set of complex entity data types is made on the entire subtype/supertype graph and no further subtype/supertype processing is done on the entities in the graph.

(c) Singleton complex entity data types are not removed from the set.

The construction of the content model and factorization proceed as specified above.

G.1.3 Deterministic content models

XML requires that the content particles be deterministic based on the first item. This is an extract from the XML specification:

> “For compatibility, it is required that content models in element type declarations be deterministic.
>
> SGML requires deterministic content models (it calls them "unambiguous"); XML processors built using SGML systems may flag non-deterministic content models as errors.
>
> For example, the content model ((b, c) | (b, d)) is non-deterministic, because given an initial b the parser cannot know which b in the model is being matched without looking ahead to see which element follows the b. In this case, the two references to b can be collapsed into a single reference, making the model read (b, (c | d)). An initial b now clearly matches only a single name in the content model. The parser doesn't need to look ahead to see what follows; either c or d would be accepted.”

The following presents one algorithm for defining a deterministic content model corresponding to a given subtype/supertype graph.

Step 1 Evaluate the set S of allowed entity instances for the subtype/supertype graph according to the algorithm specified in ISO 10303-11, Annex B. This set may comprise both simple and complex entity instances.

Step 2 Identify the set E of entities that occur within S.

NOTE -This will be all of or a subset of the entities declared within the subtype/supertype graph.

Step 3 Count the occurrences in S of each entity in E.

Step 4 Identify the entity that occurs most, M. In the event that two or more entities occur most often, select to be M the first entity according to the collating sequence of ISO 646).

Step 5 Divide the set S into two sub-sets: those entity instances that include M, referred to as with-M, and those that do not, without-M.

Step 6 Process the set with-M by removing M from each of the members and call the resulting set R.

Step 7 Compare R and without-M.

a) If they are the same and not empty, generate the content particle (M?, x) where x is the result of applying steps 2 to 6 of this algorithm with S=R.

b) If they are both empty, generate the content particle M.

c) If they are different and without-M is empty, generate the content particle (M, y) where y is the result of applying steps 2 to 6 of this algorithm with S=R.

d) If they are different and R is empty, generate the content particle (M | z) where z is the result of applying steps 2 to 6 of this algorithm with S=without-M.

e) If they are different and neither R nor without-M is empty, generate the content particle ((M, y) | (z)) where y is the result of applying steps 2 to 6 of this algorithm S=R and z is the result of applying steps 2 to 6 to S=without-M.

The result of the process will be a content particle that allows only the valid combinations in S:

```
((M, xxx) | (N, yyy) | (O, zzz) …)
```

where xxx, yyy and zzz are in turn complex or simple content particles and M, N and O are entity identifiers.

G.2 Data reuse options

EXPRESS does not define reusability of data, but there are two mechanisms that could be used (for reuse of data, and for constants). What is available with this part of ISO 10303 is the use of XML text entity replacements, where the value is held once in the XML document, and is referenced from several places in the body of the document. This is really a string replacement feature, and XML parsers typically expands all references to the real value instead, so this should not be used for semantic reuse of data (i.e. the reuse would not be defined with a persistent mechanism). It may however be used as a file compression mechanism.

A more permanent and semantically correct reuse of data would be achieved with specific XML elements for reuse, i.e. an element would be used to carry the value, and would be referenced wherever the value is used. This reuse does not have a risk of being resolved by XML parsers, but will persist. The Part 28 team did however decide against this mechanism since it introduces quite a bit of complexity to the markup declaration sets, and since EXPRESS does not formally define reuse of data it was considered introducing more complexity that benefit.

G.3 Use of architectures

The interested reader is referred to "Structuring XML Documents"[4].

G.4 XML Schema

At the time of the publication of this draft of this part of ISO 10303 the latest W3C XML Schema specification is the Working Draft 07 April 2000. For that reason, no use is yet made of XML Schema. It is expected that a future edition of this part of ISO 10303 will take advantage of XML Schema and will define the use of its concepts in addition to the use of XML markup declaration sets.

One approach for the use of XML Schema being investigated is to take advantage of the greater power for data type checking that comes with XML Schema for use with data marked up according to the representations specified in this edition of this part of ISO 10303. This approach maintains the upward compatibility of applications that create data based on this part of ISO 10303 but allows better validation to be performed on the created data.

Annex H
(informative)

Examples

H.1 Example schema in EXPRESS

```
(* This is an example model used within the STEPWISE project (Esprit
project 25110, see www.stepwise.org) to illustrate different aspects of
EXPRESS. It has been modified to have examples of many different EXPRESS
constructs. It is not intended to be an example of good modelling style. *)

SCHEMA mr_jones_garden;

REFERENCE FROM support_items;

REFERENCE FROM mr_smiths_garden (garden AS mr_smiths_garden);

USE FROM mr_smiths_garden (plant AS mr_smiths_plant);

CONSTANT
   water_volume_per_fish: INTEGER := 1;
   water_volume_per_amphibian: INTEGER := 2;
   water_volume_per_ornament: INTEGER := 3;
END_CONSTANT;

(*
** The description of the domain states that all the flowers in Mr
** Jones's garden are either coloured red, yellow or white.  For this
** reason, "flower_colour" has been modelled as an enumerated type.
*)
TYPE flower_colour
   = ENUMERATION OF (red, yellow, white);
END_TYPE;

TYPE fish_colour
   = ENUMERATION OF (yellow, black, silver, blue);
END_TYPE;

TYPE multi_coloured_fish_colour
  = SET [2:4] OF fish_colour;
END_TYPE;

TYPE observed_fish_colour
  = SELECT (fish_colour, multi_coloured_fish_colour);
END_TYPE;

TYPE pond_ornament
   = ENUMERATION OF (waterfall, fountain, bridge);
END_TYPE;

TYPE fish_type
```



```
   = ENUMERATION OF (koi, goldfish);
END_TYPE;

TYPE aquatic_plant_types
   = ENUMERATION OF (lilly, weed, lotus);
END_TYPE;

TYPE amphibian
   = ENUMERATION OF (frog, newt);
END_TYPE;


ENTITY water_treatment_system
   ABSTRACT SUPERTYPE OF (pumping_system AND filtration_system);
   capacity: positive_integer;
END_ENTITY;

ENTITY filtration_system
   SUBTYPE OF (water_treatment_system);
   filtration_efficiency: efficiency;
   number_of_filters: positive_integer;
END_ENTITY;

ENTITY pumping_system
   SUBTYPE OF (water_treatment_system);
   pumping_efficiency: efficiency;
   number_of_pumps: positive_integer;
END_ENTITY;

ENTITY pond
   ABSTRACT SUPERTYPE OF
      (ONEOF (fish_pond, amphibian_pond) ANDOR ornamental_pond);
   maintained_by: OPTIONAL water_treatment_system;
   plants: SET[1:?] OF aquatic_plant;
   water_volume: positive_integer;
   water_ph: ph;
END_ENTITY;

ENTITY livestock_container
   ABSTRACT SUPERTYPE OF (ONEOF (fish_pond, amphibian_pond));
   livestock_count : OPTIONAL INTEGER;
   maintained_by: OPTIONAL STRING;
END_ENTITY;

ENTITY fish_pond
   SUBTYPE OF (pond, livestock_container);
   fish: SET [1:?] OF fish_type;
   fish_colours : SET OF observed_fish_colour;
DERIVE
   SELF\pond.water_volume: positive_integer := water_volume_per_fish *
   SIZEOF(fish);
END_ENTITY;

ENTITY amphibian_pond
   SUBTYPE OF (pond, livestock_container);
```



```
   amphibians: SET [1:?] OF amphibian;
DERIVE
   SELF\pond.water_volume: positive_integer :=
   water_volume_per_amphibian * SIZEOF(amphibians);
END_ENTITY;

ENTITY ornamental_pond
   SUBTYPE OF (pond);
   ornaments: SET [1:?] OF pond_ornament;
DERIVE
   SELF\pond.water_volume: positive_integer :=
   water_volume_per_ornament * SIZEOF(ornaments);
END_ENTITY;


(*
** The "garden" is the main entity of the model.  All information
within
** the description which is not related to the garden and its contents
** has been ignored e.g. Mr Jones himself and the local horticultural
** society.

** The set aggregate has been used for the relationship between a
** "garden" and the "beds" in it.  List and array were rejected as
** there
** is no indication within the description of ordering.  Bag was
** rejected
** as a "bed" cannot occur in the "garden" more than once.
*)
ENTITY garden;
   has_pond                   : fish_pond;
   has_greenhouse             : greenhouse;
   climatic_temperature_range : temperature_range;
   has_beds                   : SET [5 : 5] OF bed;
   neighbours_garden          : mr_smiths_garden;
END_ENTITY;

(*
** The "only_one_garden" rule ensures that the model only allows one
** garden to exist within the domain.
*)
RULE only_one_garden FOR (garden);
WHERE
   mr_jones_has_one_garden :
      SIZEOF (garden) = 1;
END_RULE;

(*
** A "greenhouse" is a greenhouse within Mr Jones's "garden".  It
** contains a set of at least one "greenhouse_plants".  The
"greenhouse"
** also has an associated "temperature".

** The "greenhouse" is constrained to associated with one and only one
** "garden".  Since there is "only_one_garden" in the domain, there is
```



```
** no
** need to explicitly constrain the model to only allow one
** "greenhouse".
*)
ENTITY greenhouse;
   enforced_temperature_range  : temperature_range;
   holds_plants                : SET [1 : ?] OF greenhouse_plant;
INVERSE
   the_garden                  : garden FOR has_greenhouse;
END_ENTITY;

(*
** A "bed" is a flower bed within Mr Jones's "garden".  It contains a
set
** of at least one "outdoors_plant".  It also has an associated "ph"
** value.

** Each "bed" is constrained to associated with one and only one
** "garden".  Since there is "only_one_garden" in the domain, and the
** "garden" is related to exactly five flower "beds" there is no
** need to explicitly constrain the model to only allow only five
** "beds".

** The acidity of each "bed" is different i.e. unique.
*)
ENTITY bed;
   acidity                     : ph;
   holds_plants                : SET [1 : ?] OF outdoors_plant;
   planting_plan  : OPTIONAL ARRAY [1:5] OF ARRAY [1:10] OF OPTIONAL
outdoors_plant;
   flowering_order             : OPTIONAL LIST OF outdoors_plant;
   plants_ready_for_planting   : OPTIONAL BAG [1:50] OF outdoors_plant;
INVERSE
   the_garden                  : garden FOR has_beds;
UNIQUE
   every_bed_has_a_different_acidity :
      acidity;
END_ENTITY;

(*
** A "plant" is a flowering plant within either the "greenhouse" or a
** "bed".  These kinds of "plant" are distinct and are represented by
the
** two subtypes of "plant", namely "greenhouse_plant" and
** "outdoors_plant".

** All "plants" have an associated "flower_colour".  They must have a
** unique Latin "plant_name".  In addition they may have one or more
** English "plant_names".  The English "plant_names" are not unique
** since
** several different species of "plants" may have the same English
** "plant_name".

** The description states that none of the "plants" inside the
** "greenhouse" can survive outside it.  This, together with the fact
```



```
** that Mr Jones is a keen gardener, has been taken to imply that the
** "plants" he places inside the "greenhouse" can survive there.
** Similarly, the "outdoors_plants" must be able to survive in the
** "garden".  Therefore, the "temperature_range" which a "plant" can
** survive in is relevant to all "plants".
*)
ENTITY plant
   ABSTRACT SUPERTYPE OF
      (ONEOF (greenhouse_plant, outdoors_plant, aquatic_plant));
   colour                      : flower_colour;
   latin_name                  : plant_name;
   english_names               : OPTIONAL SET [1 : ?] OF plant_name;
   survival_temperature_range  : temperature_range;
UNIQUE
   the_latin_name_of_a_plant_species_is_unique :
      latin_name;
END_ENTITY;

(*
** A "greenhouse_plant" is a "plant" which can only survive within the
** "greenhouse".  In order to ensure that the "greenhouse_plant" may
grow
** within the "greenhouse" it has an associated "temperature" range.

** A "greenhouse_plant" must be inside the "greenhouse".

** A "greenhouse_plant" must be able to survive in the
** "temperature_range" of the "greenhouse".

** A "greenhouse_plant" cannot survive in the "temperature_range" of
** the
** "garden".  Note that this does not prevent the "temperature_ranges"
** of
** the "garden" and "greenhouse" from overlapping.
*)
ENTITY greenhouse_plant
   SUBTYPE OF (plant);
INVERSE
   the_greenhouse              : greenhouse FOR holds_plants;
WHERE
   r1 :
      (* A greenhouse plant can survive in the greenhouse temperature
** *)
       is_sub_range (the_greenhouse.enforced_temperature_range,
                  SELF\plant.survival_temperature_range);

   r2 :
      (* A greenhouse plan cannot survive in the garden temperature *)
      NOT is_sub_range
      (the_greenhouse.the_garden.climatic_temperature_range,
                     SELF\plant.survival_temperature_range);
END_ENTITY;

(*
** An "outdoors_plant" is a "plant" which can survive outside of the
```



```
** "greenhouse".  It must be grown in a flower "bed".  In order to
ensure
** that the "outdoors_plant" can cope with the acidity of the "bed" it
** has an associated "ph_range".

** The "ph_range" of the "outdoors_plant" must include the "ph" of the
** "bed".
*)
ENTITY outdoors_plant
   SUBTYPE OF (plant);
   survival_ph_range           : ph_range;
INVERSE
   the_beds                    : SET [1 : ?] OF bed FOR holds_plants;
WHERE
   r1 :
      (* The ph range of the outdoors plant must include the ph value
** of the
      bed *)
    QUERY (b <* the_beds |
        value_is_within_range (b.acidity, survival_ph_range))
        = the_beds;

   r2 :
      (* An outdoors plant can survive in the garden temperature *)
      is_sub_range (the_beds
      [1].the_garden.climatic_temperature_range,
                  SELF\plant.survival_temperature_range);
END_ENTITY;

ENTITY aquatic_plant
   SUBTYPE OF (plant);
   aquatic_plant_type: aquatic_plant_types;
   oxygen_volumetric_requirement: positive_integer;
   aquatic_plant_size: positive_integer;
END_ENTITY;


(*
** This function checks that a real value lies within the specified
** "real_value_range".
*)
FUNCTION value_is_within_range (v : REAL;
                                r : real_value_range) : BOOLEAN;
   RETURN ( (v >= r.minimum_value) AND (v <= r.maximum_value));
END_FUNCTION;

(*
** This function checks that one "real_value_range" is completely
** contained within another.
*)
FUNCTION is_sub_range (r1,
                       r2 : real_value_range) : BOOLEAN;
   RETURN (value_is_within_range (r1.minimum_value, r2) AND
          value_is_within_range (r1.maximum_value, r2));
END_FUNCTION;
```



```
END_SCHEMA;


(* ============================ S C H E M A  ====================== *)

SCHEMA support_items;

(*
** The appropriate base type to use for "temperature" is not clear
from
** the description.  However, given that Mr Jones takes great care of
** "temperature" values it was decided to model it as a real value.
*)
TYPE temperature
    = REAL;
END_TYPE;

(*
** Both "temperatures" and "ph" values for a plant are within a
** "real_value_range".
*)
ENTITY real_value_range
   ABSTRACT SUPERTYPE OF
      (ONEOF (temperature_range, ph_range, efficiency_range));
   minimum_value               : REAL;
   maximum_value               : REAL;
WHERE
   the_values_must_be_sensible :
      maximum_value >= minimum_value;
END_ENTITY;

(*
** A "plant" is able to survive within a "temperature_range".  The
** "garden" will throughout the year have a "temperature_range" which
the
** "outdoors_plants" must be able to survive in.  Similarly, the
** "greenhouse" has a "temperature_range" which the
"greenhouse_plants"
** can survive in.
*)
ENTITY temperature_range
   SUBTYPE OF (real_value_range);
   SELF\real_value_range.minimum_value : temperature;
   SELF\real_value_range.maximum_value : temperature;
END_ENTITY;

(*
** The description mentions both Latin names and English names for
** "plants".  These two `types' of "plant_name" were considered to
have
** sufficient in common to justify their modelling by a single defined
** type.

** It is not clear if some constraints should be specified for
```



```
** "plant_name".  For example, it may sensible to ensure that it is
** not
** empty and that it is alphabetic.
*)
TYPE plant_name
    = STRING;
END_TYPE;

(*
** The acidity of the "beds" and the range acceptable to
** "outdoors_plants" has been modelled by the defined type "ph".  It
is
** not clear which base type should be used for "ph"; however, given
that
** Mr Jones takes pride in compatibility of values, it was decided to
use
** the real type.

** A "ph" value is constrained to being between zero and fourteen.  In
** reality, it is highly unlikely that Mr Jones would ever use "ph"
** values near the extremes.  However, there is no indication of more
** realistic limits within the domain description.
*)
TYPE ph
   = REAL;
WHERE
   the_ph_is_between_0_and_14 :
     {0 <= SELF <= 14};
END_TYPE;

(*
** An "outdoors_plant", indeed any "plant", can survive a range of
"ph"
** values.
*)
ENTITY ph_range
   SUBTYPE OF (real_value_range);
   SELF\real_value_range.minimum_value : ph;
   SELF\real_value_range.maximum_value : ph;
END_ENTITY;

TYPE positive_integer = INTEGER;
WHERE
   positive_only: SELF > 0;
END_TYPE;

ENTITY efficiency_range
   SUBTYPE OF (real_value_range);
   SELF\real_value_range.minimum_value : efficiency_measure;
   SELF\real_value_range.maximum_value : efficiency_measure;
END_ENTITY;

TYPE percentage = REAL;
WHERE
   between_0_and_100: {0.0 <= SELF <= 100.0};
```



```
END_TYPE;

TYPE fractional_value = REAL;
WHERE
   between_0_and_1: {0.0 <= SELF <= 1.0};
END_TYPE;

TYPE efficiency_measure = SELECT (percentage, fractional_value);
END_TYPE;

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

END_SCHEMA;


(* ============================ S C H E M A  ====================== *)


SCHEMA mr_smiths_garden;

(*
Mr Smith's garden is a simple one and has been set up to illustrate
REFERENCE and USE from the schema mr_jones_garden.

Some of the entities in this schema have the same name as in
mr_jones_garden to illustrate name clash handling between schemas.
*)
ENTITY garden;
   has_bed                   : bed;
END_ENTITY;

ENTITY bed;
  description : STRING;
INVERSE
   the_garden                 : garden FOR has_bed;
END_ENTITY;

(*
As plant is USEd by mr_jones_garden schema, it is imported but its subtypes
will
 not be visible in that schema so the only possible combination is plant on
its own.
*)
ENTITY plant
   SUPERTYPE OF (flower_plant ANDOR vegetable_plant);
   name                      : STRING;
END_ENTITY;

ENTITY flower_plant
   SUBTYPE OF (plant);
   flower_colour : STRING;
END_ENTITY;

ENTITY vegetable_plant
```



```
   SUBTYPE OF (plant);
   vegetable_type : STRING;
END_ENTITY;

END_SCHEMA;
```

H.2 Example schema in XML

The example schema is presented in XML as a representation category of "SCHEMA" XML document that uses the markup declaration set specified in Annex C for the representation..

```
<?xml version="1.0"?>
<!DOCTYPE iso_10303_28 SYSTEM "express-part28xds-v14-exp.dtd">
<iso_10303_28 representation_category=”SCHEMA_DECL”>
 <express_schema>
  <schema_decl>
   <embedded_remark>This is an example model used within the STEPWISE
project (Esprit project 25110, see www.stepwise.org) to illustrate
different aspects of EXPRESS. It has been modified to have examples of
many different EXPRESS constructs. It is not intended to be an example of
good modelling  style.</embedded_remark>
   <schema_id>mr_jones_garden</schema_id>
   <interface_specification_block>
    <reference_from>
     <schema_ref>support_items</schema_ref><import_all/>
    </reference_from>
    <reference_from>
     <schema_ref>mr_smiths_garden</schema_ref>
     <entity_import>
      <entity_id>mr_smiths_garden</entity_id>
      <entity_ref>garden</entity_ref>
     </entity_import>
    </reference_from>
    <use_from>
     <schema_ref>mr_smiths_garden</schema_ref>
     <entity_import>
      <entity_id>mr_smiths_plant</entity_id>
      <entity_ref>plant</entity_ref>
     </entity_import>
    </use_from>
   </interface_specification_block>

   <constant_block>
    <constant_decl>
     <constant_id>water_volume_per_fish</constant_id>
     <base_type><integer/>
     </base_type>
     <integer_literal>1</integer_literal>
    </constant_decl>
    <constant_decl>
     <constant_id>water_volume_per_amphibian</constant_id>
     <base_type><integer/>
```



```
      </base_type>
      <integer_literal>2</integer_literal>
    </constant_decl>
    <constant_decl>
      <constant_id>water_volume_per_ornament</constant_id>
      <base_type><integer/>
      </base_type>
      <integer_literal>3</integer_literal>
    </constant_decl>
  </constant_block>
  <type_decl>
    <embedded_remark>** The description of the domain states that all the
      flowers in Mr ** Jones's garden are either coloured red, yellow or
white. For this ** reason, "flower_colour" has been modelled as an
enumerated type.
      </embedded_remark>
    <type_id>flower_colour</type_id>
    <underlying_type>
      <enumeration>
        <enumeration_id>red</enumeration_id>
        <enumeration_id>yellow</enumeration_id>
        <enumeration_id>white</enumeration_id>
      </enumeration>
    </underlying_type>
  </type_decl>
  <type_decl>
    <type_id>fish_colour</type_id>
    <underlying_type>
      <enumeration>
        <enumeration_id>yellow</enumeration_id>
        <enumeration_id>black</enumeration_id>
        <enumeration_id>silver</enumeration_id>
        <enumeration_id>blue</enumeration_id>
      </enumeration>
    </underlying_type>
  </type_decl>
  <type_decl>
    <type_id>multi_coloured_fish_colour</type_id>
    <underlying_type>
      <set_type>
        <bound_spec>
          <lower_bound>
            <integer_literal>2</integer_literal>
          </lower_bound>
          <upper_bound>
            <integer_literal>4</integer_literal>
          </upper_bound>
        </bound_spec>
        <base_type>
          <type_ref>fish_colour</type_ref>
        </base_type>
      </set_type>
    </underlying_type>
  </type_decl>
  <type_decl>
```



```
  <type_id>observed_fish_colour</type_id>
  <underlying_type>
   <select>
    <type_ref>fish_colour</type_ref>
    <type_ref>multi_coloured_fish_colour</type_ref>
   </select>
  </underlying_type>
 </type_decl>
 <type_decl>
  <type_id>pond_ornament</type_id>
  <underlying_type>
   <enumeration>
    <enumeration_id>waterfall</enumeration_id>
    <enumeration_id>fountain</enumeration_id>
    <enumeration_id>bridge</enumeration_id>
   </enumeration>
  </underlying_type>
 </type_decl>
 <type_decl>
  <type_id>fish_type</type_id>
  <underlying_type>
   <enumeration>
    <enumeration_id>koi</enumeration_id>
    <enumeration_id>goldfish</enumeration_id>
   </enumeration>
  </underlying_type>
 </type_decl>
 <type_decl>
  <type_id>aquatic_plant_types</type_id>
  <underlying_type>
   <enumeration>
    <enumeration_id>lilly</enumeration_id>
    <enumeration_id>weed</enumeration_id>
    <enumeration_id>lotus</enumeration_id>
   </enumeration>
  </underlying_type>
 </type_decl>
 <type_decl>
  <type_id>amphibian</type_id>
  <underlying_type>
   <enumeration>
    <enumeration_id>frog</enumeration_id>
    <enumeration_id>newt</enumeration_id>
   </enumeration>
  </underlying_type>
 </type_decl>
 <entity_decl>
  <entity_id>water_treatment_system</entity_id>
  <abstract_supertype>
   <supertype_and>
    <entity_ref>pumping_system</entity_ref>
    <entity_ref>filtration_system</entity_ref>
   </supertype_and>
  </abstract_supertype>
  <explicit_attr_block>
```



```
    <explicit_attr>
     <attribute_id>capacity</attribute_id>
     <base_type>
      <type_ref>positive_integer</type_ref>
     </base_type>
    </explicit_attr>
   </explicit_attr_block>
  </entity_decl>
  <entity_decl>
   <entity_id>filtration_system</entity_id>
   <subtype_of>
    <entity_ref>water_treatment_system</entity_ref>
   </subtype_of>
   <explicit_attr_block>
    <explicit_attr>
     <attribute_id>filtration_efficiency</attribute_id>
     <base_type>
      <type_ref>efficiency</type_ref>
     </base_type>
    </explicit_attr>
    <explicit_attr>
     <attribute_id>number_of_filters</attribute_id>
     <base_type>
      <type_ref>positive_integer</type_ref>
     </base_type>
    </explicit_attr>
   </explicit_attr_block>
  </entity_decl>
  <entity_decl>
   <entity_id>pumping_system</entity_id>
   <subtype_of>
    <entity_ref>water_treatment_system</entity_ref>
   </subtype_of>
   <explicit_attr_block>
    <explicit_attr>
     <attribute_id>pumping_efficiency</attribute_id>
     <base_type>
      <type_ref>efficiency</type_ref>
     </base_type>
    </explicit_attr>
    <explicit_attr>
     <attribute_id>number_of_pumps</attribute_id>
     <base_type>
      <type_ref>positive_integer</type_ref>
     </base_type>
    </explicit_attr>
   </explicit_attr_block>
  </entity_decl>
  <entity_decl>
   <entity_id>pond</entity_id>
   <abstract_supertype>
    <supertype_and_or>
     <supertype_one_of>
      <entity_ref>fish_pond</entity_ref>
      <entity_ref>amphibian_pond</entity_ref>
```



```
      </supertype_one_of>
      <entity_ref>ornamental_pond</entity_ref>
    </supertype_and_or>
  </abstract_supertype>
  <explicit_attr_block>
    <explicit_attr>
      <attribute_id>maintained_by</attribute_id><optional/>
      <base_type>
        <entity_ref>water_treatment_system</entity_ref>
      </base_type>
    </explicit_attr>
    <explicit_attr>
      <attribute_id>plants</attribute_id>
      <base_type>
        <set_type>
          <bound_spec>
            <lower_bound>
              <integer_literal>1</integer_literal>
            </lower_bound>
            <upper_bound><unset/>
            </upper_bound>
          </bound_spec>
          <base_type>
            <entity_ref>aquatic_plant</entity_ref>
          </base_type>
        </set_type>
      </base_type>
    </explicit_attr>
    <explicit_attr>
      <attribute_id>water_volume</attribute_id>
      <base_type>
        <type_ref>positive_integer</type_ref>
      </base_type>
    </explicit_attr>
    <explicit_attr>
      <attribute_id>water_ph</attribute_id>
      <base_type>
        <type_ref>ph</type_ref>
      </base_type>
    </explicit_attr>
  </explicit_attr_block>
</entity_decl>
<entity_decl>
  <entity_id>livestock_container</entity_id>
  <abstract_supertype>
    <supertype_one_of>
      <entity_ref>fish_pond</entity_ref>
      <entity_ref>amphibian_pond</entity_ref>
    </supertype_one_of>
  </abstract_supertype>
  <explicit_attr_block>
    <explicit_attr>
      <attribute_id>livestock_count</attribute_id><optional/>
      <base_type><integer/>
      </base_type>
```



```
    </explicit_attr>
    <explicit_attr>
      <attribute_id>maintained_by</attribute_id><optional/>
      <base_type>
        <string>
        </string>
      </base_type>
    </explicit_attr>
  </explicit_attr_block>
</entity_decl>
<entity_decl>
  <entity_id>fish_pond</entity_id>
  <subtype_of>
    <entity_ref>pond</entity_ref>
    <entity_ref>livestock_container</entity_ref>
  </subtype_of>
  <explicit_attr_block>
    <explicit_attr>
      <attribute_id>fish</attribute_id>
      <base_type>
        <set_type>
          <bound_spec>
            <lower_bound>
              <integer_literal>1</integer_literal>
            </lower_bound>
            <upper_bound><unset/>
            </upper_bound>
          </bound_spec>
          <base_type>
            <type_ref>fish_type</type_ref>
          </base_type>
        </set_type>
      </base_type>
    </explicit_attr>
    <explicit_attr>
      <attribute_id>fish_colours</attribute_id>
      <base_type>
        <set_type>
          <base_type>
            <type_ref>observed_fish_colour</type_ref>
          </base_type>
        </set_type>
      </base_type>
    </explicit_attr>
  </explicit_attr_block>
  <derive_clause>
    <derived_attr>
      <qualified_attribute>
        <entity_ref>pond</entity_ref>
        <attribute_ref>water_volume</attribute_ref>
      </qualified_attribute>
      <base_type>
        <type_ref>positive_integer</type_ref>
      </base_type>
      <term><multiply/>
```



```
      <constant_ref>water_volume_per_fish</constant_ref>
      <function_call><sizeof/>
       <attribute_ref>fish</attribute_ref>
      </function_call>
     </term>
    </derived_attr>
  </derive_clause>
</entity_decl>
<entity_decl>
  <entity_id>amphibian_pond</entity_id>
  <subtype_of>
   <entity_ref>pond</entity_ref>
   <entity_ref>livestock_container</entity_ref>
  </subtype_of>
  <explicit_attr_block>
   <explicit_attr>
    <attribute_id>amphibians</attribute_id>
    <base_type>
     <set_type>
      <bound_spec>
       <lower_bound>
        <integer_literal>1</integer_literal>
       </lower_bound>
       <upper_bound><unset/>
       </upper_bound>
      </bound_spec>
      <base_type>
       <type_ref>amphibian</type_ref>
      </base_type>
     </set_type>
    </base_type>
   </explicit_attr>
  </explicit_attr_block>
  <derive_clause>
   <derived_attr>
    <qualified_attribute>
     <entity_ref>pond</entity_ref>
     <attribute_ref>water_volume</attribute_ref>
    </qualified_attribute>
    <base_type>
     <type_ref>positive_integer</type_ref>
    </base_type>
    <term><multiply/>
     <constant_ref>water_volume_per_amphibian</constant_ref>
     <function_call><sizeof/>
      <attribute_ref>amphibians</attribute_ref>
     </function_call>
    </term>
   </derived_attr>
  </derive_clause>
</entity_decl>
<entity_decl>
  <entity_id>ornamental_pond</entity_id>
  <subtype_of>
   <entity_ref>pond</entity_ref>
```



```
    </subtype_of>
    <explicit_attr_block>
     <explicit_attr>
      <attribute_id>ornaments</attribute_id>
      <base_type>
       <set_type>
        <bound_spec>
         <lower_bound>
          <integer_literal>1</integer_literal>
         </lower_bound>
         <upper_bound><unset/>
         </upper_bound>
        </bound_spec>
        <base_type>
         <type_ref>pond_ornament</type_ref>
        </base_type>
       </set_type>
      </base_type>
     </explicit_attr>
    </explicit_attr_block>
    <derive_clause>
     <derived_attr>
      <qualified_attribute>
       <entity_ref>pond</entity_ref>
       <attribute_ref>water_volume</attribute_ref>
      </qualified_attribute>
      <base_type>
       <type_ref>positive_integer</type_ref>
      </base_type>
      <term><multiply/>
       <constant_ref>water_volume_per_ornament</constant_ref>
       <function_call><sizeof/>
        <attribute_ref>ornaments</attribute_ref>
       </function_call>
      </term>
     </derived_attr>
    </derive_clause>
   </entity_decl>
   <entity_decl>
    <embedded_remark>** The "garden" is the main entity of the model. All
     information within ** the description which is not related to the
garden and
     its contents ** has been ignored e.g. Mr Jones himself and the local
     horticultural ** society. ** The set aggregate has been used for the
     relationship between a ** "garden" and the "beds" in it. List and
array were
     rejected as ** there ** is no indication within the description of
ordering.
     Bag was ** rejected ** as a "bed" cannot occur in the "garden" more
than once.
     </embedded_remark>
    <entity_id>garden</entity_id>
    <explicit_attr_block>
     <explicit_attr>
      <attribute_id>has_pond</attribute_id>
```



```
        <base_type>
          <entity_ref>fish_pond</entity_ref>
        </base_type>
      </explicit_attr>
      <explicit_attr>
        <attribute_id>has_greenhouse</attribute_id>
        <base_type>
          <entity_ref>greenhouse</entity_ref>
        </base_type>
      </explicit_attr>
      <explicit_attr>
        <attribute_id>climatic_temperature_range</attribute_id>
        <base_type>
          <entity_ref>temperature_range</entity_ref>
        </base_type>
      </explicit_attr>
      <explicit_attr>
        <attribute_id>has_beds</attribute_id>
        <base_type>
          <set_type>
            <bound_spec>
              <lower_bound>
                <integer_literal>5</integer_literal>
              </lower_bound>
              <upper_bound>
                <integer_literal>5</integer_literal>
              </upper_bound>
            </bound_spec>
            <base_type>
              <entity_ref>bed</entity_ref>
            </base_type>
          </set_type>
        </base_type>
      </explicit_attr>
      <explicit_attr>
        <attribute_id>neighbours_garden</attribute_id>
        <base_type>
          <entity_ref>mr_smiths_garden</entity_ref>
        </base_type>
      </explicit_attr>
    </explicit_attr_block>
  </entity_decl>
  <rule_decl>
    <embedded_remark>** The "only_one_garden" rule ensures that the model
      only allows one ** garden to exist within the domain.
</embedded_remark>
    <rule_id>only_one_garden</rule_id>
    <applies_to_entities>
      <entity_ref>garden</entity_ref>
    </applies_to_entities>
    <where_clause>
      <domain_rule>
        <label>mr_jones_has_one_garden</label>
        <logical_expression>
          <relation_expression><equal/>
```



```
          <function_call><sizeof/>
            <population>
              <entity_ref>garden</entity_ref>
            </population>
          </function_call>
          <integer_literal>1</integer_literal>
        </relation_expression>
      </logical_expression>
    </domain_rule>
  </where_clause>
 </rule_decl>
 <entity_decl>
  <embedded_remark>** A "greenhouse" is a greenhouse within Mr Jones's
   "garden". It ** contains a set of at least one "greenhouse_plants".
The
   "greenhouse" ** also has an associated "temperature". ** The
"greenhouse" is
   constrained to associated with one and only one ** "garden". Since
there is
   "only_one_garden" in the domain, there is ** no ** need to explicitly
constrain
   the model to only allow one ** "greenhouse". </embedded_remark>
  <entity_id>greenhouse</entity_id>
  <explicit_attr_block>
   <explicit_attr>
    <attribute_id>enforced_temperature_range</attribute_id>
    <base_type>
     <entity_ref>temperature_range</entity_ref>
    </base_type>
   </explicit_attr>
   <explicit_attr>
    <attribute_id>holds_plants</attribute_id>
    <base_type>
     <set_type>
      <bound_spec>
       <lower_bound>
        <integer_literal>1</integer_literal>
       </lower_bound>
       <upper_bound><unset/>
       </upper_bound>
      </bound_spec>
      <base_type>
       <entity_ref>greenhouse_plant</entity_ref>
      </base_type>
     </set_type>
    </base_type>
   </explicit_attr>
  </explicit_attr_block>
  <inverse_clause>
   <inverse_attr>
    <attribute_id>the_garden</attribute_id>
    <entity_ref>garden</entity_ref>
    <attribute_ref>has_greenhouse</attribute_ref>
   </inverse_attr>
  </inverse_clause>
```



```
   </entity_decl>
   <entity_decl>
    <embedded_remark>** A "bed" is a flower bed within Mr Jones's
"garden".
     It contains a set ** of at least one "outdoors_plant". It also has an
     associated "ph" ** value. ** Each "bed" is constrained to associated
with one
     and only one ** "garden". Since there is "only_one_garden" in the
domain, and
     the ** "garden" is related to exactly five flower "beds" there is no
** need to
     explicitly constrain the model to only allow only five ** "beds". **
The
     acidity of each "bed" is different i.e. unique. </embedded_remark>
    <entity_id>bed</entity_id>
    <explicit_attr_block>
     <explicit_attr>
      <attribute_id>acidity</attribute_id>
      <base_type>
       <type_ref>ph</type_ref>
      </base_type>
     </explicit_attr>
     <explicit_attr>
      <attribute_id>holds_plants</attribute_id>
      <base_type>
       <set_type>
        <bound_spec>
         <lower_bound>
          <integer_literal>1</integer_literal>
         </lower_bound>
         <upper_bound><unset/>
         </upper_bound>
        </bound_spec>
        <base_type>
         <entity_ref>outdoors_plant</entity_ref>
        </base_type>
       </set_type>
      </base_type>
     </explicit_attr>
     <explicit_attr>
      <attribute_id>planting_plan</attribute_id><optional/>
      <base_type>
       <array_type>
        <index_spec>
         <low_index>
          <integer_literal>1</integer_literal>
         </low_index>
         <high_index>
          <integer_literal>5</integer_literal>
         </high_index>
        </index_spec>
        <base_type>
         <array_type>
          <index_spec>
           <low_index>
```



```
              <integer_literal>1</integer_literal>
             </low_index>
             <high_index>
              <integer_literal>10</integer_literal>
             </high_index>
            </index_spec>
            <base_type>
             <entity_ref>outdoors_plant</entity_ref>
            </base_type><optional/>
           </array_type>
         </base_type>
       </array_type>
     </base_type>
   </explicit_attr>
   <explicit_attr>
     <attribute_id>flowering_order</attribute_id><optional/>
     <base_type>
       <list_type>
         <base_type>
          <entity_ref>outdoors_plant</entity_ref>
         </base_type>
       </list_type>
     </base_type>
   </explicit_attr>
   <explicit_attr>
     <attribute_id>plants_ready_for_planting</attribute_id><optional/>
     <base_type>
       <bag_type>
         <bound_spec>
          <lower_bound>
           <integer_literal>1</integer_literal>
          </lower_bound>
          <upper_bound>
           <integer_literal>50</integer_literal>
          </upper_bound>
         </bound_spec>
         <base_type>
          <entity_ref>outdoors_plant</entity_ref>
         </base_type>
       </bag_type>
     </base_type>
   </explicit_attr>
 </explicit_attr_block>
 <inverse_clause>
   <inverse_attr>
     <attribute_id>the_garden</attribute_id>
     <entity_ref>garden</entity_ref>
     <attribute_ref>has_beds</attribute_ref>
   </inverse_attr>
 </inverse_clause>
 <unique_clause>
   <unique_rule>
     <label>every_bed_has_a_different_acidity</label>
     <attribute_ref>acidity</attribute_ref>
   </unique_rule>
```



```
    </unique_clause>
   </entity_decl>
   <entity_decl>
    <embedded_remark>** A "plant" is a flowering plant within either the
     "greenhouse" or a ** "bed". These kinds of "plant" are distinct and
are
     represented by the ** two subtypes of "plant", namely
"greenhouse_plant" and **
     "outdoors_plant". ** All "plants" have an associated "flower_colour".
They must
     have a ** unique Latin "plant_name". In addition they may have one or
more **
     English "plant_names". The English "plant_names" are not unique **
since **
     several different species of "plants" may have the same English **
     "plant_name". ** The description states that none of the "plants"
inside the **
     "greenhouse" can survive outside it. This, together with the fact **
that Mr
     Jones is a keen gardener, has been taken to imply that the **
"plants" he
     places inside the "greenhouse" can survive there. ** Similarly, the
     "outdoors_plants" must be able to survive in the ** "garden".
Therefore, the
     "temperature_range" which a "plant" can ** survive in is relevant to
all
     "plants".</embedded_remark>
    <entity_id>plant</entity_id>
    <abstract_supertype>
     <supertype_one_of>
      <entity_ref>greenhouse_plant</entity_ref>
      <entity_ref>outdoors_plant</entity_ref>
      <entity_ref>aquatic_plant</entity_ref>
     </supertype_one_of>
    </abstract_supertype>
    <explicit_attr_block>
     <explicit_attr>
      <attribute_id>colour</attribute_id>
      <base_type>
       <type_ref>flower_colour</type_ref>
      </base_type>
     </explicit_attr>
     <explicit_attr>
      <attribute_id>latin_name</attribute_id>
      <base_type>
       <type_ref>plant_name</type_ref>
      </base_type>
     </explicit_attr>
     <explicit_attr>
      <attribute_id>english_names</attribute_id><optional/>
      <base_type>
       <set_type>
        <bound_spec>
         <lower_bound>
          <integer_literal>1</integer_literal>
```



```
            </lower_bound>
            <upper_bound><unset/>
            </upper_bound>
           </bound_spec>
           <base_type>
            <type_ref>plant_name</type_ref>
           </base_type>
         </set_type>
        </base_type>
       </explicit_attr>
       <explicit_attr>
        <attribute_id>survival_temperature_range</attribute_id>
        <base_type>
         <entity_ref>temperature_range</entity_ref>
        </base_type>
       </explicit_attr>
     </explicit_attr_block>
     <unique_clause>
       <unique_rule>
        <label>the_latin_name_of_a_plant_is_unique</label>
        <attribute_ref>latin_name</attribute_ref>
       </unique_rule>
     </unique_clause>
   </entity_decl>
   <entity_decl>
     <embedded_remark>** A "greenhouse_plant" is a "plant" which can only
       survive within the ** "greenhouse". In order to ensure that the
       "greenhouse_plant" may grow ** within the "greenhouse" it has an
associated
       "temperature" range. ** A "greenhouse_plant" must be inside the
"greenhouse".
       ** A "greenhouse_plant" must be able to survive in the **
"temperature_range"
       of the "greenhouse". ** A "greenhouse_plant" cannot survive in the
       "temperature_range" of ** the ** "garden". Note that this does not
prevent the
       "temperature_ranges" ** of ** the "garden" and "greenhouse" from
overlapping.
       *) </embedded_remark>
     <entity_id>greenhouse_plant</entity_id>
     <subtype_of>
       <entity_ref>plant</entity_ref>
     </subtype_of>
     <inverse_clause>
       <inverse_attr>
        <attribute_id>the_greenhouse</attribute_id>
        <entity_ref>greenhouse</entity_ref>
        <attribute_ref>holds_plants</attribute_ref>
       </inverse_attr>
     </inverse_clause>
     <where_clause>
       <domain_rule>
        <label>r1</label>
        <logical_expression>
         <embedded_remark>A greenhouse plant can survive in the greenhouse
```



```
          temperature</embedded_remark>
         <function_call>
          <function_ref>is_sub_range</function_ref>
          <qualified_factor>
           <attribute_ref>the_greenhouse</attribute_ref>
           <qualifier>
            <attribute_ref>enforced_temperature_range</attribute_ref>
           </qualifier>
          </qualified_factor>
          <qualified_factor><self/>
           <qualifier>
            <entity_ref>plant</entity_ref>
            <attribute_ref>survival_temperature_range</attribute_ref>
           </qualifier>
          </qualified_factor>
         </function_call>
        </logical_expression>
       </domain_rule>
       <domain_rule>
        <label>r2</label>
        <logical_expression>
         <embedded_remark>A greenhouse plan cannot survive in the garden
          temperature</embedded_remark>
         <unary_op><not/>
          <function_call>
           <function_ref>is_sub_range</function_ref>
           <qualified_factor>
            <attribute_ref>the_greenhouse</attribute_ref>
            <qualifier>
             <attribute_ref>the_garden</attribute_ref>
             <attribute_ref>climatic_temperature_range</attribute_ref>
            </qualifier>
           </qualified_factor>
           <qualified_factor><self/>
            <qualifier>
             <entity_ref>plant</entity_ref>
             <attribute_ref>survival_temperature_range</attribute_ref>
            </qualifier>
           </qualified_factor>
          </function_call>
         </unary_op>
        </logical_expression>
       </domain_rule>
     </where_clause>
    </entity_decl>
    <entity_decl>
     <embedded_remark>** An "outdoors_plant" is a "plant" which can survive
      outside of the ** "greenhouse". It must be grown in a flower "bed".
In order to
      ensure ** that the "outdoors_plant" can cope with the acidity of the
"bed" it
      ** has an associated "ph_range". ** The "ph_range" of the
"outdoors_plant" must
      include the "ph" of the ** "bed".</embedded_remark>
     <entity_id>outdoors_plant</entity_id>
```



```
<subtype_of>
 <entity_ref>plant</entity_ref>
</subtype_of>
<explicit_attr_block>
 <explicit_attr>
  <attribute_id>survival_ph_range</attribute_id>
  <base_type>
   <entity_ref>ph_range</entity_ref>
  </base_type>
 </explicit_attr>
</explicit_attr_block>
<inverse_clause>
 <inverse_attr>
  <attribute_id>the_beds</attribute_id>
  <entity_ref>bed</entity_ref>
  <attribute_ref>holds_plants</attribute_ref>
  <inverse_set>
   <bound_spec>
    <lower_bound>
     <integer_literal>1</integer_literal>
    </lower_bound>
    <upper_bound><unset/>
    </upper_bound>
   </bound_spec>
  </inverse_set>
 </inverse_attr>
</inverse_clause>
<where_clause>
 <domain_rule>
  <embedded_remark>The ph range of the outdoors plant must include the
   ph value ** of the bed </embedded_remark>
  <label>r1</label>
  <logical_expression>
   <relation_expression><equal/>
    <query>
     <variable_id>b</variable_id>
     <aggregate_source>
      <attribute_ref>the_beds</attribute_ref>
     </aggregate_source>
     <logical_expression>
      <function_call>
       <function_ref>value_is_within_range</function_ref>
       <qualified_factor>
        <variable_ref>b</variable_ref>
        <qualifier>
         <attribute_ref>acidity</attribute_ref>
        </qualifier>
       </qualified_factor>
       <attribute_ref>survival_ph_range</attribute_ref>
      </function_call>
     </logical_expression>
    </query>
    <population>
     <entity_ref>the_beds</entity_ref>
    </population>
```



```
      </relation_expression>
     </logical_expression>
    </domain_rule>
    <domain_rule>
     <embedded_remark>An outdoors plant can survive in the garden
      temperature </embedded_remark>
     <logical_expression>
      <function_call>
       <function_ref>is_sub_range</function_ref>
       <qualified_factor>
        <population>
         <entity_ref>the_beds</entity_ref>
        </population>
        <qualifier>
         <index_qualifier>
          <low_index>
           <integer_literal>1</integer_literal>
          </low_index>
         </index_qualifier>
         <attribute_ref>the_garden</attribute_ref>
         <attribute_ref>climatic_temperature_range</attribute_ref>
        </qualifier>
       </qualified_factor>
       <qualified_factor><self/>
        <qualifier>
         <entity_ref>plant</entity_ref>
         <attribute_ref>survival_temperature_range</attribute_ref>
        </qualifier>
       </qualified_factor>
      </function_call>
     </logical_expression>
    </domain_rule>
   </where_clause>
  </entity_decl>
  <entity_decl>
   <entity_id>aquatic_plant</entity_id>
   <subtype_of>
    <entity_ref>plant</entity_ref>
   </subtype_of>
   <explicit_attr_block>
    <explicit_attr>
     <attribute_id>aquatic_plant_type</attribute_id>
     <base_type>
      <type_ref>aquatic_plant_types</type_ref>
     </base_type>
    </explicit_attr>
    <explicit_attr>
     <attribute_id>oxygen_volumetric_requirement</attribute_id>
     <base_type>
      <type_ref>positive_integer</type_ref>
     </base_type>
    </explicit_attr>
    <explicit_attr>
     <attribute_id>aquatic_plant_size</attribute_id>
     <base_type>
```



```
        <type_ref>positive_integer</type_ref>
       </base_type>
      </explicit_attr>
    </explicit_attr_block>
   </entity_decl>
   <function_decl>
    <embedded_remark>** This function checks that a real value lies within
     the specified ** "real_value_range". </embedded_remark>
    <function_id>value_is_within_range</function_id>
    <formal_parameter_block>
     <formal_parameter>
      <parameter_id>v</parameter_id>
      <real>
      </real>
     </formal_parameter>
     <formal_parameter>
      <parameter_id>r</parameter_id>
      <entity_ref>real_value_range</entity_ref>
     </formal_parameter>
    </formal_parameter_block>
    <function_return_type><boolean/>
    </function_return_type>
    <statement_block>
     <return_stmt>
      <term><and/>
       <parenthetic_expression>
        <relation_expression><greater_than_or_equal/>
         <parameter_ref>v</parameter_ref>
         <qualified_factor>
          <parameter_ref>r</parameter_ref>
          <qualifier>
           <attribute_ref>minimum_value</attribute_ref>
          </qualifier>
         </qualified_factor>
        </relation_expression>
       </parenthetic_expression>
       <parenthetic_expression>
        <relation_expression><less_than_or_equal/>
         <parameter_ref>v</parameter_ref>
         <qualified_factor>
          <parameter_ref>r</parameter_ref>
          <qualifier>
           <attribute_ref>maximum_value</attribute_ref>
          </qualifier>
         </qualified_factor>
        </relation_expression>
       </parenthetic_expression>
      </term>
     </return_stmt>
    </statement_block>
   </function_decl>
   <function_decl>
    <embedded_remark>** This function checks that one "real_value_range"
is
     completely ** contained within another. </embedded_remark>
```



```
    <function_id>is_sub_range</function_id>
    <formal_parameter_block>
     <formal_parameter>
      <parameter_id>r1</parameter_id>
      <entity_ref>real_value_range</entity_ref>
     </formal_parameter>
     <formal_parameter>
      <parameter_id>r2</parameter_id>
      <entity_ref>real_value_range</entity_ref>
     </formal_parameter>
    </formal_parameter_block>
    <function_return_type><boolean/>
    </function_return_type>
    <statement_block>
     <return_stmt>
      <parenthetic_expression>
       <term><and/>
        <function_call>
         <function_ref>value_is_within_range</function_ref>
         <qualified_factor>
          <parameter_ref>r1</parameter_ref>
          <qualifier>
           <attribute_ref>minimum_value</attribute_ref>
          </qualifier>
         </qualified_factor>
         <parameter_ref>r2</parameter_ref>
        </function_call>
        <function_call>
         <function_ref>value_is_within_range</function_ref>
         <qualified_factor>
          <parameter_ref>r1</parameter_ref>
          <qualifier>
           <attribute_ref>maximum_value</attribute_ref>
          </qualifier>
         </qualified_factor>
         <parameter_ref>r2</parameter_ref>
        </function_call>
       </term>
      </parenthetic_expression>
     </return_stmt>
    </statement_block>
   </function_decl>
  </schema_decl>
</express_schema>
<express_schema>
 <schema_decl>
  <embedded_remark>============================ S C H E M A
   ====================== </embedded_remark>
  <schema_id>support_items</schema_id>
  <type_decl>
   <embedded_remark>** The appropriate base type to use for "temperature"
is
    not clear from ** the description. However, given that Mr Jones takes
great
```



```
      care of ** "temperature" values it was decided to model it as a real
value.
      </embedded_remark>
     <type_id>temperature</type_id>
     <underlying_type>
      <real>
      </real>
     </underlying_type>
   </type_decl>
   <entity_decl>
     <embedded_remark>** Both "temperatures" and "ph" values for a plant
are
      within a ** "real_value_range". </embedded_remark>
     <entity_id>real_value_range</entity_id>
     <abstract_supertype>
      <supertype_one_of>
       <entity_ref>temperature_range</entity_ref>
       <entity_ref>ph_range</entity_ref>
       <entity_ref>efficiency_range</entity_ref>
      </supertype_one_of>
     </abstract_supertype>
     <explicit_attr_block>
      <explicit_attr>
       <attribute_id>minimum_value</attribute_id>
       <base_type>
        <real>
        </real>
       </base_type>
      </explicit_attr>
      <explicit_attr>
       <attribute_id>maximum_value</attribute_id>
       <base_type>
        <real>
        </real>
       </base_type>
      </explicit_attr>
     </explicit_attr_block>
     <where_clause>
      <domain_rule>
       <label>the_values_must_be_sensible</label>
       <logical_expression>
        <relation_expression><greater_than_or_equal/>
         <attribute_ref>maximum_value</attribute_ref>
         <attribute_ref>minimum_value</attribute_ref>
        </relation_expression>
       </logical_expression>
      </domain_rule>
     </where_clause>
   </entity_decl>
   <entity_decl>
     <embedded_remark>** A "plant" is able to survive within a
      "temperature_range". The ** "garden" will throughout the year have a
      "temperature_range" which the ** "outdoors_plants" must be able to
survive in.
      Similarly, the ** "greenhouse" has a "temperature_range" which the
```



```
      "greenhouse_plants" ** can survive in. </embedded_remark>
     <entity_id>temperature_range</entity_id>
     <subtype_of>
      <entity_ref>real_value_range</entity_ref>
     </subtype_of>
     <explicit_attr_block>
      <explicit_attr>
       <qualified_attribute>
        <entity_ref>real_value_range</entity_ref>
        <attribute_ref>minimum_value</attribute_ref>
       </qualified_attribute>
       <base_type>
        <type_ref>temperature</type_ref>
       </base_type>
      </explicit_attr>
      <explicit_attr>
       <qualified_attribute>
        <entity_ref>real_value_range</entity_ref>
        <attribute_ref>maximum_value</attribute_ref>
       </qualified_attribute>
       <base_type>
        <type_ref>temperature</type_ref>
       </base_type>
      </explicit_attr>
     </explicit_attr_block>
    </entity_decl>
    <type_decl>
     <embedded_remark>** The description mentions both Latin names and
English
      names for ** "plants". These two `types' of "plant_name" were
considered to
      have ** sufficient in common to justify their modelling by a single
defined **
      type. ** It is not clear if some constraints should be specified for
**
      "plant_name". For example, it may sensible to ensure that it is **
not ** empty
      and that it is alphabetic. </embedded_remark>
     <type_id>plant_name</type_id>
     <underlying_type>
      <string>
      </string>
     </underlying_type>
    </type_decl>
    <type_decl>
     <embedded_remark>** The acidity of the "beds" and the range acceptable
to
      ** "outdoors_plants" has been modelled by the defined type "ph". It
is ** not
      clear which base type should be used for "ph"; however, given that **
Mr Jones
      takes pride in compatibility of values, it was decided to use ** the
real type.
      ** A "ph" value is constrained to being between zero and fourteen. In
**
```



```
      reality, it is highly unlikely that Mr Jones would ever use "ph" **
values near
      the extremes. However, there is no indication of more ** realistic
limits
      within the domain description. </embedded_remark>
     <type_id>ph</type_id>
     <underlying_type>
      <real>
      </real>
     </underlying_type>
     <where_clause>
      <domain_rule>
       <label>the_ph_is_between_0_and_14</label>
       <logical_expression>
        <interval>
         <interval_low_inclusive>
          <integer_literal>0</integer_literal>
         </interval_low_inclusive>
         <interval_item><self/>
         </interval_item>
         <interval_high_inclusive>
          <integer_literal>14</integer_literal>
         </interval_high_inclusive>
        </interval>
       </logical_expression>
      </domain_rule>
     </where_clause>
   </type_decl>
   <entity_decl>
    <embedded_remark>** An "outdoors_plant", indeed any "plant", can
survive
      a range of "ph" ** values. </embedded_remark>
     <entity_id>ph_range</entity_id>
     <subtype_of>
      <entity_ref>real_value_range</entity_ref>
     </subtype_of>
     <explicit_attr_block>
      <explicit_attr>
       <qualified_attribute>
        <entity_ref>real_value_range</entity_ref>
        <attribute_ref>minimum_value</attribute_ref>
       </qualified_attribute>
       <base_type>
        <type_ref>ph</type_ref>
       </base_type>
      </explicit_attr>
      <explicit_attr>
       <qualified_attribute>
        <entity_ref>real_value_range</entity_ref>
        <attribute_ref>maximum_value</attribute_ref>
       </qualified_attribute>
       <base_type>
        <type_ref>ph</type_ref>
       </base_type>
      </explicit_attr>
```



```
  </explicit_attr_block>
</entity_decl>
<type_decl>
  <type_id>positive_integer</type_id>
  <underlying_type><integer/>
  </underlying_type>
  <where_clause>
    <domain_rule>
      <label>positive_only</label>
      <logical_expression>
        <relation_expression><greater_than/><self/>
          <integer_literal>0</integer_literal>
        </relation_expression>
      </logical_expression>
    </domain_rule>
  </where_clause>
</type_decl>
<entity_decl>
  <entity_id>efficiency_range</entity_id>
  <subtype_of>
    <entity_ref>real_value_range</entity_ref>
  </subtype_of>
  <explicit_attr_block>
    <explicit_attr>
      <qualified_attribute>
        <entity_ref>real_value_range</entity_ref>
        <attribute_ref>minimum_value</attribute_ref>
      </qualified_attribute>
      <base_type>
        <type_ref>percentage</type_ref>
      </base_type>
    </explicit_attr>
    <explicit_attr>
      <qualified_attribute>
        <entity_ref>real_value_range</entity_ref>
        <attribute_ref>maximum_value</attribute_ref>
      </qualified_attribute>
      <base_type>
        <type_ref>percentage</type_ref>
      </base_type>
    </explicit_attr>
  </explicit_attr_block>
</entity_decl>
<type_decl>
  <type_id>percentage</type_id>
  <underlying_type>
    <real>
    </real>
  </underlying_type>
  <where_clause>
    <domain_rule>
      <label>between_0_and_100</label>
      <logical_expression>
        <interval>
          <interval_low_inclusive>
```



```
          <real_literal>0.0</real_literal>
         </interval_low_inclusive>
         <interval_item><self/>
         </interval_item>
         <interval_high_inclusive>
          <real_literal>100.0</real_literal>
         </interval_high_inclusive>
        </interval>
       </logical_expression>
     </domain_rule>
   </where_clause>
  </type_decl>
  <type_decl>
   <type_id>fractional_value</type_id>
   <underlying_type>
    <real>
    </real>
   </underlying_type>
   <where_clause>
    <domain_rule>
     <label>between_0_and_1</label>
     <logical_expression>
      <interval>
       <interval_low_inclusive>
        <real_literal>0.0</real_literal>
       </interval_low_inclusive>
       <interval_item><self/>
       </interval_item>
       <interval_high_inclusive>
        <real_literal>1.0</real_literal>
       </interval_high_inclusive>
      </interval>
     </logical_expression>
    </domain_rule>
   </where_clause>
  </type_decl>
  <type_decl>
   <type_id>efficiency_measure</type_id>
   <underlying_type>
    <select>
     <type_ref>percentage</type_ref>
     <type_ref>fractional_value</type_ref>
    </select>
   </underlying_type>
  </type_decl>
  <type_decl>
   <type_id>efficiency</type_id>
   <underlying_type>
    <select>
     <type_ref>efficiency_measure</type_ref>
     <entity_ref>efficiency_range</entity_ref>
    </select>
   </underlying_type>
  </type_decl>
 </schema_decl>
```



```
</express_schema>
<express_schema>
 <schema_decl>
  <embedded_remark>Mr Smith's garden is a simple one and has been set up
to
    illustrate REFERENCE and USE from the schema mr_jones_garden. Some of
the
    entities in this schema have the same name as in mr_jones_garden to
illustrate
    name clash handling between schemas. *) </embedded_remark>
  <schema_id>mr_smiths_garden</schema_id>
  <entity_decl>
    <entity_id>garden</entity_id>
    <explicit_attr_block>
     <explicit_attr>
      <attribute_id>has_bed</attribute_id>
      <base_type>
       <entity_ref>bed</entity_ref>
      </base_type>
     </explicit_attr>
    </explicit_attr_block>
  </entity_decl>
  <entity_decl>
    <entity_id>bed</entity_id>
    <explicit_attr_block>
     <explicit_attr>
      <attribute_id>description</attribute_id>
      <base_type>
       <string>
       </string>
      </base_type>
     </explicit_attr>
    </explicit_attr_block>
    <inverse_clause>
     <inverse_attr>
      <attribute_id>the_garden</attribute_id>
      <entity_ref>garden</entity_ref>
      <attribute_ref>has_beds</attribute_ref>
     </inverse_attr>
    </inverse_clause>
  </entity_decl>
  <entity_decl>
    <embedded_remark>As plant is USEd by mr_jones_garden schema, it is
     imported but its subtypes will not be visible in that schema so the
only
     possible combination is plant on its own.</embedded_remark>
    <entity_id>plant</entity_id>
    <supertype_of>
     <supertype_and_or>
      <entity_ref>flower_plant</entity_ref>
      <entity_ref>vegetable_plant</entity_ref>
     </supertype_and_or>
    </supertype_of>
    <explicit_attr_block>
     <explicit_attr>
```



```
      <attribute_id>name</attribute_id>
      <base_type>
       <string>
       </string>
      </base_type>
     </explicit_attr>
    </explicit_attr_block>
   </entity_decl>
   <entity_decl>
    <entity_id>flower_plant</entity_id>
    <subtype_of>
     <entity_ref>plant</entity_ref>
    </subtype_of>
    <explicit_attr_block>
     <explicit_attr>
      <attribute_id>flower_colour</attribute_id>
      <base_type>
       <string>
       </string>
      </base_type>
     </explicit_attr>
    </explicit_attr_block>
   </entity_decl>
   <entity_decl>
    <entity_id>vegetable_plant</entity_id>
    <subtype_of>
     <entity_ref>plant</entity_ref>
    </subtype_of>
    <explicit_attr_block>
     <explicit_attr>
      <attribute_id>vegetable_type</attribute_id>
      <base_type>
       <string>
       </string>
      </base_type>
     </explicit_attr>
    </explicit_attr_block>
   </entity_decl>
  </schema_decl>
 </express_schema>
</iso_10303_28>
```

H.3 Example data represented using the late binding

This subclause presents a representation_category of “LB” XML document (see 10.3.1) corresponding to the late bound DTD found in Annex D.

```
<?xml version="1.0" encoding="utf-8"?>
<iso_10303_28 representation_category=”LB”>
 <express_data id=”data1”>
  <data_section_header>
```



```
    <documentation>This example file has been produced using a combination
of
     tools based on an original hand-crafted example file by Robin La
Fontaine on 21
     March 2000. The DTD used was generated from the Express model of Mr
Jones'
     Garden according to the ETEB definition of 2000-06-07. Updated by RLF
15
     June 2000 to add enumeration_ref in type_literal.</documentation>
   </data_section_header>
   <schema_instance express_schema_name="mr_jones_garden" id="data-id1">
    <entity_instance express_entity_name="garden" id="g1">
     <attribute_instance express_attribute_name="has_pond">
      <entity_instance_ref refid="fp1"/>
     </attribute_instance>
     <attribute_instance express_attribute_name="has_greenhouse">
      <entity_instance_ref refid="gr1"/>
     </attribute_instance>
     <attribute_instance
     express_attribute_name="climatic_temperature_range">
      <entity_instance_ref refid="tr1"/>
     </attribute_instance>
     <attribute_instance express_attribute_name="has_beds">
      <set_literal> <entity_instance_ref refid="bed1"/>
       <entity_instance_ref refid="bed2"/>
<entity_instance_ref refid="bed3"/>
       <entity_instance_ref refid="bed4"/>
<entity_instance_ref refid="bed5"/>
      </set_literal>
     </attribute_instance>
     <attribute_instance express_attribute_name="neighbours_garden">
      <entity_instance_ref refid="sg1"/>
     </attribute_instance>
    </entity_instance>
    <entity_instance_as_group id="po1">
     <partial_entity_instance express_entity_name="pond">
      <attribute_instance express_attribute_name="maintained_by">
       <entity_instance_ref refid="wts1"/>
      </attribute_instance>
      <attribute_instance express_attribute_name="plants">
       <set_literal>
<entity_instance_ref refid="aquatic_plant1"/>
       </set_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="water_volume">
       <type_literal express_type_name="positive_integer">
        <integer_literal>300</integer_literal>
       </type_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="water_ph">
       <type_literal express_type_name="ph">
        <real_literal>3.5</real_literal>
       </type_literal>
      </attribute_instance>
     </partial_entity_instance>
```



```
  <partial_entity_instance express_entity_name="fish_pond" id="fp1">
   <attribute_instance express_attribute_name="fish">
    <set_literal>
     <type_literal express_type_name="fish_type">
      <enumeration_ref>goldfish</enumeration_ref>
     </type_literal>
    </set_literal>
   </attribute_instance>
   <attribute_instance express_attribute_name="fish_colours">
    <set_literal>
     <type_literal express_type_name="observed_fish_colour">
      <type_literal express_type_name="fish_colour">
       <enumeration_ref>yellow</enumeration_ref>
      </type_literal>
     </type_literal>
    </set_literal>
   </attribute_instance>
  </partial_entity_instance>
 </entity_instance_as_group>
 <entity_instance express_entity_name="plant" id="plant1">
  <attribute_instance express_attribute_name="colour">
   <type_literal express_type_name="flower_colour">
    <enumeration_ref>red</enumeration_ref>
   </type_literal>
  </attribute_instance>
  <attribute_instance express_attribute_name="latin_name">
   <type_literal express_type_name="plant_name">
    <string_literal>Lilium</string_literal>
   </type_literal>
  </attribute_instance>
  <attribute_instance
  express_attribute_name="survival_temperature_range">
   <entity_instance_ref refid="tr2"/>
  </attribute_instance>
  <partial_entity_instance express_entity_name="aquatic_plant"
  id="aquatic_plant1">
   <attribute_instance express_attribute_name="aquatic_plant_type">
    <type_literal express_type_name="aquatic_plant_types">
     <enumeration_ref>lilly</enumeration_ref>
    </type_literal>
   </attribute_instance>
   <attribute_instance
   express_attribute_name="oxygen_volumetric_requirement">
    <type_literal express_type_name="positive_integer">
     <integer_literal>3</integer_literal>
    </type_literal>
   </attribute_instance>
   <attribute_instance express_attribute_name="aquatic_plant_size">
    <type_literal express_type_name="positive_integer">
     <integer_literal>1</integer_literal>
    </type_literal>
   </attribute_instance>
  </partial_entity_instance>
 </entity_instance>
 <entity_instance express_entity_name="real_value_range" id="rvr2"
```



```
    express_schema_name="support_items">
      <attribute_instance express_attribute_name="minimum_value">
        <real_literal>-5.0</real_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="maximum_value">
        <real_literal>20.0</real_literal>
      </attribute_instance>
      <partial_entity_instance express_entity_name="temperature_range"
      id="tr2">
      </partial_entity_instance>
    </entity_instance>
    <entity_instance express_entity_name="bed" id="bed1">
      <attribute_instance express_attribute_name="acidity">
        <type_literal express_type_name="ph">
          <real_literal>4.0</real_literal>
        </type_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="holds_plants">
        <set_literal>
<entity_instance_ref refid="op1"/>
          <entity_instance_ref refid="op2"/>
        </set_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="planting_plan">
        <array_literal>
          <array_literal>
<entity_instance_ref refid="op1"/>
            <entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
            <entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
            <entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
            <entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
            <entity_instance_ref refid="op1"/>
          </array_literal>
          <array_literal>
<entity_instance_ref refid="op2"/>
            <entity_instance_ref refid="op2"/>
<entity_instance_ref refid="op2"/>
            <entity_instance_ref refid="op2"/>
<entity_instance_ref refid="op2"/>
            <entity_instance_ref refid="op2"/>
<entity_instance_ref refid="op2"/>
            <entity_instance_ref refid="op2"/>
<entity_instance_ref refid="op2"/>
            <entity_instance_ref refid="op2"/>
          </array_literal>
          <array_literal> <unset/>
<unset/>
<unset/>
<unset/>
<unset/>
            <unset/>
```



```
<unset/>
<unset/>
<unset/>
<unset/>
        </array_literal>
        <array_literal>
<entity_instance_ref refid="op1"/>
         <entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
         <entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
         <entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
         <entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
         <entity_instance_ref refid="op1"/>
        </array_literal>
        <array_literal>
<entity_instance_ref refid="op1"/>
         <entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
         <entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
         <entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
         <entity_instance_ref refid="op1"/>
<entity_instance_ref refid="op1"/>
         <entity_instance_ref refid="op1"/>
        </array_literal>
       </array_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="flowering_order">
       <list_literal>
<entity_instance_ref refid="op2"/>
        <entity_instance_ref refid="op1"/>
       </list_literal>
      </attribute_instance>
    </entity_instance>
    <entity_instance express_entity_name="bed" id="bed2">
     <attribute_instance express_attribute_name="acidity">
      <type_literal express_type_name="ph">
       <real_literal>4.2</real_literal>
      </type_literal>
     </attribute_instance>
     <attribute_instance express_attribute_name="holds_plants">
      <set_literal>
<entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op2"/>
       </set_literal>
     </attribute_instance>
     <attribute_instance express_attribute_name="flowering_order">
      <list_literal>
<entity_instance_ref refid="op2"/>
        <entity_instance_ref refid="op1"/>
       </list_literal>
```



```
      </attribute_instance>
    </entity_instance>
    <entity_instance express_entity_name="bed" id="bed3">
      <attribute_instance express_attribute_name="acidity">
        <type_literal express_type_name="ph">
          <real_literal>4.3</real_literal>
        </type_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="holds_plants">
        <set_literal>
<entity_instance_ref refid="op1"/>
          <entity_instance_ref refid="op2"/>
        </set_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="flowering_order">
        <list_literal>
<entity_instance_ref refid="op2"/>
          <entity_instance_ref refid="op1"/>
        </list_literal>
      </attribute_instance>
    </entity_instance>
    <entity_instance express_entity_name="bed" id="bed4">
      <attribute_instance express_attribute_name="acidity">
        <type_literal express_type_name="ph">
          <real_literal>4.4</real_literal>
        </type_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="holds_plants">
        <set_literal>
<entity_instance_ref refid="op1"/>
          <entity_instance_ref refid="op2"/>
        </set_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="flowering_order">
        <list_literal>
<entity_instance_ref refid="op2"/>
          <entity_instance_ref refid="op1"/>
        </list_literal>
      </attribute_instance>
    </entity_instance>
    <entity_instance express_entity_name="bed" id="bed5">
      <attribute_instance express_attribute_name="acidity">
        <type_literal express_type_name="ph">
          <real_literal>4.5</real_literal>
        </type_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="holds_plants">
        <set_literal>
<entity_instance_ref refid="op1"/>
          <entity_instance_ref refid="op2"/>
        </set_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="flowering_order">
        <list_literal>
<entity_instance_ref refid="op2"/>
```



```
        <entity_instance_ref refid="op1"/>
       </list_literal>
      </attribute_instance>
    </entity_instance>
    <entity_instance express_entity_name="plant" id="p1">
      <attribute_instance express_attribute_name="colour">
       <type_literal express_type_name="flower_colour">
        <enumeration_ref>red</enumeration_ref>
       </type_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="latin_name">
       <type_literal express_type_name="plant_name">
        <string_literal>rosa</string_literal>
       </type_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="english_names">
       <set_literal>
        <type_literal express_type_name="plant_name">
         <string_literal>rose</string_literal>
        </type_literal>
       </set_literal>
      </attribute_instance>
      <attribute_instance
      express_attribute_name="survival_temperature_range">
       <entity_instance_ref refid="tr3"/>
      </attribute_instance>
      <partial_entity_instance express_entity_name="outdoors_plant"
id="op1">

       <attribute_instance express_attribute_name="survival_ph_range">
        <entity_instance_ref refid="phr1"/>
       </attribute_instance>
      </partial_entity_instance>
    </entity_instance>
    <entity_instance express_entity_name="plant" id="p2">
      <attribute_instance express_attribute_name="colour">
       <type_literal express_type_name="flower_colour">
        <enumeration_ref>yellow</enumeration_ref>
       </type_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="latin_name">
       <type_literal express_type_name="plant_name">
        <string_literal>Narcissus</string_literal>
       </type_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="english_names">
       <set_literal>
        <type_literal express_type_name="plant_name">
         <string_literal>daffodil</string_literal>
        </type_literal>
       </set_literal>
      </attribute_instance>
      <attribute_instance
      express_attribute_name="survival_temperature_range">
       <entity_instance_ref refid="tr3"/>
```



```
      </attribute_instance>
      <partial_entity_instance express_entity_name="outdoors_plant"
id="op2">

       <attribute_instance express_attribute_name="survival_ph_range">
        <entity_instance_ref refid="phr1"/>
       </attribute_instance>
      </partial_entity_instance>
    </entity_instance>
    <entity_instance express_entity_name="greenhouse" id="gr1">
      <attribute_instance
      express_attribute_name="enforced_temperature_range">
        <entity_instance_ref refid="tr1"/>
      </attribute_instance>
      <attribute_instance express_attribute_name="holds_plants">
        <set_literal>
<entity_instance_ref refid="grp1"/>
       </set_literal>
      </attribute_instance>
    </entity_instance>
    <entity_instance express_entity_name="plant" id="p-grp1">
      <attribute_instance express_attribute_name="colour">
        <type_literal express_type_name="flower_colour">
          <enumeration_ref>red</enumeration_ref>
        </type_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="latin_name">
        <type_literal express_type_name="plant_name">
          <string_literal>orchidaceae</string_literal>
        </type_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="english_names">
        <set_literal>
          <type_literal express_type_name="plant_name">
            <string_literal>orchid</string_literal>
          </type_literal>
        </set_literal>
      </attribute_instance>
      <attribute_instance
      express_attribute_name="survival_temperature_range">
        <entity_instance_ref refid="tr1"/>
      </attribute_instance>
      <partial_entity_instance express_entity_name="greenhouse_plant"
      id="grp1">
      </partial_entity_instance>
    </entity_instance>
    <entity_instance express_entity_name="real_value_range" id="rvr1"
    express_schema_name="support_items">
      <attribute_instance express_attribute_name="minimum_value">
        <real_literal>3.5</real_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="maximum_value">
        <real_literal>27.0</real_literal>
      </attribute_instance>
      <partial_entity_instance express_entity_name="temperature_range"
```



```
      id="tr1">
      </partial_entity_instance>
    </entity_instance>
    <entity_instance express_entity_name="real_value_range" id="rvr3"
    express_schema_name="support_items">
      <attribute_instance express_attribute_name="minimum_value">
        <real_literal>-3.0</real_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="maximum_value">
        <real_literal>33.0</real_literal>
      </attribute_instance>
      <partial_entity_instance express_entity_name="temperature_range"
      id="tr3">
      </partial_entity_instance>
    </entity_instance>
    <entity_instance express_entity_name="real_value_range" id="rvr-phr1"
    express_schema_name="support_items">
      <attribute_instance express_attribute_name="minimum_value">
        <real_literal>3.1</real_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="maximum_value">
        <real_literal>5.0</real_literal>
      </attribute_instance>
      <partial_entity_instance express_entity_name="ph_range" id="phr1">
      </partial_entity_instance>
    </entity_instance>
    <entity_instance express_entity_name="garden" id="sg1"
    express_schema_name="mr_smiths_garden">
      <attribute_instance express_attribute_name="has_bed">
        <entity_instance_ref refid="sbed1"/>
      </attribute_instance>
    </entity_instance>
    <entity_instance express_entity_name="bed"
    id="sbed1" express_schema_name="mr_smiths_garden">
      <attribute_instance express_attribute_name="description">
        <string_literal>Mr Smith's flower bed.</string_literal>
      </attribute_instance>
    </entity_instance>
    <entity_instance express_entity_name="plant" id="splant1"
     express_schema_name="mr_smiths_garden">
      <attribute_instance express_attribute_name="name">
        <string_literal>rose</string_literal>
      </attribute_instance>
    </entity_instance>
    <entity_instance express_entity_name="water_treatment_system"
id="wts1">
      <attribute_instance express_attribute_name="capacity">
        <type_literal express_type_name="positive_integer">
          <integer_literal>500</integer_literal>
        </type_literal>
      </attribute_instance>
      <partial_entity_instance express_entity_name="pumping_system">
        <attribute_instance express_attribute_name="pumping_efficiency">
          <type_literal express_type_name="efficiency">
            <entity_instance_ref refid="efr1"/>
```



```
        </type_literal>
       </attribute_instance>
       <attribute_instance express_attribute_name="number_of_pumps">
        <type_literal express_type_name="positive_integer">
         <integer_literal>1</integer_literal>
        </type_literal>
       </attribute_instance>
      </partial_entity_instance>
     </entity_instance>
     <entity_instance express_entity_name="real_value_range" id="rvr-efr1"
      express_schema_name="support_items">
      <attribute_instance express_attribute_name="minimum_value">
       <real_literal>75</real_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="maximum_value">
       <real_literal>85</real_literal>
      </attribute_instance>
      <partial_entity_instance express_entity_name="efficiency_range"
       id="efr1">
      </partial_entity_instance>
     </entity_instance>
    </schema_instance>
  </express_data>
</iso_10303_28>
```

H.4 Examples using the EXPRESS-Typed Early Binding

This subclause presents the examples schema from H.1 as an ETEB DTD and a `representation_category` of “ETEB” (see 10.3.2) XML document corresponding to that DTD.

H.4.1 Example DTD according to the EXPRESS-Typed Early Binding

The following DTD results from applying the rules defined in clause 8 to the example EXPRESS schemas given in H.1.

```
<!-- DTD elements: total number = 139
Referenced not defined: NIL
Defined not referenced: (binary boolean logical number temperature
iso_10303_28)
Duplicate definitions: NIL
 -->



<!ELEMENT Amphibian
 (enumeration-item)>
<!ATTLIST Amphibian
      express_type_name NMTOKEN #FIXED "amphibian"
      late-bound-element NMTOKEN #FIXED "type_literal"
      value-space NMTOKENS #FIXED "frog newt"
      late-bound-name CDATA #FIXED "value-space enumeration_domain">
```



```
<!ELEMENT Amphibian_pond
 (Amphibian_pond.amphibians)>
<!ATTLIST Amphibian_pond
      express_entity_name NMTOKEN #FIXED "amphibian_pond"
      id ID #IMPLIED
      late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Amphibian_pond.amphibians
 (set-of-Amphibian)>
<!ATTLIST Amphibian_pond.amphibians
      express_attribute_name NMTOKEN #FIXED "amphibians"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Aquatic_plant
 (Aquatic_plant.aquatic_plant_type,
Aquatic_plant.oxygen_volumetric_requirement,
Aquatic_plant.aquatic_plant_size)>
<!ATTLIST Aquatic_plant
      express_entity_name NMTOKEN #FIXED "aquatic_plant"
      id ID #IMPLIED
      late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Aquatic_plant-ref EMPTY>
<!ATTLIST Aquatic_plant-ref
      refid IDREF #REQUIRED
      reftype CDATA #FIXED "refid (aquatic_plant | external_refid)"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      late-bound-element NMTOKEN #FIXED "entity_instance_ref">

<!ELEMENT Aquatic_plant.aquatic_plant_size
 (Positive_integer)>
<!ATTLIST Aquatic_plant.aquatic_plant_size
      express_attribute_name NMTOKEN #FIXED "aquatic_plant_size"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Aquatic_plant.aquatic_plant_type
 (Aquatic_plant_types)>
<!ATTLIST Aquatic_plant.aquatic_plant_type
      express_attribute_name NMTOKEN #FIXED "aquatic_plant_type"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Aquatic_plant.oxygen_volumetric_requirement
 (Positive_integer)>
<!ATTLIST Aquatic_plant.oxygen_volumetric_requirement
      express_attribute_name NMTOKEN #FIXED "oxygen_volumetric_requirement"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Aquatic_plant_types
 (enumeration-item)>
<!ATTLIST Aquatic_plant_types
      express_type_name NMTOKEN #FIXED "aquatic_plant_types"
      late-bound-element NMTOKEN #FIXED "type_literal"
      value-space NMTOKENS #FIXED "lilly weed lotus"
      late-bound-name CDATA #FIXED "value-space enumeration_domain">
```



```
<!ELEMENT array-of-array-of-optional-Outdoors_plant
 (array-of-optional-Outdoors_plant*)>
<!ATTLIST array-of-array-of-optional-Outdoors_plant
      late-bound-element NMTOKEN #FIXED "array_literal">

<!ELEMENT array-of-optional-Outdoors_plant
 (Outdoors_plant-ref | unset)*>
<!ATTLIST array-of-optional-Outdoors_plant
      late-bound-element NMTOKEN #FIXED "array_literal">

<!ELEMENT Author
 (#PCDATA)>
<!ATTLIST Author
      late-bound-element NMTOKEN #FIXED "author">

<!ELEMENT Authorization
 (#PCDATA)>
<!ATTLIST Authorization
      late-bound-element NMTOKEN #FIXED "authorization">

<!ELEMENT bag-of-Outdoors_plant
 (Outdoors_plant-ref*)>
<!ATTLIST bag-of-Outdoors_plant
      late-bound-element NMTOKEN #FIXED "bag_literal">

<!ELEMENT Bed
 (Bed.acidity, Bed.holds_plants, Bed.planting_plan?, Bed.flowering_order?,
Bed.plants_ready_for_planting?)>
<!ATTLIST Bed
      express_entity_name NMTOKEN #FIXED "bed"
      id ID #REQUIRED
      late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Bed-ref EMPTY>
<!ATTLIST Bed-ref
      refid IDREF #REQUIRED
      reftype CDATA #FIXED "refid (bed | external_refid)"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      late-bound-element NMTOKEN #FIXED "entity_instance_ref">

<!ELEMENT Bed.acidity
 (Ph)>
<!ATTLIST Bed.acidity
      express_attribute_name NMTOKEN #FIXED "acidity"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Bed.flowering_order
 (list-of-Outdoors_plant)>
<!ATTLIST Bed.flowering_order
      express_attribute_name NMTOKEN #FIXED "flowering_order"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Bed.holds_plants
 (set-of-Outdoors_plant)>
```



```
<!ATTLIST Bed.holds_plants
      express_attribute_name NMTOKEN #FIXED "holds_plants"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Bed.planting_plan
 (array-of-array-of-optional-Outdoors_plant)>
<!ATTLIST Bed.planting_plan
      express_attribute_name NMTOKEN #FIXED "planting_plan"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Bed.plants_ready_for_planting
 (bag-of-Outdoors_plant)>
<!ATTLIST Bed.plants_ready_for_planting
      express_attribute_name NMTOKEN #FIXED "plants_ready_for_planting"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT binary
 (#PCDATA)>
<!ATTLIST binary
      express_constant_name NMTOKEN #IMPLIED
      external_binary_literal ENTITY #IMPLIED
      late-bound-element NMTOKEN #FIXED "binary_literal"
      notation  (hex | base64) #IMPLIED
      empty_bits CDATA #REQUIRED >

<!ELEMENT boolean
 (false | true)>
<!ATTLIST boolean
      late-bound-element NMTOKEN #FIXED "boolean_literal">


<!ELEMENT data_section_header
 (documentation?)>
<!ATTLIST data_section_header
      late-bound-element NMTOKEN #FIXED "data_section_header">

<!ELEMENT documentation
 (#PCDATA)>
<!ATTLIST documentation
      late-bound-element NMTOKEN #FIXED "documentation">

<!ELEMENT document_name
 (#PCDATA)>
<!ATTLIST document_name
      late-bound-element NMTOKEN #FIXED "document_name">

<!ELEMENT Efficiency
 (Efficiency_measure | Efficiency_range-ref)>
<!ATTLIST Efficiency
      express_schema_name NMTOKEN #FIXED "support_items"
      express_type_name NMTOKEN #FIXED "efficiency"
      late-bound-element NMTOKEN #FIXED "type_literal">

<!ELEMENT Efficiency_measure
 (Percentage | Fractional_value)>
```



```
<!ATTLIST Efficiency_measure
      express_schema_name NMTOKEN #FIXED "support_items"
      express_type_name NMTOKEN #FIXED "efficiency_measure"
      late-bound-element NMTOKEN #FIXED "type_literal">

<!ELEMENT Efficiency_range EMPTY>
<!ATTLIST Efficiency_range
      express_entity_name NMTOKEN #FIXED "efficiency_range"
      express_schema_name NMTOKEN #FIXED "support_items"
      id ID #IMPLIED
      late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Efficiency_range-ref EMPTY>
<!ATTLIST Efficiency_range-ref
      refid IDREF #REQUIRED
      reftype CDATA #FIXED "refid (efficiency_range | external_refid)"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      late-bound-element NMTOKEN #FIXED "entity_instance_ref">

<!ELEMENT enumeration-item (#PCDATA)>
<!ATTLIST enumeration-item
    late-bound-element NMTOKEN #FIXED "enumeration_ref">

<!ENTITY % schema_instance "Mr_jones_garden-schema">

<!ELEMENT express_data
 (data_section_header?, %schema_instance;)>
<!ATTLIST express_data
    id ID #REQUIRED
    name CDATA #IMPLIED
    representation_category NMTOKEN #REQUIRED
    late-bound-element NMTOKEN #FIXED "express_data">


<!ENTITY % schema_decl "">

<!ELEMENT express_schema
(%schema_decl; schema_text | external_refid)>
<!ATTLIST express_schema
      id ID #IMPLIED
      express_schema_description CDATA #IMPLIED
      express_schema_identifier CDATA #IMPLIED
      express_schema_version CDATA #IMPLIED
      late-bound-element NMTOKEN #FIXED "express_schema">

<!ELEMENT external_refid EMPTY>
<!ATTLIST external_refid
  id ID #REQUIRED
  xlink:type CDATA #FIXED "simple"
  xlink:href CDATA #REQUIRED
  xlink:arcrole CDATA #REQUIRED
  xlink:title CDATA #IMPLIED
  xlink:role CDATA #IMPLIED
  xlink:show CDATA #IMPLIED
  xlink:actuate CDATA #IMPLIED
```



```
  late-bound-element NMTOKEN #FIXED "external_refid">

<!ELEMENT false EMPTY>
<!ATTLIST false
      late-bound-element NMTOKEN #FIXED "false">

<!ELEMENT Filtration_system
 (Filtration_system.filtration_efficiency,
Filtration_system.number_of_filters)>
<!ATTLIST Filtration_system
      express_entity_name NMTOKEN #FIXED "filtration_system"
      id ID #IMPLIED
      late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Filtration_system.filtration_efficiency
 (Efficiency)>
<!ATTLIST Filtration_system.filtration_efficiency
      express_attribute_name NMTOKEN #FIXED "filtration_efficiency"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Filtration_system.number_of_filters
 (Positive_integer)>
<!ATTLIST Filtration_system.number_of_filters
      express_attribute_name NMTOKEN #FIXED "number_of_filters"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Fish_colour
 (enumeration-item)>
<!ATTLIST Fish_colour
      express_type_name NMTOKEN #FIXED "fish_colour"
      late-bound-element NMTOKEN #FIXED "type_literal"
      value-space NMTOKENS #FIXED "yellow black silver blue"
      late-bound-name CDATA #FIXED "value-space enumeration_domain">

<!ELEMENT Fish_pond
 (Fish_pond.fish, Fish_pond.fish_colours)>
<!ATTLIST Fish_pond
      express_entity_name NMTOKEN #FIXED "fish_pond"
      id ID #IMPLIED
      late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Fish_pond-ref EMPTY>
<!ATTLIST Fish_pond-ref
      refid IDREF #REQUIRED
      reftype CDATA #FIXED "refid (fish_pond | external_refid)"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      late-bound-element NMTOKEN #FIXED "entity_instance_ref">

<!ELEMENT Fish_pond.fish
 (set-of-Fish_type)>
<!ATTLIST Fish_pond.fish
      express_attribute_name NMTOKEN #FIXED "fish"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Fish_pond.fish_colours
```



```
 (set-of-Observed_fish_colour)>
<!ATTLIST Fish_pond.fish_colours
      express_attribute_name NMTOKEN #FIXED "fish_colours"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Fish_type
 (enumeration-item)>
<!ATTLIST Fish_type
      express_type_name NMTOKEN #FIXED "fish_type"
      late-bound-element NMTOKEN #FIXED "type_literal"
      value-space NMTOKENS #FIXED "koi goldfish"
      late-bound-name CDATA #FIXED "value-space enumeration_domain">

<!ELEMENT Flower_colour
 (enumeration-item)>
<!ATTLIST Flower_colour
      express_type_name NMTOKEN #FIXED "flower_colour"
      late-bound-element NMTOKEN #FIXED "type_literal"
      value-space NMTOKENS #FIXED "red yellow white"
      late-bound-name CDATA #FIXED "value-space enumeration_domain">

<!ELEMENT Fractional_value
 (real)>
<!ATTLIST Fractional_value
      express_schema_name NMTOKEN #FIXED "support_items"
      express_type_name NMTOKEN #FIXED "fractional_value"
      late-bound-element NMTOKEN #FIXED "type_literal">

<!ELEMENT Garden
 (Garden.has_pond, Garden.has_greenhouse,
Garden.climatic_temperature_range, Garden.has_beds,
Garden.neighbours_garden)>
<!ATTLIST Garden
      express_entity_name NMTOKEN #FIXED "garden"
      id ID #REQUIRED
      late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Garden.climatic_temperature_range
 (Temperature_range-ref)>
<!ATTLIST Garden.climatic_temperature_range
      express_attribute_name NMTOKEN #FIXED "climatic_temperature_range"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Garden.has_beds
 (set-of-Bed)>
<!ATTLIST Garden.has_beds
      express_attribute_name NMTOKEN #FIXED "has_beds"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Garden.has_greenhouse
 (Greenhouse-ref)>
<!ATTLIST Garden.has_greenhouse
      express_attribute_name NMTOKEN #FIXED "has_greenhouse"
      late-bound-element NMTOKEN #FIXED "attribute_instance">
```



```
<!ELEMENT Garden.has_pond
 (Fish_pond-ref)>
<!ATTLIST Garden.has_pond
      express_attribute_name NMTOKEN #FIXED "has_pond"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Garden.neighbours_garden
 (Mr_smiths_garden-ref)>
<!ATTLIST Garden.neighbours_garden
      express_attribute_name NMTOKEN #FIXED "neighbours_garden"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Greenhouse
 (Greenhouse.enforced_temperature_range, Greenhouse.holds_plants)>
<!ATTLIST Greenhouse
      express_entity_name NMTOKEN #FIXED "greenhouse"
      id ID #REQUIRED
      late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Greenhouse-ref EMPTY>
<!ATTLIST Greenhouse-ref
      refid IDREF #REQUIRED
      reftype CDATA #FIXED "refid (greenhouse | external_refid)"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      late-bound-element NMTOKEN #FIXED "entity_instance_ref">

<!ELEMENT Greenhouse.enforced_temperature_range
 (Temperature_range-ref)>
<!ATTLIST Greenhouse.enforced_temperature_range
      express_attribute_name NMTOKEN #FIXED "enforced_temperature_range"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Greenhouse.holds_plants
 (set-of-Greenhouse_plant)>
<!ATTLIST Greenhouse.holds_plants
      express_attribute_name NMTOKEN #FIXED "holds_plants"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Greenhouse_plant EMPTY>
<!ATTLIST Greenhouse_plant
      express_entity_name NMTOKEN #FIXED "greenhouse_plant"
      id ID #IMPLIED
      late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Greenhouse_plant-ref EMPTY>
<!ATTLIST Greenhouse_plant-ref
      refid IDREF #REQUIRED
      reftype CDATA #FIXED "refid (greenhouse_plant | external_refid)"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      late-bound-element NMTOKEN #FIXED "entity_instance_ref">

<!ELEMENT integer
 (#PCDATA)>
<!ATTLIST integer
      late-bound-element NMTOKEN #FIXED "integer_literal">
```



```
<!ELEMENT iso_10303_28
 (iso_10303_28_header?,
  (express_schema | express_data)+)>
<!ATTLIST iso_10303_28
    representation_category NMTOKENS #REQUIRED
    version CDATA #FIXED "PDTS"
    late-bound-element NMTOKEN #FIXED "iso_10303_28">

<!ELEMENT iso_10303_28_header
 (document_name, purpose?, time_stamp?, author?, originating_organization?,
authorization?, originating_system?, preprocessor_version?,
documentation?)>
<!ATTLIST iso_10303_28_header
    late-bound-element NMTOKEN #FIXED "iso_10303_28_header">

<!ELEMENT list-of-Outdoors_plant
 (Outdoors_plant-ref*)>
<!ATTLIST list-of-Outdoors_plant
      late-bound-element NMTOKEN #FIXED "list_literal">

<!ELEMENT Livestock_container
 (Livestock_container.livestock_count?,
Livestock_container.maintained_by?)>
<!ATTLIST Livestock_container
      express_entity_name NMTOKEN #FIXED "livestock_container"
      id ID #IMPLIED
      late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Livestock_container.livestock_count
 (integer)>
<!ATTLIST Livestock_container.livestock_count
      express_attribute_name NMTOKEN #FIXED "livestock_count"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Livestock_container.maintained_by
 (string)>
<!ATTLIST Livestock_container.maintained_by
      express_attribute_name NMTOKEN #FIXED "maintained_by"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Livestock_containerPond
 (Amphibian_pond | Fish_pond | Livestock_container | Ornamental_pond |
Pond)+>
<!ATTLIST Livestock_containerPond
      id ID #REQUIRED
      late-bound-element NMTOKEN #FIXED "entity_instance_as_group">

<!ELEMENT logical
 (false | unknown | true)>
<!ATTLIST Logical
      late-bound-element NMTOKEN #FIXED "logical_literal">

<!ELEMENT Mr_jones_garden-schema
```



```
 (Bed | Garden | Greenhouse | Livestock_containerPond | Mr_smiths_garden |
Mr_smiths_garden-schema.bed | Mr_smiths_plant | Plant | Real_value_range |
Water_treatment_system)*>
<!ATTLIST Mr_jones_garden-schema
      express_schema_identifier CDATA #IMPLIED
      express_schema_name NMTOKEN #FIXED "mr_jones_garden"
      express_schema_version CDATA #IMPLIED
      id ID #IMPLIED
      late-bound-element NMTOKEN #FIXED "schema_instance"
      refid IDREF #IMPLIED
      reftype CDATA #FIXED "refid (express_schema | external_refid)">

<!ELEMENT Mr_smiths_garden
 (Mr_smiths_garden.has_bed)>
<!ATTLIST Mr_smiths_garden
      express_entity_name NMTOKEN #FIXED "garden"
      express_schema_name NMTOKEN #FIXED "mr_smiths_garden"
      id ID #REQUIRED
      late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Mr_smiths_garden-ref EMPTY>
<!ATTLIST Mr_smiths_garden-ref
      refid IDREF #REQUIRED
      reftype CDATA #FIXED "refid (mr_smiths_garden | external_refid)"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      late-bound-element NMTOKEN #FIXED "entity_instance_ref">

<!ELEMENT Mr_smiths_garden-schema.bed
 (Mr_smiths_garden-schema.bed.description)>
<!ATTLIST Mr_smiths_garden-schema.bed
      express_entity_name NMTOKEN #FIXED "bed"
      express_schema_name NMTOKEN #FIXED "mr_smiths_garden"
      id ID #REQUIRED
      late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Mr_smiths_garden-schema.bed-ref EMPTY>
<!ATTLIST Mr_smiths_garden-schema.bed-ref
      refid IDREF #REQUIRED
      reftype CDATA #FIXED "refid (mr_smiths_garden-schema.bed |
external_refid)"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      late-bound-element NMTOKEN #FIXED "entity_instance_ref">

<!ELEMENT Mr_smiths_garden-schema.bed.description
 (string)>
<!ATTLIST Mr_smiths_garden-schema.bed.description
      express_attribute_name NMTOKEN #FIXED "description"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Mr_smiths_garden.has_bed
 (Mr_smiths_garden-schema.bed-ref)>
<!ATTLIST Mr_smiths_garden.has_bed
      express_attribute_name NMTOKEN #FIXED "has_bed"
      late-bound-element NMTOKEN #FIXED "attribute_instance">
```



```
<!ELEMENT Mr_smiths_plant
 (Mr_smiths_plant.name)>
<!ATTLIST Mr_smiths_plant
      express_entity_name NMTOKEN #FIXED "plant"
      express_schema_name NMTOKEN #FIXED "mr_smiths_garden"
      id ID #REQUIRED
      late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Mr_smiths_plant.name
 (string)>
<!ATTLIST Mr_smiths_plant.name
      express_attribute_name NMTOKEN #FIXED "name"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Multi_coloured_fish_colour
 (set-of-Fish_colour)>
<!ATTLIST Multi_coloured_fish_colour
      express_type_name NMTOKEN #FIXED "multi_coloured_fish_colour"
      late-bound-element NMTOKEN #FIXED "type_literal">

<!ELEMENT number
 (real | integer)>
<!ATTLIST number
      late-bound-element NMTOKEN #FIXED "number">

<!ELEMENT Observed_fish_colour
 (Fish_colour | Multi_coloured_fish_colour)>
<!ATTLIST Observed_fish_colour
      express_type_name NMTOKEN #FIXED "observed_fish_colour"
      late-bound-element NMTOKEN #FIXED "type_literal">

<!ELEMENT originating_organization
 (#PCDATA)>
<!ATTLIST originating_organization
      late-bound-element NMTOKEN #FIXED "originating_organization">

<!ELEMENT originating_system
 (#PCDATA)>
<!ATTLIST originating_system
      late-bound-element NMTOKEN #FIXED "originating_system">

<!ELEMENT Ornamental_pond
 (Ornamental_pond.ornaments)>
<!ATTLIST Ornamental_pond
      express_entity_name NMTOKEN #FIXED "ornamental_pond"
      id ID #IMPLIED
      late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Ornamental_pond.ornaments
 (set-of-Pond_ornament)>
<!ATTLIST Ornamental_pond.ornaments
      express_attribute_name NMTOKEN #FIXED "ornaments"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Outdoors_plant
```



```
 (Outdoors_plant.survival_ph_range)>
<!ATTLIST Outdoors_plant
      express_entity_name NMTOKEN #FIXED "outdoors_plant"
      id ID #IMPLIED
      late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Outdoors_plant-ref EMPTY>
<!ATTLIST Outdoors_plant-ref
      refid IDREF #REQUIRED
      reftype CDATA #FIXED "refid (outdoors_plant | external_refid)"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      late-bound-element NMTOKEN #FIXED "entity_instance_ref">

<!ELEMENT Outdoors_plant.survival_ph_range
 (Ph_range-ref)>
<!ATTLIST Outdoors_plant.survival_ph_range
      express_attribute_name NMTOKEN #FIXED "survival_ph_range"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Percentage
 (real)>
<!ATTLIST Percentage
      express_schema_name NMTOKEN #FIXED "support_items"
      express_type_name NMTOKEN #FIXED "percentage"
      late-bound-element NMTOKEN #FIXED "type_literal">

<!ELEMENT Ph
 (real)>
<!ATTLIST Ph
      express_schema_name NMTOKEN #FIXED "support_items"
      express_type_name NMTOKEN #FIXED "ph"
      late-bound-element NMTOKEN #FIXED "type_literal">

<!ELEMENT Ph_range EMPTY>
<!ATTLIST Ph_range
      express_entity_name NMTOKEN #FIXED "ph_range"
      express_schema_name NMTOKEN #FIXED "support_items"
      id ID #IMPLIED
      late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Ph_range-ref EMPTY>
<!ATTLIST Ph_range-ref
      refid IDREF #REQUIRED
      reftype CDATA #FIXED "refid (ph_range | external_refid)"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      late-bound-element NMTOKEN #FIXED "entity_instance_ref">

<!ELEMENT Plant
 (Plant.colour, Plant.latin_name, Plant.english_names?,
Plant.survival_temperature_range, Plant-subtypes)>
<!ATTLIST Plant
      express_entity_name NMTOKEN #FIXED "plant"
      id ID #REQUIRED
      late-bound-element NMTOKEN #FIXED "entity_instance">
```



```
<!ELEMENT Plant-subtypes
 (Greenhouse_plant | Outdoors_plant | Aquatic_plant)+>

<!ELEMENT Plant.colour
 (Flower_colour)>
<!ATTLIST Plant.colour
      express_attribute_name NMTOKEN #FIXED "colour"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Plant.english_names
 (set-of-Plant_name)>
<!ATTLIST Plant.english_names
      express_attribute_name NMTOKEN #FIXED "english_names"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Plant.latin_name
 (Plant_name)>
<!ATTLIST Plant.latin_name
      express_attribute_name NMTOKEN #FIXED "latin_name"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Plant.survival_temperature_range
 (Temperature_range-ref)>
<!ATTLIST Plant.survival_temperature_range
      express_attribute_name NMTOKEN #FIXED "survival_temperature_range"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Plant_name
 (string)>
<!ATTLIST Plant_name
      express_schema_name NMTOKEN #FIXED "support_items"
      express_type_name NMTOKEN #FIXED "plant_name"
      late-bound-element NMTOKEN #FIXED "type_literal">

<!ELEMENT Pond
 (Pond.maintained_by?, Pond.plants, Pond.water_volume, Pond.water_ph)>
<!ATTLIST Pond
      express_entity_name NMTOKEN #FIXED "pond"
      id ID #IMPLIED
      late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Pond.maintained_by
 (Water_treatment_system-ref)>
<!ATTLIST Pond.maintained_by
      express_attribute_name NMTOKEN #FIXED "maintained_by"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Pond.plants
 (set-of-Aquatic_plant)>
<!ATTLIST Pond.plants
      express_attribute_name NMTOKEN #FIXED "plants"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Pond.water_ph
 (Ph)>
```



```
<!ATTLIST Pond.water_ph
      express_attribute_name NMTOKEN #FIXED "water_ph"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Pond.water_volume
 (Positive_integer)>
<!ATTLIST Pond.water_volume
      express_attribute_name NMTOKEN #FIXED "water_volume"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Pond_ornament
 (enumeration-item)>
<!ATTLIST Pond_ornament
      express_type_name NMTOKEN #FIXED "pond_ornament"
      late-bound-element NMTOKEN #FIXED "type_literal"
      value-space NMTOKENS #FIXED "waterfall fountain bridge"
      late-bound-name CDATA #FIXED "value-space enumeration_domain">

<!ELEMENT Positive_integer
 (integer)>
<!ATTLIST Positive_integer
      express_schema_name NMTOKEN #FIXED "support_items"
      express_type_name NMTOKEN #FIXED "positive_integer"
      late-bound-element NMTOKEN #FIXED "type_literal">

<!ELEMENT preprocessor_version
 (#PCDATA)>
<!ATTLIST preprocessor_version
      late-bound-element NMTOKEN #FIXED "preprocessor_version">

<!ELEMENT Pumping_system
 (Pumping_system.pumping_efficiency, Pumping_system.number_of_pumps)>
<!ATTLIST Pumping_system
      express_entity_name NMTOKEN #FIXED "pumping_system"
      id ID #IMPLIED
      late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Pumping_system.number_of_pumps
 (Positive_integer)>
<!ATTLIST Pumping_system.number_of_pumps
      express_attribute_name NMTOKEN #FIXED "number_of_pumps"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Pumping_system.pumping_efficiency
 (Efficiency)>
<!ATTLIST Pumping_system.pumping_efficiency
      express_attribute_name NMTOKEN #FIXED "pumping_efficiency"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT purpose
 (#PCDATA)>
<!ATTLIST purpose
      late-bound-element NMTOKEN #FIXED "purpose">

<!ELEMENT real
```



```
 (#PCDATA)>
<!ATTLIST real
      late-bound-element NMTOKEN #FIXED "real_literal"
      precision CDATA #IMPLIED >

<!ELEMENT Real_value_range
 (Real_value_range.minimum_value, Real_value_range.maximum_value,
Real_value_range-subtypes)>
<!ATTLIST Real_value_range
      express_entity_name NMTOKEN #FIXED "real_value_range"
      express_schema_name NMTOKEN #FIXED "support_items"
      id ID #REQUIRED
      late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Real_value_range-subtypes
 (Temperature_range | Ph_range | Efficiency_range)+>

<!ELEMENT Real_value_range.maximum_value
 (real)>
<!ATTLIST Real_value_range.maximum_value
      express_attribute_name NMTOKEN #FIXED "maximum_value"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT Real_value_range.minimum_value
 (real)>
<!ATTLIST Real_value_range.minimum_value
      express_attribute_name NMTOKEN #FIXED "minimum_value"
      late-bound-element NMTOKEN #FIXED "attribute_instance">

<!ELEMENT schema_text
(#PCDATA)>
<!ATTLIST schema_text
      late-bound-element NMTOKEN #FIXED "schema_text">

<!ELEMENT set-of-Amphibian
 (Amphibian*)>
<!ATTLIST set-of-Amphibian
      late-bound-element NMTOKEN #FIXED "set_literal">

<!ELEMENT set-of-Aquatic_plant
 (Aquatic_plant-ref*)>
<!ATTLIST set-of-Aquatic_plant
      late-bound-element NMTOKEN #FIXED "set_literal">

<!ELEMENT set-of-Bed
 (Bed-ref*)>
<!ATTLIST set-of-Bed
      late-bound-element NMTOKEN #FIXED "set_literal">

<!ELEMENT set-of-Fish_colour
 (Fish_colour*)>
<!ATTLIST set-of-Fish_colour
      late-bound-element NMTOKEN #FIXED "set_literal">

<!ELEMENT set-of-Fish_type
```



```
 (Fish_type*)>
<!ATTLIST set-of-Fish_type
      late-bound-element NMTOKEN #FIXED "set_literal">

<!ELEMENT set-of-Greenhouse_plant
 (Greenhouse_plant-ref*)>
<!ATTLIST set-of-Greenhouse_plant
      late-bound-element NMTOKEN #FIXED "set_literal">

<!ELEMENT set-of-Observed_fish_colour
 (Observed_fish_colour*)>
<!ATTLIST set-of-Observed_fish_colour
      late-bound-element NMTOKEN #FIXED "set_literal">

<!ELEMENT set-of-Outdoors_plant
 (Outdoors_plant-ref*)>
<!ATTLIST set-of-Outdoors_plant
      late-bound-element NMTOKEN #FIXED "set_literal">

<!ELEMENT set-of-Plant_name
 (Plant_name*)>
<!ATTLIST set-of-Plant_name
      late-bound-element NMTOKEN #FIXED "set_literal">

<!ELEMENT set-of-Pond_ornament
 (Pond_ornament*)>
<!ATTLIST set-of-Pond_ornament
      late-bound-element NMTOKEN #FIXED "set_literal">

<!ELEMENT string
 (#PCDATA)>
<!ATTLIST string
      late-bound-element NMTOKEN #FIXED "string_literal"
      width CDATA #IMPLIED >

<!ELEMENT Temperature
 (real)>
<!ATTLIST Temperature
      express_schema_name NMTOKEN #FIXED "support_items"
      express_type_name NMTOKEN #FIXED "temperature"
      late-bound-element NMTOKEN #FIXED "type_literal">

<!ELEMENT Temperature_range EMPTY>
<!ATTLIST Temperature_range
      express_entity_name NMTOKEN #FIXED "temperature_range"
      express_schema_name NMTOKEN #FIXED "support_items"
      id ID #IMPLIED
      late-bound-element NMTOKEN #FIXED "partial_entity_instance">

<!ELEMENT Temperature_range-ref EMPTY>
<!ATTLIST Temperature_range-ref
      refid IDREF #REQUIRED
      reftype CDATA #FIXED "refid (temperature_range | external_refid)"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      late-bound-element NMTOKEN #FIXED "entity_instance_ref">
```



```
<!ELEMENT time_stamp
 (#PCDATA)>
<!ATTLIST time_stamp
      late-bound-element NMTOKEN #FIXED "time_stamp">

<!ELEMENT true EMPTY>
<!ATTLIST true
      late-bound-element NMTOKEN #FIXED "true">

<!ELEMENT unknown EMPTY>
<!ATTLIST unknown
      late-bound-element NMTOKEN #FIXED "unknown">

<!ELEMENT unset EMPTY>
<!ATTLIST unset
      late-bound-element NMTOKEN #FIXED "unset">

<!ELEMENT Water_treatment_system
 (Water_treatment_system.capacity, Water_treatment_system-subtypes)>
<!ATTLIST Water_treatment_system
      express_entity_name NMTOKEN #FIXED "water_treatment_system"
      id ID #REQUIRED
      late-bound-element NMTOKEN #FIXED "entity_instance">

<!ELEMENT Water_treatment_system-ref EMPTY>
<!ATTLIST Water_treatment_system-ref
      refid IDREF #REQUIRED
      reftype CDATA #FIXED "refid (water_treatment_system | pumping_system
| filtration_system | external_refid)"
      late-bound-name CDATA #FIXED "reftype #DEFAULT"
      late-bound-element NMTOKEN #FIXED "entity_instance_ref">

<!ELEMENT Water_treatment_system-subtypes
 (Filtration_system | Pumping_system)+>

<!ELEMENT Water_treatment_system.capacity
 (Positive_integer)>
<!ATTLIST Water_treatment_system.capacity
      express_attribute_name NMTOKEN #FIXED "capacity"
      late-bound-element NMTOKEN #FIXED "attribute_instance">
```

H.4.2 Example data according to the EXPRESS-Typed Early Binding

The following XML presents the same data as presented in H.3 using the DTD given in the preceding subclause.

```
<?xml version="1.0"?>
<!DOCTYPE iso_10303_28 SYSTEM "mrjg-eteb-15june00-exp.dtd">
<iso_10303_28 representation_category="ETEB">
  <express_data id="ed001" representation_category="ETEB">
  <data_section_header>
```



```
      <documentation>This example file has been produced using a
combination of tools based on an original hand-crafted example file by
Robin La Fontaine on 21 March 2000. The DTD used was generated from the
Express model of Mr Jones's Garden according to the ETEB definition of
2000-06-07. </documentation>
  </data_section_header>
  <Mr_jones_garden-schema id="data-id1">
    <Garden id="g1">
      <Garden.has_pond><Fish_pond-ref refid="fp1"/>
      </Garden.has_pond>
      <Garden.has_greenhouse><Greenhouse-ref refid="gr1"/>
      </Garden.has_greenhouse>
      <Garden.climatic_temperature_range><Temperature_range-ref
refid="tr1"/>
      </Garden.climatic_temperature_range>
      <Garden.has_beds>
        <set-of-Bed><Bed-ref refid="bed1"/><Bed-ref refid="bed2"/>
          <Bed-ref refid="bed3"/><Bed-ref refid="bed4"/>
          <Bed-ref refid="bed5"/>
        </set-of-Bed>
      </Garden.has_beds>
      <Garden.neighbours_garden><Mr_smiths_garden-ref refid="sg1"/>
      </Garden.neighbours_garden>
    </Garden>
    <Livestock_containerPond id="po1">
      <Pond>
        <Pond.maintained_by><Water_treatment_system-ref refid="wts1"/>
        </Pond.maintained_by>
        <Pond.plants>
          <set-of-Aquatic_plant><Aquatic_plant-ref refid="aquatic_plant1"/>
          </set-of-Aquatic_plant>
        </Pond.plants>
        <Pond.water_volume>
          <Positive_integer>
            <integer>300</integer>
          </Positive_integer>
        </Pond.water_volume>
        <Pond.water_ph>
          <Ph>
            <real>3.5</real>
          </Ph>
        </Pond.water_ph>
      </Pond>
                  <Fish_pond id="fp1">
                   <Fish_pond.fish>
                               <set-of-Fish_type>
                                 <Fish_type><enumeration-
item>goldfish</enumeration-item></Fish_type>
                               </set-of-Fish_type>
                   </Fish_pond.fish>
                   <Fish_pond.fish_colours>
                               <set-of-Observed_fish_colour>
                                 <Observed_fish_colour>
                                  <Fish_colour><enumeration-
item>yellow</enumeration-item></Fish_colour>
```



```
                                    </Observed_fish_colour>
                                  </set-of-Observed_fish_colour>
                    </Fish_pond.fish_colours>
                  </Fish_pond>
                </Livestock_containerPond>
                <Plant id="plant1">
                  <Plant.colour>
                   <Flower_colour><enumeration-item>red</enumeration-
item></Flower_colour>
                  </Plant.colour>
                  <Plant.latin_name>
                   <Plant_name>
                                  <string>Lilium</string>
                   </Plant_name>
                  </Plant.latin_name>
                  <Plant.survival_temperature_range><Temperature_range-ref
refid="tr2"/>
                  </Plant.survival_temperature_range>
                  <Plant-subtypes>
                   <Aquatic_plant id="aquatic_plant1">
                                  <Aquatic_plant.aquatic_plant_type>
                                    <Aquatic_plant_types><enumeration-
item>lilly</enumeration-item></Aquatic_plant_types>
                                  </Aquatic_plant.aquatic_plant_type>

<Aquatic_plant.oxygen_volumetric_requirement>
                                    <Positive_integer>
                                     <integer>3</integer>
                                    </Positive_integer>

</Aquatic_plant.oxygen_volumetric_requirement>
                                  <Aquatic_plant.aquatic_plant_size>
                                    <Positive_integer>
                                     <integer>1</integer>
                                    </Positive_integer>
                                  </Aquatic_plant.aquatic_plant_size>
                    </Aquatic_plant>
                  </Plant-subtypes>
                </Plant>
                <Real_value_range id="rvr2">
                  <Real_value_range.minimum_value>
                   <real>-5.0</real>
                  </Real_value_range.minimum_value>
                  <Real_value_range.maximum_value>
                   <real>20.0</real>
                  </Real_value_range.maximum_value>
                  <Real_value_range-subtypes><Temperature_range id="tr2"/>
                  </Real_value_range-subtypes>
                </Real_value_range>
                <Bed id="bed1">
                  <Bed.acidity>
                   <Ph>
                                  <real>4.0</real>
                   </Ph>
                  </Bed.acidity>
```



```
                  <Bed.holds_plants>
                   <set-of-Outdoors_plant><Outdoors_plant-ref
                               refid="op1"/><Outdoors_plant-ref
refid="op2"/>
                   </set-of-Outdoors_plant>
                  </Bed.holds_plants>
                  <Bed.planting_plan>
                   <array-of-array-of-optional-Outdoors_plant>
                               <array-of-optional-
Outdoors_plant><Outdoors_plant-ref
                                 refid="op1"/><Outdoors_plant-ref
refid="op1"/><Outdoors_plant-ref
                                 refid="op1"/><Outdoors_plant-ref
refid="op1"/><Outdoors_plant-ref
                                 refid="op1"/><Outdoors_plant-ref
refid="op1"/><Outdoors_plant-ref
                                 refid="op1"/><Outdoors_plant-ref
refid="op1"/><Outdoors_plant-ref
                                 refid="op1"/><Outdoors_plant-ref
refid="op1"/>
                               </array-of-optional-Outdoors_plant>
                               <array-of-optional-
Outdoors_plant><Outdoors_plant-ref
                                 refid="op2"/><Outdoors_plant-ref
refid="op2"/><Outdoors_plant-ref
                                 refid="op2"/><Outdoors_plant-ref
refid="op2"/><Outdoors_plant-ref
                                 refid="op2"/><Outdoors_plant-ref
refid="op2"/><Outdoors_plant-ref
                                 refid="op2"/><Outdoors_plant-ref
refid="op2"/><Outdoors_plant-ref
                                 refid="op2"/><Outdoors_plant-ref
refid="op2"/>
                               </array-of-optional-Outdoors_plant>
                               <array-of-optional-Outdoors_plant>

<unset/><unset/><unset/><unset/><unset/><unset/><unset/><unset/><unset/><un
set/>

                               </array-of-optional-Outdoors_plant>
                               <array-of-optional-
Outdoors_plant><Outdoors_plant-ref
                                 refid="op1"/><Outdoors_plant-ref
refid="op1"/><Outdoors_plant-ref
                                 refid="op1"/><Outdoors_plant-ref
refid="op1"/><Outdoors_plant-ref
                                 refid="op1"/><Outdoors_plant-ref
refid="op1"/><Outdoors_plant-ref
                                 refid="op1"/><Outdoors_plant-ref
refid="op1"/><Outdoors_plant-ref
                                 refid="op1"/><Outdoors_plant-ref
refid="op1"/>
                               </array-of-optional-Outdoors_plant>
                               <array-of-optional-
Outdoors_plant><Outdoors_plant-ref
```



```
                                refid="op1"/><Outdoors_plant-ref
refid="op1"/><Outdoors_plant-ref
                                refid="op1"/><Outdoors_plant-ref
refid="op1"/><Outdoors_plant-ref
                                refid="op1"/><Outdoors_plant-ref
refid="op1"/><Outdoors_plant-ref
                                refid="op1"/><Outdoors_plant-ref
refid="op1"/><Outdoors_plant-ref
                                refid="op1"/><Outdoors_plant-ref
refid="op1"/>
                            </array-of-optional-Outdoors_plant>
                     </array-of-array-of-optional-Outdoors_plant>
                   </Bed.planting_plan>
                   <Bed.flowering_order>
                    <list-of-Outdoors_plant><Outdoors_plant-ref
                                refid="op2"/><Outdoors_plant-ref
refid="op1"/>
                    </list-of-Outdoors_plant>
                   </Bed.flowering_order>
                 </Bed>
                 <Bed id="bed2">
                   <Bed.acidity>
                    <Ph>
                                <real>4.2</real>
                    </Ph>
                   </Bed.acidity>
                   <Bed.holds_plants>
                    <set-of-Outdoors_plant><Outdoors_plant-ref
                                refid="op1"/><Outdoors_plant-ref
refid="op2"/>
                    </set-of-Outdoors_plant>
                   </Bed.holds_plants>
                   <Bed.flowering_order>
                    <list-of-Outdoors_plant><Outdoors_plant-ref
                                refid="op2"/><Outdoors_plant-ref
refid="op1"/>
                    </list-of-Outdoors_plant>
                   </Bed.flowering_order>
                 </Bed>
                 <Bed id="bed3">
                   <Bed.acidity>
                    <Ph>
                                <real>4.3</real>
                    </Ph>
                   </Bed.acidity>
                   <Bed.holds_plants>
                    <set-of-Outdoors_plant><Outdoors_plant-ref
                                refid="op1"/><Outdoors_plant-ref
refid="op2"/>
                    </set-of-Outdoors_plant>
                   </Bed.holds_plants>
                   <Bed.flowering_order>
                    <list-of-Outdoors_plant><Outdoors_plant-ref
                                refid="op2"/><Outdoors_plant-ref
refid="op1"/>
```



```
                    </list-of-Outdoors_plant>
                  </Bed.flowering_order>
                </Bed>
                <Bed id="bed4">
                  <Bed.acidity>
                   <Ph>
                              <real>4.4</real>
                   </Ph>
                  </Bed.acidity>
                  <Bed.holds_plants>
                   <set-of-Outdoors_plant><Outdoors_plant-ref
                              refid="op1"/><Outdoors_plant-ref
refid="op2"/>
                   </set-of-Outdoors_plant>
                  </Bed.holds_plants>
                  <Bed.flowering_order>
                   <list-of-Outdoors_plant><Outdoors_plant-ref
                              refid="op2"/><Outdoors_plant-ref
refid="op1"/>
                   </list-of-Outdoors_plant>
                  </Bed.flowering_order>
                </Bed>
                <Bed id="bed5">
                  <Bed.acidity>
                   <Ph>
                              <real>4.5</real>
                   </Ph>
                  </Bed.acidity>
                  <Bed.holds_plants>
                   <set-of-Outdoors_plant><Outdoors_plant-ref
                              refid="op1"/><Outdoors_plant-ref
refid="op2"/>
                   </set-of-Outdoors_plant>
                  </Bed.holds_plants>
                  <Bed.flowering_order>
                   <list-of-Outdoors_plant><Outdoors_plant-ref
                              refid="op2"/><Outdoors_plant-ref
refid="op1"/>
                   </list-of-Outdoors_plant>
                  </Bed.flowering_order>
                </Bed>
                <Plant id="p1">
                  <Plant.colour>
                   <Flower_colour><enumeration-item>red</enumeration-
item></Flower_colour>
                  </Plant.colour>
                  <Plant.latin_name>
                   <Plant_name>
                              <string>rosa</string>
                   </Plant_name>
                  </Plant.latin_name>
                  <Plant.english_names>
                   <set-of-Plant_name>
                              <Plant_name>
                                <string>rose</string>
```



```
                              </Plant_name>
                    </set-of-Plant_name>
                  </Plant.english_names>
                  <Plant.survival_temperature_range><Temperature_range-ref
refid="tr3"/>
                  </Plant.survival_temperature_range>
                  <Plant-subtypes>
                    <Outdoors_plant id="op1">

<Outdoors_plant.survival_ph_range><Ph_range-ref refid="phr1"/>
                              </Outdoors_plant.survival_ph_range>
                    </Outdoors_plant>
                  </Plant-subtypes>
                </Plant>
                <Plant id="p2">
                  <Plant.colour>
                    <Flower_colour><enumeration-item>yellow</enumeration-
item></Flower_colour>
                  </Plant.colour>
                  <Plant.latin_name>
                    <Plant_name>
                              <string>Narcissus</string>
                    </Plant_name>
                  </Plant.latin_name>
                  <Plant.english_names>
                    <set-of-Plant_name>
                              <Plant_name>
                                <string>daffodil</string>
                              </Plant_name>
                    </set-of-Plant_name>
                  </Plant.english_names>
                  <Plant.survival_temperature_range><Temperature_range-ref
refid="tr3"/>
                  </Plant.survival_temperature_range>
                  <Plant-subtypes>
                    <Outdoors_plant id="op2">

<Outdoors_plant.survival_ph_range><Ph_range-ref refid="phr1"/>
                              </Outdoors_plant.survival_ph_range>
                    </Outdoors_plant>
                  </Plant-subtypes>
                </Plant>
                <Greenhouse id="gr1">

<Greenhouse.enforced_temperature_range><Temperature_range-ref
                    refid="tr1"/>
                  </Greenhouse.enforced_temperature_range>
                  <Greenhouse.holds_plants>
                    <set-of-Greenhouse_plant><Greenhouse_plant-ref
refid="grp1"/>
                    </set-of-Greenhouse_plant>
                  </Greenhouse.holds_plants>
                </Greenhouse>
                <Plant id="p-grp1">
                  <Plant.colour>
```



```
                    <Flower_colour><enumeration-item>red</enumeration-
item></Flower_colour>
                  </Plant.colour>
                  <Plant.latin_name>
                   <Plant_name>
                               <string>orchidaceae</string>
                   </Plant_name>
                  </Plant.latin_name>
                  <Plant.english_names>
                   <set-of-Plant_name>
                               <Plant_name>
                                 <string>orchid</string>
                               </Plant_name>
                   </set-of-Plant_name>
                  </Plant.english_names>
                  <Plant.survival_temperature_range><Temperature_range-ref
refid="tr1"/>
                  </Plant.survival_temperature_range>
                  <Plant-subtypes><Greenhouse_plant id="grp1"/>
                  </Plant-subtypes>
                </Plant>
                <Real_value_range id="rvr1">
                  <Real_value_range.minimum_value>
                   <real>3.5</real>
                  </Real_value_range.minimum_value>
                  <Real_value_range.maximum_value>
                   <real>27.0</real>
                  </Real_value_range.maximum_value>
                  <Real_value_range-subtypes><Temperature_range id="tr1"/>
                  </Real_value_range-subtypes>
                </Real_value_range>
                <Real_value_range id="rvr3">
                  <Real_value_range.minimum_value>
                   <real>-3.0</real>
                  </Real_value_range.minimum_value>
                  <Real_value_range.maximum_value>
                   <real>33.0</real>
                  </Real_value_range.maximum_value>
                  <Real_value_range-subtypes><Temperature_range id="tr3"/>
                  </Real_value_range-subtypes>
                </Real_value_range>
                <Real_value_range id="rvr-phr1">
                  <Real_value_range.minimum_value>
                   <real>3.1</real>
                  </Real_value_range.minimum_value>
                  <Real_value_range.maximum_value>
                   <real>5.0</real>
                  </Real_value_range.maximum_value>
                  <Real_value_range-subtypes><Ph_range id="phr1"/>
                  </Real_value_range-subtypes>
                </Real_value_range>
                <Mr_smiths_garden id="sg1">
                  <Mr_smiths_garden.has_bed><Mr_smiths_garden-schema.bed-
ref
                   refid="sbed1"/>
```



```
                  </Mr_smiths_garden.has_bed>
                </Mr_smiths_garden>
                <Mr_smiths_garden-schema.bed id="sbed1">
                  <Mr_smiths_garden-schema.bed.description>
                   <string>Mr Smith's flower bed.</string>
                  </Mr_smiths_garden-schema.bed.description>
                </Mr_smiths_garden-schema.bed>
                <Mr_smiths_plant id="splant1">
                  <Mr_smiths_plant.name>
                 <string>rose</string>
                  </Mr_smiths_plant.name>
                </Mr_smiths_plant>
                <Water_treatment_system id="wts1">
                  <Water_treatment_system.capacity>
                   <Positive_integer>
                               <integer>500</integer>
                   </Positive_integer>
                  </Water_treatment_system.capacity>
                  <Water_treatment_system-subtypes>
                   <Pumping_system>
                               <Pumping_system.pumping_efficiency>
                                 <Efficiency><Efficiency_range-ref
refid="efr1"/>
                                 </Efficiency>
                               </Pumping_system.pumping_efficiency>
                               <Pumping_system.number_of_pumps>
                                 <Positive_integer>
                                  <integer>1</integer>
                                 </Positive_integer>
                               </Pumping_system.number_of_pumps>
                   </Pumping_system>
                  </Water_treatment_system-subtypes>
                </Water_treatment_system>
                <Real_value_range id="rvr-efr1">
                  <Real_value_range.minimum_value>
                   <real>75</real>
                  </Real_value_range.minimum_value>
                  <Real_value_range.maximum_value>
                   <real>85</real>
                  </Real_value_range.maximum_value>
                  <Real_value_range-subtypes><Efficiency_range id="efr1"/>
                  </Real_value_range-subtypes>
                </Real_value_range>
  </Mr_jones_garden-schema>
  </express_data>
</iso_10303_28>
```

H.5 Examples using the Object Serialization Early Binding

This subclause presents an example markup declaration set and example data set, corresponding to the schemas presented in H.1, represented in XML as specified in clause 9.

H.5.1 Example Object Serialization Early Binding markup declaration set

This subclause presents a markup declaration set corresponding to the schema from H.1 based on the rules specified in clause 9.

```
<!NOTATION base64 PUBLIC "base-64 encoded data">
<!NOTATION hex PUBLIC "hexadecimal digits">

<!ENTITY % schema_decl "">
<!ENTITY % schema_instance "osb:uos">

<!ELEMENT iso_10303_28 (iso_10303_28_header?,(express_schema |
express_data)+)>
<!ATTLIST iso_10303_28
      version CDATA #FIXED "PDTS"
      representation_category NMTOKENS #REQUIRED>

<!ELEMENT iso_10303_28_header
 (document_name, purpose?, time_stamp?, author?, originating_organization?,
authorization?, originating_system?, preprocessor_version?,
documentation?)>

<!ELEMENT document_name (#PCDATA)>
<!ELEMENT time_stamp (#PCDATA)>
<!ELEMENT author  (#PCDATA)>
<!ELEMENT originating_organization   (#PCDATA)>
<!ELEMENT authorization (#PCDATA)>
<!ELEMENT originating_system (#PCDATA)>
<!ELEMENT preprocessor_version  (#PCDATA)>
<!ELEMENT purpose (#PCDATA)>
<!ELEMENT documentation (#PCDATA)>

<!ELEMENT express_schema (%schema_decl; schema_text | external_refid)>
   <!ATTLIST express_schema
      id ID #IMPLIED
      express_schema_description CDATA #IMPLIED
      express_schema_identifier CDATA #IMPLIED
      express_schema_version CDATA #IMPLIED >

   <!ELEMENT schema_text (#PCDATA)>

<!ELEMENT express_data
 (data_section_header?, %schema_instance;)>
<!ATTLIST express_data
    id ID #REQUIRED
    representation_category NMTOKEN #IMPLIED
    name CDATA #IMPLIED >

<!ELEMENT data_section_header
 (documentation?)>

<!ELEMENT external_refid EMPTY>
<!ATTLIST external_refid
  id ID #REQUIRED
```



```
  xlink:type CDATA #FIXED 'simple'
  xlink:href CDATA #REQUIRED
  xlink:arcrole CDATA #REQUIRED
  xlink:title CDATA #IMPLIED
  xlink:role CDATA #IMPLIED
  xlink:show CDATA #IMPLIED
  xlink:actuate CDATA #IMPLIED
>

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

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

<!ELEMENT osb:edokey (osb:this?,osb:lockey*, osb:go*) >
<!ATTLIST osb: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 osb:this (#PCDATA) >
<!ATTLIST osb:this
 xlink:type (resource)       #REQUIRED
 xlink:title  CDATA  #REQUIRED
 xlink:role   CDATA  #IMPLIED>

<!ELEMENT osb:go EMPTY>
<!ATTLIST osb: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>

<!ELEMENT osb:ctn EMPTY >
<!ATTLIST osb:ctn
 x-id ID  #REQUIRED
```



```
  ctype CDATA  #REQUIRED
  c IDREFS #REQUIRED>

<!ELEMENT osb:ectn EMPTY >
<!ATTLIST osb:ectn
  x-id ID  #REQUIRED
  c CDATA          #IMPLIED>

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

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

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

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

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

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

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

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

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

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



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

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

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

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

<!ELEMENT osb:unset EMPTY>
<!ATTLIST osb:unset
  x-idID #REQUIRED >

<!ELEMENT mjg:Amphibian EMPTY>
<!ATTLIST mjg:Amphibian
   x-id   ID   #REQUIRED
   val   ( frog | newt )   #REQUIRED>

<!ELEMENT mjg:Aquatic_plant_types EMPTY>
<!ATTLIST mjg:Aquatic_plant_types
   x-id   ID   #REQUIRED
   val   ( lilly | weed | lotus )   #REQUIRED>

<!ELEMENT mjg:Fish_colour EMPTY>
<!ATTLIST mjg:Fish_colour
   x-id   ID   #REQUIRED
   val   ( yellow | black | silver | blue )   #REQUIRED>

<!ELEMENT mjg:Fish_type EMPTY>
<!ATTLIST mjg:Fish_type
   x-id   ID   #REQUIRED
   val   ( koi | goldfish )   #REQUIRED>

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

<!ELEMENT mjg:Multi_coloured_fish_colour EMPTY>
<!ATTLIST mjg:Multi_coloured_fish_colour
   x-id   ID   #REQUIRED
   val-r   IDREF   #REQUIRED>

<!ELEMENT mjg:Observed_fish_colour EMPTY>
<!ATTLIST mjg:Observed_fish_colour
   x-id   ID   #REQUIRED
   utype   NMTOKEN   #REQUIRED
```



```
   val-r   IDREF   #REQUIRED>

<!ELEMENT mjg:Pond_ornament EMPTY>
<!ATTLIST mjg:Pond_ornament
   x-id   ID   #REQUIRED
   val   ( waterfall | fountain | bridge )   #REQUIRED>

<!ELEMENT mjg:Amphibian_pond EMPTY>
<!ATTLIST mjg:Amphibian_pond
   x-id   ID   #REQUIRED
   Amphibians-r   IDREF   #REQUIRED
   Pond.Maintained_by-r   IDREF   #IMPLIED
   Plants-r   IDREF   #REQUIRED
   Water_ph   CDATA   #REQUIRED
   Water_volume   CDATA   #REQUIRED
   Livestock_count   CDATA   #IMPLIED
   Livestock_container.Maintained_by-r   IDREF   #IMPLIED>

<!ELEMENT mjg:Amphibian_pondOrnamental_pond  EMPTY>
<!ATTLIST mjg:Amphibian_pondOrnamental_pond
  x-id   ID  #REQUIRED
  Pond.Maintained_by-r IDREF  #IMPLIED
  Plants-r  IDREF  #REQUIRED
  Water_volume CDATA  #REQUIRED
  Water_ph  CDATA  #REQUIRED
  Amphibians-r IDREF  #REQUIRED
  Ornaments-r  IDREF  #REQUIRED
  Livestock_count     CDATA  #IMPLIED
  Livestock_container.Maintained_by-r IDREF  #IMPLIED>

<!ELEMENT mjg:Aquatic_plant EMPTY>
<!ATTLIST mjg:Aquatic_plant
   x-id   ID   #REQUIRED
   Aquatic_plant_size   CDATA   #REQUIRED
   Aquatic_plant_type     (lilly | weed | lotus)    #REQUIRED
   Oxygen_volumetric_requirement   CDATA   #REQUIRED
   Colour     (red | yellow | white)    #REQUIRED
   English_names-r   IDREF   #IMPLIED
   Latin_name-r   IDREF   #REQUIRED
   Survival_temperature_range-r   IDREF   #REQUIRED>

<!ELEMENT mjg:Bed EMPTY>
<!ATTLIST mjg:Bed
   x-id   ID   #REQUIRED
   Acidity   CDATA   #REQUIRED
   Flowering_order-r   IDREF   #IMPLIED
   Holds_plants-r   IDREF   #REQUIRED
   Planting_plan-r   IDREF   #IMPLIED
   Plants_ready_for_planting-r   IDREF   #IMPLIED
   I-The_garden-r   IDREFS   #IMPLIED
   U-Every_bed_has_a_different_acidity   ( true | false | unknown )
#IMPLIED>

<!ELEMENT mjg:Filtration_system EMPTY>
<!ATTLIST mjg:Filtration_system
```



```
   x-id   ID   #REQUIRED
   Filtration_efficiency-r   IDREF   #REQUIRED
   Number_of_filters   CDATA   #REQUIRED
   Capacity   CDATA   #REQUIRED>

<!ELEMENT mjg:Filtration_systemPumping_system  EMPTY >
<!ATTLIST mjg:Filtration_systemPumping_system
  x-id     ID  #REQUIRED
  Capacity   CDATA  #REQUIRED
  Filtration_efficiency-r         IDREF   #REQUIRED
  Number_of_filters   CDATA  #REQUIRED
  Pumping_efficiency-r   IDREF  #REQUIRED
  Number_of_pumps    CDATA  #REQUIRED>

<!ELEMENT mjg:Fish_pond EMPTY>
<!ATTLIST mjg:Fish_pond
   x-id   ID   #REQUIRED
   Fish-r   IDREF   #REQUIRED
   Fish_colours-r   IDREF   #REQUIRED
   Pond.Maintained_by-r   IDREF   #IMPLIED
   Plants-r   IDREF   #REQUIRED
   Water_ph   CDATA   #REQUIRED
   Water_volume   CDATA   #REQUIRED
   Livestock_count   CDATA   #IMPLIED
   Livestock_container.Maintained_by-r   IDREF   #IMPLIED>

<!ELEMENT mjg:Fish_pondOrnamental_pond  EMPTY>
<!ATTLIST mjg:Fish_pondOrnamental_pond
  x-id   ID  #REQUIRED
  Pond.Maintained_by-r  IDREF  #IMPLIED
  Plants-r  IDREF  #REQUIRED
  Water_volume         CDATA   #REQUIRED
  Water_ph  CDATA  #REQUIRED
  Fish-r    IDREF  #REQUIRED
  Fish_colours-r  IDREF  #REQUIRED
  Ornaments-r  IDREF  #REQUIRED
  Livestock_count     CDATA  #IMPLIED
  Livestock_container.Maintained_by-r IDREF  #IMPLIED>

<!ELEMENT mjg:Garden EMPTY>
<!ATTLIST mjg:Garden
   x-id   ID   #REQUIRED
   Climatic_temperature_range-r   IDREF   #REQUIRED
   Has_beds-r   IDREF   #REQUIRED
   Has_greenhouse-r   IDREF   #REQUIRED
   Has_pond-r   IDREF   #REQUIRED
   Neighbours_garden-r   IDREF   #REQUIRED>

<!ELEMENT mjg:Greenhouse EMPTY>
<!ATTLIST mjg:Greenhouse
   x-id   ID   #REQUIRED
   Enforced_temperature_range-r   IDREF   #REQUIRED
   Holds_plants-r   IDREF   #REQUIRED
   I-The_garden-r   IDREFS   #IMPLIED>
```



```
<!ELEMENT mjg:Greenhouse_plant EMPTY>
<!ATTLIST mjg:Greenhouse_plant
   x-id   ID   #REQUIRED
   Colour     (red | yellow | white)    #REQUIRED
   English_names-r   IDREF   #IMPLIED
   Latin_name-r   IDREF   #REQUIRED
   Survival_temperature_range-r   IDREF   #REQUIRED
   I-The_greenhouse-r   IDREFS   #IMPLIED
   W-R1   ( true | false | unknown )   #IMPLIED
   W-R2   ( true | false | unknown )   #IMPLIED>

<!ELEMENT mjg:Ornamental_pond EMPTY>
<!ATTLIST mjg:Ornamental_pond
   x-id   ID   #REQUIRED
   Ornaments-r   IDREF   #REQUIRED
   Maintained_by-r   IDREF   #IMPLIED
   Plants-r   IDREF   #REQUIRED
   Water_ph   CDATA   #REQUIRED
   Water_volume   CDATA   #REQUIRED>

<!ELEMENT mjg:Outdoors_plant EMPTY>
<!ATTLIST mjg:Outdoors_plant
   x-id   ID   #REQUIRED
   Survival_ph_range-r   IDREF   #REQUIRED
   Colour     (red | yellow | white)    #REQUIRED
   English_names-r   IDREF   #IMPLIED
   Latin_name-r   IDREF   #REQUIRED
   Survival_temperature_range-r   IDREF   #REQUIRED
   I-The_beds-r   IDREFS   #IMPLIED
   W-R1   ( true | false | unknown )   #IMPLIED
   W-R2   ( true | false | unknown )   #IMPLIED>

<!ELEMENT mjg:Pumping_system EMPTY>
<!ATTLIST mjg:Pumping_system
   x-id   ID   #REQUIRED
   Number_of_pumps   CDATA   #REQUIRED
   Pumping_efficiency-r   IDREF   #REQUIRED
   Capacity   CDATA   #REQUIRED>

<!ELEMENT msg:Bed EMPTY>
<!ATTLIST msg:Bed
   x-id   ID   #REQUIRED
   Description-r   IDREF   #REQUIRED
   I-The_garden-r    IDREFS   #IMPLIED>

<!ELEMENT msg:Flower_plant EMPTY>
<!ATTLIST msg:Flower_plant
   x-id   ID   #REQUIRED
   Flower_colour-r   IDREF   #REQUIRED
   Name-r   IDREF   #REQUIRED>

<!ELEMENT msg:Garden EMPTY>
<!ATTLIST msg:Garden
   x-id   ID   #REQUIRED
   Has_bed-r   IDREF   #REQUIRED>
```



```
<!ELEMENT mjg:Mr_smiths_plant EMPTY>
<!ATTLIST mjg:Mr_smiths_plant
   x-id   ID   #REQUIRED
   Name-r   IDREF   #REQUIRED>

<!ELEMENT msg:Vegetable_plant EMPTY>
<!ATTLIST msg:Vegetable_plant
   x-id   ID   #REQUIRED
   Vegetable_type-r   IDREF   #REQUIRED
   Name-r   IDREF   #REQUIRED>

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

<!ELEMENT si:Efficiency_measure EMPTY>
<!ATTLIST si:Efficiency_measure
   x-id   ID   #REQUIRED
   utype   NMTOKEN   #REQUIRED
   val-r   IDREF   #REQUIRED>

<!ELEMENT si:Fractional_value EMPTY>
<!ATTLIST si:Fractional_value
   x-id   ID   #REQUIRED
   val   CDATA   #REQUIRED>

<!ELEMENT si:Percentage EMPTY>
<!ATTLIST si:Percentage
   x-id   ID   #REQUIRED
   val   CDATA   #REQUIRED>

<!ELEMENT si:Ph EMPTY>
<!ATTLIST si:Ph
   x-id   ID   #REQUIRED
   val   CDATA   #REQUIRED>

<!ELEMENT si:Plant_name (#PCDATA)>
<!ATTLIST si:Plant_name
   x-id   ID   #REQUIRED>

<!ELEMENT si:Positive_integer EMPTY>
<!ATTLIST si:Positive_integer
   x-id   ID   #REQUIRED
   val   CDATA   #REQUIRED>

<!ELEMENT si:Temperature EMPTY>
<!ATTLIST si:Temperature
   x-id   ID   #REQUIRED
   val   CDATA   #REQUIRED>

<!ELEMENT si:Efficiency_range EMPTY>
<!ATTLIST si:Efficiency_range
```



```
   x-id   ID   #REQUIRED
   Maximum_value   CDATA   #REQUIRED
   Minimum_value   CDATA   #REQUIRED>

<!ELEMENT si:Ph_range EMPTY>
<!ATTLIST si:Ph_range
   x-id   ID   #REQUIRED
   Maximum_value   CDATA   #REQUIRED
   Minimum_value   CDATA   #REQUIRED>

<!ELEMENT si:Temperature_range EMPTY>
<!ATTLIST si:Temperature_range
   x-id   ID   #REQUIRED
   Maximum_value   CDATA   #REQUIRED
   Minimum_value   CDATA   #REQUIRED>
```

H.5.2 Example Object Serialization Early Binding data set

The following markup presents the same data as presented in H.3 using the markup declarations from H.5.1.

```
<?xml version = "1.0"?>
<!DOCTYPE iso_10303_28 SYSTEM "mjg.dtd">
<iso_10303_28 representation_category="OSEB" xmlns:osb="urn:10303-28:oseb"
xmlns:mjg="urn:10303-28:oseb/Mr_jones_garden"
  xmlns:si="urn:10303-28:oseb/Support_items" xmlns:msg="urn:10303-
28:oseb/Mr_smiths_garden">
 <iso_10303_28_header>
    <document_name> OSEB MJG Example </document_name>
    <purpose>Example OSEB representation of Mr Jones Garden EXPRESS
schema</purpose>
           <time_stamp>2000-09-11 1200 UCT</time_stamp>
           <author>J.Trausch</author>
    <originating_organization>Electric Boat</originating_organization>
    <authorization>Approved by D405</authorization>
    <originating_system>ByHand</originating_system>
    <preprocessor_version>1.0</preprocessor_version>
 </iso_10303_28_header>
     <express_data id="mjg" representation_category="OSEB">
 <osb:uos c="g1 splant1" unset="unset" schema="mr_jones_garden">
  <mjg:Garden
    x-id="g1"
    Has_pond-r="po1"
    Has_greenhouse-r="gh1"
    Climatic_temperature_range-r="tr1"
    Has_beds-r="bed_set1"
    Neighbours_garden-r="sg1"/>

  <mjg:Fish_pond x-id="po1" Pond.Maintained_by-r="fs1" Plants-
r="plant_set1" Water_volume="4000" Water_ph="6.8" Fish-r="ect1"
Fish_colours-r="ofc_set1" />
```



```
   <mjg:Greenhouse x-id="gh1" Enforced_temperature_range-r="tr2"
    Holds_plants-r="plant_set2" I-The_garden-r="g1"/>

   <osb:ctn x-id="bed_set1"  ctype="mjg:Bed[]" c="bed1 bed2 bed4 bed3
bed5"/>
   <osb:ctn x-id="ofc_set1"  ctype="mjg:Observed_fish_colour[]" c="ofc1
ofc2 ofc3"/>
   <osb:ctn x-id="plant_set1" ctype="mjg:Aquatic_plant[]" c="ap1 ap2"/>
   <osb:ctn x-id="plant_set2" ctype="mjg:Greenhouse_plant[]" c="gp1 gp2"/>
   <osb:ctn x-id="plant_set3" ctype="mjg:Outdoor_plant[]" c="op1 op2"/>
   <osb:ctn x-id="en1" ctype="osb:string[]" c="str1"/>
   <osb:ctn x-id="en4" ctype="osb:string[]" c="str4 str5"/>
   <osb:ctn x-id="en5" ctype="osb:string[]" c="str4 str6"/>
   <osb:ctn x-id="en6" ctype="osb:string[]" c="str7"/>
   <osb:ctn x-id="en7" ctype="osb:string[]" c="str8"/>
   <osb:ctn x-id="pp1"  ctype="mjg:ctn[][]" c="ar1 ar2 ar3 ar4 ar5"/>
   <osb:ctn x-id="ar1"  ctype="mjg:Outdoors_plant[]" c="op1 op1 op1 op1 op1
op1 op1 op1 op1 op1"/>
   <osb:ctn x-id="ar2"  ctype="mjg:Outdoors_plant[]" c="op2 op2 op2 op2 op2
op2 op2 op2 op2 op2"/>
   <osb:ctn x-id="ar3"  ctype="mjg:Outdoors_plant[]" c="unset unset unset
unset unset unset unset unset unset unset"/>
   <osb:ctn x-id="ar4"  ctype="mjg:Outdoors_plant[]" c="op1 op1 op1 op1 op1
op1 op1 op1 op1 op1"/>
   <osb:ctn x-id="ar5"  ctype="mjg:Outdoors_plant[]" c="op1 op1 op1 op1 op1
op1 op1 op1 op1 op1"/>
   <osb:ctn x-id="fo1"  ctype="mjg:Outdoors_plant[]" c="op2 op1"/>
   <osb:ectn x-id="ect1" c="koi goldfish"/>
   <osb:ectn x-id="ect2" c="silver blue yellow"/>

   <mjg:Bed x-id="bed1" Acidity="6.1" Holds_plants-r="plant_set3"
Planting_plan-r="pp1" Flowering_order-r="fo1" I-The_garden-r="g1"/>
   <mjg:Bed x-id="bed2" Acidity="6.2" Holds_plants-r="plant_set3"
Flowering_order-r="fo1" I-The_garden-r="g1"/>
   <mjg:Bed x-id="bed3" Acidity="6.3" Holds_plants-r="plant_set3"
Flowering_order-r="fo1" I-The_garden-r="g1"/>
   <mjg:Bed x-id="bed4" Acidity="6.4" Holds_plants-r="plant_set3"
Flowering_order-r="fo1" I-The_garden-r="g1"/>
   <mjg:Bed x-id="bed5" Acidity="6.5" Holds_plants-r="plant_set3"
Flowering_order-r="fo1" I-The_garden-r="g1"/>

   <mjg:Outdoors_plant x-id="op1" Colour="red" Latin_name-r="str12"
English_names-r="en1"   Survival_temperature_range-r="tr3"
Survival_ph_range-r="phr1" I-The_beds-r="bed1 bed2 bed3 bed4 bed5"/>
   <mjg:Outdoors_plant x-id="op2" Colour="yellow" Latin_name-r="str12"
English_names-r="en1"   Survival_temperature_range-r="tr3"
Survival_ph_range-r="phr1" I-The_beds-r="bed1 bed2 bed3 bed4 bed5"/>

   <mjg:Aquatic_plant x-id="ap1" Colour="white" Latin_name-r="str16"
English_names-r="en4"   Survival_temperature_range-r="tr4"
Aquatic_plant_type="lilly" Oxygen_volumetric_requirement="72"
 Aquatic_plant_size="20"/>
   <mjg:Aquatic_plant x-id="ap2" Colour="yellow" Latin_name-r="str16"
English_names-r="en5"   Survival_temperature_range-r="tr4"
```



```
Aquatic_plant_type="lilly" Oxygen_volumetric_requirement="72"
  Aquatic_plant_size="20"/>

   <mjg:Greenhouse_plant x-id="gp1" Colour="yellow" Latin_name-r="str17"
English_names-r="en6"   Survival_temperature_range-r="tr5" I-
The_greenhouse-r="gh1"/>
   <mjg:Greenhouse_plant x-id="gp2" Colour="red" Latin_name-r="str18"
English_names-r="en7"   Survival_temperature_range-r="tr5" I-
The_greenhouse-r="gh1"/>

   <si:Temperature_range x-id="tr1" Minimum_value="60" Maximum_value="90"/>
   <si:Temperature_range x-id="tr2" Minimum_value="70" Maximum_value="92"/>
   <si:Temperature_range x-id="tr3" Minimum_value="65" Maximum_value="85"/>
   <si:Temperature_range x-id="tr4" Minimum_value="65" Maximum_value="90"/>
   <si:Temperature_range x-id="tr5" Minimum_value="75" Maximum_value="90"/>
   <si:Efficiency_range x-id="efr1" Minimum_value="75" Maximum_value="85"/>

   <si:Ph_range x-id="phr1" Minimum_value="6.0" Maximum_value="8.0"/>

   <mjg:Filtration_system x-id="fs1" Capacity="5000" Filtration_efficiency-
r="eff1" Number_of_filters="4"/>

   <si:Efficiency x-id="eff1" utype="si:Efficiency_range" val-r="efr1"/>

   <mjg:Observed_fish_colour x-id="ofc1" utype="mjg:Fish_colour" val-
r="fic1"/>
   <mjg:Observed_fish_colour x-id="ofc2" utype="mjg:Fish_colour" val-
r="fic2"/>
   <mjg:Observed_fish_colour x-id="ofc3"
utype="mjg:Multi_coloured_fish_colour" val-r="mcfc1"/>

   <osb:string x-id="str1">rose</osb:string>
   <osb:string x-id="str12">rosa</osb:string>
   <osb:string x-id="str4">water lilly</osb:string>
   <osb:string x-id="str5">Fairy lilly</osb:string>
   <osb:string x-id="str6">Calla lillies</osb:string>
   <osb:string x-id="str16">Zephyranthis</osb:string>
   <osb:string x-id="str7">Violets</osb:string>
   <osb:string x-id="str17">Viola pedata</osb:string>
   <osb:string x-id="str8">Pansies</osb:string>
   <osb:string x-id="str18">Viola sp</osb:string>

   <mjg:Fish_colour x-id="fic1" val="yellow"/>
   <mjg:Fish_colour x-id="fic2" val="black"/>
   <mjg:Multi_coloured_fish_colour x-id="mcfc1" val-r="ect2"/>

   <si:Percentage x-id="d1" val="88" />

   <osb:unset x-id="unset" />

   <msg:Garden
     x-id="sg1"
     Has_bed-r="msbed1"/>

   <msg:Bed
```



```
    x-id="msbed1"
    Description-r="ids1"
    I-The_garden-r="sg1" />

   <osb:string x-id="ids1">Mr. Smith's flower bed</osb:string>

   <mjg:Mr_smiths_plant
    x-id="splant1"
    Name-r="str1" />

   </osb:uos>
 </express_data>
</iso_10303_28>
```

H.5.3 Example XSLT stylesheet to map an Object Serialization Early Binding to the Late Bound

```
<?xml version="1.0" ?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
    xmlns:osb="urn:10303-28:oseb"
    xmlns:mjg="urn:10303-28:oseb/Mr_jones_garden"
    xmlns:si="urn:10303-28:oseb/Support_items"
                xmlns:msg="urn:10303-28:oseb/Mr_smiths_garden"
                xmlns:xlink="http://www.w3.org/1999/xlink">

 <xsl:output omit-xml-declaration="yes" indent="yes" />

 <!--
   <xsl:strip-space elements="*"/>
 -->

 <xsl:variable name="lschema">mr_jones_garden</xsl:variable>
  <xsl:variable name="refschema1">support_items</xsl:variable>
  <xsl:variable name="refschema2">mr_smiths_garden</xsl:variable>

  <xsl:template match="/">
   <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="osb:uos">
 <schema_instance express_schema_name="{@schema}">
   <xsl:apply-templates/>
      </schema_instance>
  </xsl:template>

  <xsl:template
match="iso_10303_28_header|data_section_header|documentation|purpose|time_s
tamp|document_name|author|originating_organization|authorization|originatin
g_system|preprocessor_version">
  <xsl:copy>
   <xsl:apply-templates/>
```



```
  </xsl:copy>
</xsl:template>

<xsl:template match="express_schema|external_refid|schema_decl">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>

<xsl:template match="iso_10303_28">
  <iso_10303_28>
    <xsl:attribute name="version">PDTS</xsl:attribute>
    <xsl:attribute name="representation_category">LB</xsl:attribute>
    <xsl:apply-templates/>
  </iso_10303_28>
</xsl:template>

<xsl:template match="express_data">
  <express_data>
    <xsl:attribute name="id"><xsl:value-of select="@id"/></xsl:attribute>
    <xsl:attribute name="representation_category">LB</xsl:attribute>
    <xsl:apply-templates/>
  </express_data>
</xsl:template>

<xsl:template
match="external_refid/@id|external_refid/@xlink:type|external_refid/@xlink:
href|external_refid/@xlink:arcrole|external_refid/@xlink:title|external_ref
id/@xlink:role|external_refid/@xlink:show|external_refid/@xlink:actuate">
  <xsl:copy/>
</xsl:template>

<xsl:template
match="express_schema/@id|express_schema/@express_schema_description|expres
s_schema/@express_schema_identifier|express_schema/@express_schema_version"
>
  <xsl:copy/>
</xsl:template>

<xsl:template name="process_ctn">
 <xsl:param name="lbtype"/>
 <xsl:param name="deftype">none</xsl:param>
 <xsl:param name="stext"/>
 <xsl:variable name="ntext" select="concat(normalize-space($stext),' ')"/>
  <xsl:choose>
   <xsl:when test="$lbtype='entity_instance_ref'">
     <xsl:variable name="pid" select="substring-before($ntext,' ')"/>
     <xsl:choose>
      <xsl:when test="local-name(id($pid))='unset'">
  <unset></unset>
      </xsl:when>
      <xsl:otherwise>
       <entity_instance_ref>
   <xsl:attribute name="refid">
```



```
    <xsl:value-of select="$pid"/>
   </xsl:attribute>
  </entity_instance_ref>
     </xsl:otherwise>
    </xsl:choose>
   </xsl:when>
   <xsl:when test="$lbtype='string_literal' or $lbtype='binary_literal'">
    <xsl:choose>
      <xsl:when test="$deftype='none'">
         <xsl:element name="{$lbtype}">
           <xsl:value-of select="id(substring-before($ntext,' '))"/>
         </xsl:element>
      </xsl:when>
      <xsl:otherwise>
       <xsl:element name="type_literal">
   <xsl:attribute name="express_type_name"><xsl:value-of
select="$deftype"/></xsl:attribute>
         <xsl:element name="{$lbtype}">
          <xsl:value-of select="id(substring-before($ntext,' '))"/>
         </xsl:element>
        </xsl:element>
       </xsl:otherwise>
      </xsl:choose>
   </xsl:when>
   <xsl:when test="$lbtype='select'">
        <xsl:apply-templates select="id(substring-before($ntext,' '))"
mode="xselect"/>
   </xsl:when>
   <xsl:otherwise>
     <xsl:choose>
      <xsl:when test="$deftype='none'">
         <xsl:element name="{$lbtype}">
    <xsl:value-of select="substring-before($ntext,' ')"/>
         </xsl:element>
      </xsl:when>
      <xsl:otherwise>
       <xsl:element name="type_literal">
  <xsl:attribute name="express_type_name"><xsl:value-of
select="$deftype"/></xsl:attribute>
         <xsl:element name="{$lbtype}">
    <xsl:value-of select="substring-before($ntext,' ')"/>
         </xsl:element>
       </xsl:element>
      </xsl:otherwise>
     </xsl:choose>
   </xsl:otherwise>
  </xsl:choose>
  <xsl:if test="substring-after($ntext,' ')">
    <xsl:call-template name="process_ctn">
      <xsl:with-param name="lbtype"><xsl:value-of
select="$lbtype"/></xsl:with-param>

 <xsl:with-param name="deftype"><xsl:value-of
select="$deftype"/></xsl:with-param>
 <xsl:with-param name="stext" select="substring-after($ntext,' ')"/>
```



```
  </xsl:call-template>
   </xsl:if>
</xsl:template>

<xsl:template name="process_nested_ctn">
 <xsl:param name="inner_aggs"/>
 <xsl:param name="lbtype"/>
 <xsl:param name="deftype">none</xsl:param>
 <xsl:param name="stext"/>
 <xsl:variable name="naggs" select="concat(normalize-space($inner_aggs),'
')"/>
 <xsl:variable name="ntext" select="concat(normalize-space($stext),' ')"/>
  <xsl:choose>
     <xsl:when test="substring-after($naggs,' ')">
               <xsl:element name="{substring-before($naggs,' ')}">
         <xsl:call-template name="process_nested_ctn">
         <xsl:with-param name="lbtype"><xsl:value-of
select="$lbtype"/></xsl:with-param>
     <xsl:with-param name="inner_aggs"><xsl:value-of select="substring-
after($naggs,' ')"/></xsl:with-param>
         <xsl:with-param name="stext"><xsl:value-of select="id(substring-
before($ntext,' '))/@c"/></xsl:with-param>
      </xsl:call-template>
               </xsl:element>
   </xsl:when>
   <xsl:otherwise>
             <xsl:element name="{substring-before($naggs,' ')}">
              <xsl:variable name="ctntext" select="substring-
before($ntext,' ')"/>
     <xsl:call-template name="process_ctn">
                <xsl:with-param name="lbtype"><xsl:value-of
select="$lbtype"/></xsl:with-param>
            <xsl:with-param name="deftype"><xsl:value-of
select="$deftype"/></xsl:with-param>
            <xsl:with-param name="stext" select="id($ctntext)/@c"/>
     </xsl:call-template>
              </xsl:element>
     </xsl:otherwise>
      </xsl:choose>
      <xsl:if test="substring-after($ntext,' ')">
       <xsl:call-template name="process_nested_ctn">
         <xsl:with-param name="lbtype"><xsl:value-of
select="$lbtype"/></xsl:with-param>
     <xsl:with-param name="inner_aggs"><xsl:value-of
select="$inner_aggs"/></xsl:with-param>
         <xsl:with-param name="stext"><xsl:value-of select="substring-
after($ntext,' ')"/></xsl:with-param>
      </xsl:call-template>
      </xsl:if>
</xsl:template>


<xsl:template match="*">
</xsl:template>
```



```
<xsl:template match="mjg:Garden">
      <entity_instance  express_entity_name="garden" id="{@x-id}">
     <attribute_instance express_attribute_name="has_pond">
      <entity_instance_ref  refid="{@Has_pond-r}" />
     </attribute_instance>
     <attribute_instance
express_attribute_name="climatic_temperature_range">
      <entity_instance_ref  refid="{@Climatic_temperature_range-r}" />
     </attribute_instance>
     <attribute_instance express_attribute_name="has_greenhouse">
      <entity_instance_ref  refid="{@Has_greenhouse-r}" />
     </attribute_instance>
     <attribute_instance express_attribute_name="has_beds">
            <set_literal>
       <xsl:call-template name="process_ctn">
        <xsl:with-param name="lbtype">entity_instance_ref</xsl:with-param>
        <xsl:with-param name="stext" select="id(@Has_beds-r)/@c"/>
     </xsl:call-template>
            </set_literal>
     </attribute_instance>
     <attribute_instance express_attribute_name="neighbours_garden">
      <entity_instance_ref  refid="{@Neighbours_garden-r}" />
     </attribute_instance>
   </entity_instance>
 </xsl:template>

  <xsl:template match="mjg:Greenhouse">
      <entity_instance  express_entity_name="greenhouse" id="{@x-id}">
     <attribute_instance
express_attribute_name="enforced_temperature_range">
     <entity_instance_ref  refid="{@Enforced_temperature_range-r}" />
     </attribute_instance>
     <attribute_instance express_attribute_name="holds_plants">
           <set_literal>
     <xsl:call-template name="process_ctn">
        <xsl:with-param name="lbtype">entity_instance_ref</xsl:with-param>
        <xsl:with-param name="stext" select="id(@Holds_plants-r)/@c"/>
   </xsl:call-template>
           </set_literal>
     </attribute_instance>
   </entity_instance>
 </xsl:template>

 <xsl:template match="mjg:Amphibian_pond">
  <entity_instance_as_group id="{@x-id}">
       <partial_entity_instance  express_entity_name="pond">
     <attribute_instance express_attribute_name="maintained_by">
      <entity_instance_ref>
   <xsl:attribute name="refid">
   <xsl:value-of select="@Pond.Maintained_by-r"/>
   </xsl:attribute>
      </entity_instance_ref>
     </attribute_instance>
     <attribute_instance express_attribute_name="plants">
           <set_literal>
```



```
    <xsl:call-template name="process_ctn">
       <xsl:with-param name="lbtype">entity_instance_ref</xsl:with-param>
       <xsl:with-param name="stext" select="id(@Plants-r)/@c"/>
  </xsl:call-template>
          </set_literal>
    </attribute_instance>
    <attribute_instance express_attribute_name="water_volume">
  <integer_literal>
       <xsl:value-of select="@Water_volume"/>
  </integer_literal>
    </attribute_instance>
    <attribute_instance express_attribute_name="water_ph">
       <type_literal express_type_name="ph">
        <real_literal>
        <xsl:value-of select="@Water_ph"/>
             </real_literal>
       </type_literal>
     </attribute_instance>
  </partial_entity_instance>
       <partial_entity_instance  express_entity_name="amphibian_pond">
         <attribute_instance express_attribute_name="amphibians">
           <set_literal>
     <xsl:call-template name="process_ctn">
    <xsl:with-param name="lbtype">enumeration_ref</xsl:with-param>
    <xsl:with-param name="deftype">amphibian</xsl:with-param>
    <xsl:with-param name="stext" select="id(@Amphibians-r)/@c"/>
   </xsl:call-template>
          </set_literal>
        </attribute_instance>
  </partial_entity_instance>
 </entity_instance_as_group>
</xsl:template>

<xsl:template match="mjg:Fish_pond">
 <entity_instance_as_group id="{@x-id}">
       <partial_entity_instance  express_entity_name="pond">
    <attribute_instance express_attribute_name="maintained_by">
     <entity_instance_ref>
  <xsl:attribute name="refid">
  <xsl:value-of select="@Pond.Maintained_by-r"/>
  </xsl:attribute>
      </entity_instance_ref>
    </attribute_instance>
    <attribute_instance express_attribute_name="plants">
          <set_literal>
    <xsl:call-template name="process_ctn">
       <xsl:with-param name="lbtype">entity_instance_ref</xsl:with-param>
       <xsl:with-param name="stext" select="id(@Plants-r)/@c"/>
  </xsl:call-template>
          </set_literal>
    </attribute_instance>
    <attribute_instance express_attribute_name="water_volume">
  <integer_literal>
       <xsl:value-of select="@Water_volume"/>
  </integer_literal>
```



```
    </attribute_instance>
    <attribute_instance express_attribute_name="water_ph">
       <type_literal express_type_name="ph">
        <real_literal>
        <xsl:value-of select="@Water_ph"/>
             </real_literal>
       </type_literal>
     </attribute_instance>
  </partial_entity_instance>
      <partial_entity_instance  express_entity_name="fish_pond">
        <attribute_instance express_attribute_name="fish">
          <set_literal>
     <xsl:call-template name="process_ctn">
    <xsl:with-param name="lbtype">enumeration_ref</xsl:with-param>
    <xsl:with-param name="deftype">fish_type</xsl:with-param>
    <xsl:with-param name="stext" select="id(@Fish-r)/@c"/>
   </xsl:call-template>
          </set_literal>
        </attribute_instance>
        <attribute_instance express_attribute_name="fish_colours">
      <set_literal>
     <xsl:call-template name="process_ctn">
    <xsl:with-param name="lbtype">select</xsl:with-param>
    <xsl:with-param name="stext" select="id(@Fish_colours-r)/@c"/>
   </xsl:call-template>
  </set_literal>
        </attribute_instance>
  </partial_entity_instance>
 </entity_instance_as_group>
</xsl:template>

<xsl:template match="mjg:Ornamental_pond">
 <entity_instance_as_group id="{@x-id}">
       <partial_entity_instance  express_entity_name="pond">
     <attribute_instance express_attribute_name="maintained_by">
      <entity_instance_ref>
  <xsl:attribute name="refid">
  <xsl:value-of select="@Maintained_by-r"/>
  </xsl:attribute>
      </entity_instance_ref>
    </attribute_instance>
    <attribute_instance express_attribute_name="plants">
          <set_literal>
    <xsl:call-template name="process_ctn">
       <xsl:with-param name="lbtype">entity_instance_ref</xsl:with-param>
       <xsl:with-param name="stext" select="id(@Plants-r)/@c"/>
  </xsl:call-template>
          </set_literal>
    </attribute_instance>
    <attribute_instance express_attribute_name="water_volume">
  <integer_literal>
       <xsl:value-of select="@Water_volume"/>
  </integer_literal>
    </attribute_instance>
    <attribute_instance express_attribute_name="water_ph">
```



```
       <type_literal express_type_name="ph">
        <real_literal>
        <xsl:value-of select="@Water_ph"/>
             </real_literal>
       </type_literal>
      </attribute_instance>
  </partial_entity_instance>
       <partial_entity_instance  express_entity_name="ornamental_pond">
         <attribute_instance express_attribute_name="ornaments">
           <set_literal>
     <xsl:call-template name="process_ctn">
    <xsl:with-param name="lbtype">enumeration_ref</xsl:with-param>
    <xsl:with-param name="deftype">pond_ornament</xsl:with-param>
    <xsl:with-param name="stext" select="id(@Ornaments-r)/@c"/>
   </xsl:call-template>
           </set_literal>
         </attribute_instance>
  </partial_entity_instance>
 </entity_instance_as_group>
</xsl:template>

<xsl:template match="mjg:Observed_fish_colour" mode="xselect">
  <type_literal express_type_name="observed_fish_colour">
 <xsl:choose>
   <xsl:when test="contains(@utype,'Fish_colour')">
           <type_literal express_type_name="fish_colour">
            <enumeration_ref>
        <xsl:value-of select="id(@val-r)/@val"/>
     </enumeration_ref>
      </type_literal>
   </xsl:when>
   <xsl:when test="contains(@utype,'Multi_coloured_fish_colour')">
           <type_literal express_type_name="multi_coloured_fish_colour">
            <set_literal>
     <xsl:call-template name="process_ctn">
    <xsl:with-param name="lbtype">enumeration_ref</xsl:with-param>
    <xsl:with-param name="deftype">fish_colour</xsl:with-param>
    <xsl:with-param name="stext" select="id(id(@val-r)/@val-r)/@c"/>
   </xsl:call-template>
       </set_literal>
           </type_literal>
   </xsl:when>
   <xsl:otherwise>
   </xsl:otherwise>
 </xsl:choose>
  </type_literal>
</xsl:template>

 <xsl:template match="mjg:Bed">
      <entity_instance  express_entity_name="bed" id="{@x-id}">
    <attribute_instance express_attribute_name="acidity">
     <type_literal  express_type_name="ph"
express_schema_name="support_items">
    <real_literal><xsl:value-of select="@Acidity" /></real_literal>
     </type_literal>
```



```
    </attribute_instance>
    <attribute_instance express_attribute_name="holds_plants">
          <set_literal>
    <xsl:call-template name="process_ctn">
       <xsl:with-param name="lbtype">entity_instance_ref</xsl:with-param>
       <xsl:with-param name="stext" select="id(@Holds_plants-r)/@c"/>
  </xsl:call-template>
          </set_literal>
    </attribute_instance>
    <xsl:if test="@Planting_plan-r">
     <attribute_instance express_attribute_name="planting_plan">
      <array_literal>
    <xsl:call-template name="process_nested_ctn">
        <xsl:with-param name="lbtype">entity_instance_ref</xsl:with-param>
    <xsl:with-param name="inner_aggs">array_literal</xsl:with-param>
        <xsl:with-param name="stext" select="id(@Planting_plan-r)/@c"/>
  </xsl:call-template>
     </array_literal>
     </attribute_instance>
    </xsl:if>
    <attribute_instance express_attribute_name="flowering_order">
          <list_literal>
     <xsl:call-template name="process_ctn">
      <xsl:with-param name="lbtype">entity_instance_ref</xsl:with-param>
      <xsl:with-param name="stext" select="id(@Flowering_order-r)/@c"/>
  </xsl:call-template>
          </list_literal>
    </attribute_instance>
  </entity_instance>
</xsl:template>

 <xsl:template match="mjg:Outdoors_plant">
<entity_instance_as_group id="{@x-id}">
      <partial_entity_instance  express_entity_name="plant">
    <attribute_instance express_attribute_name="colour">
  <type_literal express_type_name="flower_colour">
      <enumeration_ref><xsl:value-of select="@Colour"/>
  </enumeration_ref>
  </type_literal>
    </attribute_instance>
    <attribute_instance express_attribute_name="latin_name">
     <type_literal  express_type_name="plant_name">
      <string_literal>
        <xsl:value-of select="id(@Latin_name-r)"/>
      </string_literal>
  </type_literal>
    </attribute_instance>
    <attribute_instance express_attribute_name="english_names">
           <set_literal>
    <xsl:call-template name="process_ctn">
   <xsl:with-param name="lbtype">string_literal</xsl:with-param>
   <xsl:with-param name="deftype">plant_name</xsl:with-param>
   <xsl:with-param name="stext" select="id(@English_names-r)/@c"/>
  </xsl:call-template>
          </set_literal>
```



```
    </attribute_instance>
        <attribute_instance
express_attribute_name="survival_temperature_range"><entity_instance_ref
     refid="{@Survival_temperature_range-r}"/>
        </attribute_instance>
  </partial_entity_instance>
   <partial_entity_instance express_entity_name="outdoors_plant">
       <attribute_instance express_attribute_name="survival_ph_range">
      <entity_instance_ref refid="{@Survival_ph_range-r}"/>
       </attribute_instance>
   </partial_entity_instance>
    </entity_instance_as_group>
 </xsl:template>

  <xsl:template match="mjg:Greenhouse_plant">
 <entity_instance_as_group id="{@x-id}">
       <partial_entity_instance  express_entity_name="plant">
    <attribute_instance express_attribute_name="colour">
   <type_literal express_type_name="flower_colour">
       <enumeration_ref><xsl:value-of select="@Colour"/>
   </enumeration_ref>
   </type_literal>
    </attribute_instance>
    <attribute_instance express_attribute_name="latin_name">
     <type_literal  express_type_name="plant_name">
      <string_literal>
        <xsl:value-of select="id(@Latin_name-r)"/>
      </string_literal>
   </type_literal>
    </attribute_instance>
    <attribute_instance express_attribute_name="english_names">
            <set_literal>
    <xsl:call-template name="process_ctn">
    <xsl:with-param name="lbtype">string_literal</xsl:with-param>
    <xsl:with-param name="deftype">plant_name</xsl:with-param>
    <xsl:with-param name="stext" select="id(@English_names-r)/@c"/>
   </xsl:call-template>
            </set_literal>
    </attribute_instance>
        <attribute_instance
express_attribute_name="survival_temperature_range"><entity_instance_ref
     refid="{@Survival_temperature_range-r}"/>
        </attribute_instance>
      </partial_entity_instance>
   <partial_entity_instance express_entity_name="greenhouse_plant">
   </partial_entity_instance>
    </entity_instance_as_group>
 </xsl:template>

  <xsl:template match="mjg:Aquatic_plant">
 <entity_instance_as_group id="{@x-id}">
       <partial_entity_instance  express_entity_name="plant">
    <attribute_instance express_attribute_name="colour">
   <type_literal express_type_name="flower_colour">
       <enumeration_ref><xsl:value-of select="@Colour"/>
```



```
   </enumeration_ref>
   </type_literal>
     </attribute_instance>
     <attribute_instance express_attribute_name="latin_name">
      <type_literal  express_type_name="plant_name">
       <string_literal>
         <xsl:value-of select="id(@Latin_name-r)"/>
       </string_literal>
   </type_literal>
     </attribute_instance>
     <attribute_instance express_attribute_name="english_names">
            <set_literal>
     <xsl:call-template name="process_ctn">
    <xsl:with-param name="lbtype">string_literal</xsl:with-param>
    <xsl:with-param name="deftype">plant_name</xsl:with-param>
    <xsl:with-param name="stext" select="id(@English_names-r)/@c"/>
   </xsl:call-template>
           </set_literal>
     </attribute_instance>
         <attribute_instance
express_attribute_name="survival_temperature_range"><entity_instance_ref
     refid="{@Survival_temperature_range-r}"/>
         </attribute_instance>
  </partial_entity_instance>
  <partial_entity_instance express_entity_name="aquatic_plant">
         <attribute_instance express_attribute_name="aquatic_plant_type">
           <type_literal express_type_name="aquatic_plant_types">
             <enumeration_ref><xsl:value-of
select="@Aquatic_plant_type"/></enumeration_ref>
           </type_literal>
         </attribute_instance>
     <attribute_instance
express_attribute_name="oxygen_volumetric_requirement">
        <integer_literal><xsl:value-of
select="@Oxygen_volumetric_requirement"/>
    </integer_literal>
     </attribute_instance>
     <attribute_instance express_attribute_name="aquatic_plant_size">
       <integer_literal><xsl:value-of select="@Aquatic_plant_size"/>
   </integer_literal>
     </attribute_instance>
  </partial_entity_instance>
    </entity_instance_as_group>
 </xsl:template>

 <xsl:template match="mjg:Filtration_system">
     <entity_instance_as_group id="{@x-id}">
       <partial_entity_instance  express_entity_name="filtration_system">
  <attribute_instance express_attribute_name="filtration_efficiency">
           <xsl:apply-templates select="id(@Filtration_efficiency-r)"
mode="xselect"/>
  </attribute_instance>
  <attribute_instance express_attribute_name="number_of_filters">
            <integer_literal>
   <xsl:value-of select="@Number_of_filters"/>
```



```
            </integer_literal>
   </attribute_instance>
        </partial_entity_instance>
        <partial_entity_instance
express_entity_name="water_treatment_system">
     <attribute_instance express_attribute_name="capacity">
       <integer_literal>
              <xsl:value-of select="@Capacity"/>
       </integer_literal>
     </attribute_instance>
   </partial_entity_instance>
  </entity_instance_as_group>
 </xsl:template>

<xsl:template match="si:Efficiency" mode="xselect">
  <type_literal express_type_name="efficiency">
 <xsl:choose>
   <xsl:when test="contains(@utype,'Efficiency_measure')">
            <xsl:apply-templates select="id(@val-r)" mode="xselect"/>
   </xsl:when>
   <xsl:when test="contains(@utype,'Efficiency_range')">
     <entity_instance_ref>
   <xsl:attribute name="refid">
   <xsl:value-of select="@val-r"/>
   </xsl:attribute>
     </entity_instance_ref>
   </xsl:when>
   <xsl:otherwise>
   </xsl:otherwise>
 </xsl:choose>
  </type_literal>
</xsl:template>

<xsl:template match="si:Efficiency_measure" mode="xselect">
  <type_literal express_type_name="efficiency_measure">
 <xsl:choose>
   <xsl:when test="contains(@utype,'Fractional_value')">
            <type_literal express_type_name="fractional_value">
             <real_literal>
       <xsl:value-of select="id(@val-r)/@val"/>
      </real_literal>
            </type_literal>
   </xsl:when>
   <xsl:when test="contains(@utype,'Percentage')">
            <type_literal express_type_name="percentage">
             <real_literal>
       <xsl:value-of select="id(@val-r)/@val"/>
      </real_literal>
            </type_literal>
   </xsl:when>
   <xsl:otherwise>
   </xsl:otherwise>
 </xsl:choose>
  </type_literal>
</xsl:template>
```



```
 <xsl:template match="si:Temperature_range">
      <entity_instance  express_entity_name="real_value_range"
   express_schema_name="{$refschema1}"  id="{@x-id}">
       <attribute_instance express_attribute_name="minimum_value">
      <real_literal>
       <xsl:value-of select="@Minimum_value"/>
      </real_literal>
       </attribute_instance>
       <attribute_instance express_attribute_name="maximum_value">
      <real_literal>
       <xsl:value-of select="@Maximum_value"/>
      </real_literal>
       </attribute_instance>
   <partial_entity_instance express_entity_name="temperature_range"
    express_schema_name="support_items">
   </partial_entity_instance>
 </entity_instance>
</xsl:template>

<xsl:template match="si:Ph_range">
      <entity_instance  express_entity_name="real_value_range"
   express_schema_name="{$refschema1}"  id="{@x-id}">
       <attribute_instance express_attribute_name="minimum_value">
      <real_literal>
       <xsl:value-of select="@Minimum_value"/>
      </real_literal>
       </attribute_instance>
       <attribute_instance express_attribute_name="maximum_value">
      <real_literal>
       <xsl:value-of select="@Maximum_value"/>
      </real_literal>
       </attribute_instance>
   <partial_entity_instance express_entity_name="ph_range"
    express_schema_name="support_items">
   </partial_entity_instance>
 </entity_instance>
</xsl:template>

<xsl:template match="si:Efficiency_range">
      <entity_instance  express_entity_name="efficiency_range"
     id="{@x-id}">
    <partial_entity_instance express_entity_name="real_value_range">
       <attribute_instance express_attribute_name="minimum_value">
      <real_literal>
       <xsl:value-of select="@Minimum_value"/>
      </real_literal>
       </attribute_instance>
       <attribute_instance express_attribute_name="maximum_value">
      <real_literal>
       <xsl:value-of select="@Maximum_value"/>
      </real_literal>
       </attribute_instance>
    </partial_entity_instance>
 </entity_instance>
```



```
</xsl:template>

<xsl:template match="msg:Garden">
      <entity_instance  express_entity_name="garden"
express_schema_name="{$refschema2}" id="{@x-id}">
     <attribute_instance express_attribute_name="has_bed">
      <entity_instance_ref>
   <xsl:attribute name="refid">
   <xsl:value-of select="@Has_bed-r"/>
   </xsl:attribute>
       </entity_instance_ref>
     </attribute_instance>
   </entity_instance>
</xsl:template>

<xsl:template match="msg:Bed">
      <entity_instance  express_entity_name="bed"
express_schema_name="{$refschema2}" id="{@x-id}">
     <attribute_instance express_attribute_name="description">
       <string_literal><xsl:value-of select="id(@Description-
r)"/></string_literal>
     </attribute_instance>
   </entity_instance>
</xsl:template>

<xsl:template match="mjg:Mr_smiths_plant">
      <entity_instance  express_entity_name="plant"
express_schema_name="{$refschema2}" id="{@x-id}">
     <attribute_instance express_attribute_name="name">
       <string_literal><xsl:value-of select="id(@Name-
r)"/></string_literal>
     </attribute_instance>
   </entity_instance>
</xsl:template>

</xsl:stylesheet>
```

H.5.4 Late bound representation of OSEB data

The following is the result of applying the XSLT stylesheet in H.5.3 to the OSEB data in H.5.2.

```
<iso_10303_28 xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:msg="urn:10303-28:oseb/Mr_smiths_garden" xmlns:si="urn:10303-
28:oseb/Support_items" xmlns:mjg="urn:10303-28:oseb/Mr_jones_garden"
xmlns:osb="urn:10303-28:oseb" version="PDTS" representation_category="LB">
  <iso_10303_28_header>
     <document_name> OSEB MJG Example </document_name>
     <purpose>Example OSEB representation of Mr Jones Garden EXPRESS
schema</purpose>
           <time_stamp>2000-09-11 1200 UCT</time_stamp>
           <author>J.Trausch</author>
```



```
    <originating_organization>Electric Boat</originating_organization>
    <authorization>Approved by D405</authorization>
    <originating_system>ByHand</originating_system>
    <preprocessor_version>1.0</preprocessor_version>
 </iso_10303_28_header>
     <express_data id="mjg" representation_category="LB">
 <schema_instance express_schema_name="mr_jones_garden">
  <entity_instance id="g1" express_entity_name="garden">
                <attribute_instance express_attribute_name="has_pond">
                    <entity_instance_ref refid="po1"/>
                </attribute_instance>
                <attribute_instance
express_attribute_name="climatic_temperature_range">
                    <entity_instance_ref refid="tr1"/>
                </attribute_instance>
                <attribute_instance
express_attribute_name="has_greenhouse">
                    <entity_instance_ref refid="gh1"/>
                </attribute_instance>
                <attribute_instance express_attribute_name="has_beds">
                    <set_literal>
                        <entity_instance_ref refid="bed1"/>
                        <entity_instance_ref refid="bed2"/>
                        <entity_instance_ref refid="bed4"/>
                        <entity_instance_ref refid="bed3"/>
                        <entity_instance_ref refid="bed5"/>
                    </set_literal>
                </attribute_instance>
                <attribute_instance
express_attribute_name="neighbours_garden">
                    <entity_instance_ref refid="sg1"/>
                </attribute_instance>
            </entity_instance>

   <entity_instance_as_group id="po1">
                <partial_entity_instance express_entity_name="pond">
                    <attribute_instance
express_attribute_name="maintained_by">
                        <entity_instance_ref refid="fs1"/>
                    </attribute_instance>
                    <attribute_instance express_attribute_name="plants">
                        <set_literal>
                            <entity_instance_ref refid="ap1"/>
                            <entity_instance_ref refid="ap2"/>
                        </set_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="water_volume">
                        <integer_literal>4000</integer_literal>
                    </attribute_instance>
                    <attribute_instance express_attribute_name="water_ph">
                        <type_literal express_type_name="ph">
                            <real_literal>6.8</real_literal>
                        </type_literal>
                    </attribute_instance>
```



```
                </partial_entity_instance>
                <partial_entity_instance express_entity_name="fish_pond">
                    <attribute_instance express_attribute_name="fish">
                        <set_literal>
                            <type_literal express_type_name="fish_type">
                                <enumeration_ref>koi</enumeration_ref>
                            </type_literal>
                            <type_literal express_type_name="fish_type">
                                <enumeration_ref>goldfish</enumeration_ref>
                            </type_literal>
                        </set_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="fish_colours">
                        <set_literal>
                            <type_literal
express_type_name="observed_fish_colour">
                                <type_literal
express_type_name="fish_colour">

<enumeration_ref>yellow</enumeration_ref>
                                </type_literal>
                            </type_literal>
                            <type_literal
express_type_name="observed_fish_colour">
                                <type_literal
express_type_name="fish_colour">

<enumeration_ref>black</enumeration_ref>
                                </type_literal>
                            </type_literal>
                            <type_literal
express_type_name="observed_fish_colour">
                                <type_literal
express_type_name="multi_coloured_fish_colour">
                                    <set_literal>
                                        <type_literal
express_type_name="fish_colour">

<enumeration_ref>silver</enumeration_ref>
                                        </type_literal>
                                        <type_literal
express_type_name="fish_colour">

<enumeration_ref>blue</enumeration_ref>
                                        </type_literal>
                                        <type_literal
express_type_name="fish_colour">

<enumeration_ref>yellow</enumeration_ref>
                                        </type_literal>
                                    </set_literal>
                                </type_literal>
                            </type_literal>
                        </set_literal>
```



```
                    </attribute_instance>
                </partial_entity_instance>
            </entity_instance_as_group>

   <entity_instance id="gh1" express_entity_name="greenhouse">
                <attribute_instance
express_attribute_name="enforced_temperature_range">
                    <entity_instance_ref refid="tr2"/>
                </attribute_instance>
                <attribute_instance express_attribute_name="holds_plants">
                    <set_literal>
                        <entity_instance_ref refid="gp1"/>
                        <entity_instance_ref refid="gp2"/>
                    </set_literal>
                </attribute_instance>
            </entity_instance>









   <entity_instance id="bed1" express_entity_name="bed">
                <attribute_instance express_attribute_name="acidity">
                    <type_literal express_schema_name="support_items"
express_type_name="ph">
                        <real_literal>6.1</real_literal>
                    </type_literal>
                </attribute_instance>
                <attribute_instance express_attribute_name="holds_plants">
                    <set_literal>
                        <entity_instance_ref refid="op1"/>
                        <entity_instance_ref refid="op2"/>
                    </set_literal>
                </attribute_instance>
                <attribute_instance express_attribute_name="planting_plan">
                    <array_literal>
                        <array_literal>
                            <entity_instance_ref refid="op1"/>
                            <entity_instance_ref refid="op1"/>
```



```
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
    </array_literal>
    <array_literal>
        <entity_instance_ref refid="op2"/>
        <entity_instance_ref refid="op2"/>
        <entity_instance_ref refid="op2"/>
        <entity_instance_ref refid="op2"/>
        <entity_instance_ref refid="op2"/>
        <entity_instance_ref refid="op2"/>
        <entity_instance_ref refid="op2"/>
        <entity_instance_ref refid="op2"/>
        <entity_instance_ref refid="op2"/>
        <entity_instance_ref refid="op2"/>
    </array_literal>
    <array_literal>
        <unset/>
        <unset/>
        <unset/>
        <unset/>
        <unset/>
        <unset/>
        <unset/>
        <unset/>
        <unset/>
        <unset/>
    </array_literal>
    <array_literal>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
    </array_literal>
    <array_literal>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
        <entity_instance_ref refid="op1"/>
```



```
                        <entity_instance_ref refid="op1"/>
                    </array_literal>
                </array_literal>
            </attribute_instance>
            <attribute_instance
express_attribute_name="flowering_order">
                <list_literal>
                    <entity_instance_ref refid="op2"/>
                    <entity_instance_ref refid="op1"/>
                </list_literal>
            </attribute_instance>
        </entity_instance>
   <entity_instance id="bed2" express_entity_name="bed">
            <attribute_instance express_attribute_name="acidity">
                <type_literal express_schema_name="support_items"
express_type_name="ph">
                    <real_literal>6.2</real_literal>
                </type_literal>
            </attribute_instance>
            <attribute_instance express_attribute_name="holds_plants">
                <set_literal>
                    <entity_instance_ref refid="op1"/>
                    <entity_instance_ref refid="op2"/>
                </set_literal>
            </attribute_instance>
            <attribute_instance
express_attribute_name="flowering_order">
                <list_literal>
                    <entity_instance_ref refid="op2"/>
                    <entity_instance_ref refid="op1"/>
                </list_literal>
            </attribute_instance>
        </entity_instance>
   <entity_instance id="bed3" express_entity_name="bed">
            <attribute_instance express_attribute_name="acidity">
                <type_literal express_schema_name="support_items"
express_type_name="ph">
                    <real_literal>6.3</real_literal>
                </type_literal>
            </attribute_instance>
            <attribute_instance express_attribute_name="holds_plants">
                <set_literal>
                    <entity_instance_ref refid="op1"/>
                    <entity_instance_ref refid="op2"/>
                </set_literal>
            </attribute_instance>
            <attribute_instance
express_attribute_name="flowering_order">
                <list_literal>
                    <entity_instance_ref refid="op2"/>
                    <entity_instance_ref refid="op1"/>
                </list_literal>
            </attribute_instance>
        </entity_instance>
   <entity_instance id="bed4" express_entity_name="bed">
```



```
                <attribute_instance express_attribute_name="acidity">
                    <type_literal express_schema_name="support_items"
express_type_name="ph">
                        <real_literal>6.4</real_literal>
                    </type_literal>
                </attribute_instance>
                <attribute_instance express_attribute_name="holds_plants">
                    <set_literal>
                        <entity_instance_ref refid="op1"/>
                        <entity_instance_ref refid="op2"/>
                    </set_literal>
                </attribute_instance>
                <attribute_instance
express_attribute_name="flowering_order">
                    <list_literal>
                        <entity_instance_ref refid="op2"/>
                        <entity_instance_ref refid="op1"/>
                    </list_literal>
                </attribute_instance>
            </entity_instance>
   <entity_instance id="bed5" express_entity_name="bed">
                <attribute_instance express_attribute_name="acidity">
                    <type_literal express_schema_name="support_items"
express_type_name="ph">
                        <real_literal>6.5</real_literal>
                    </type_literal>
                </attribute_instance>
                <attribute_instance express_attribute_name="holds_plants">
                    <set_literal>
                        <entity_instance_ref refid="op1"/>
                        <entity_instance_ref refid="op2"/>
                    </set_literal>
                </attribute_instance>
                <attribute_instance
express_attribute_name="flowering_order">
                    <list_literal>
                        <entity_instance_ref refid="op2"/>
                        <entity_instance_ref refid="op1"/>
                    </list_literal>
                </attribute_instance>
            </entity_instance>

   <entity_instance_as_group id="op1">
                <partial_entity_instance express_entity_name="plant">
                    <attribute_instance express_attribute_name="colour">
                        <type_literal express_type_name="flower_colour">
                            <enumeration_ref>red</enumeration_ref>
                        </type_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="latin_name">
                        <type_literal express_type_name="plant_name">
                            <string_literal>rosa</string_literal>
                        </type_literal>
                    </attribute_instance>
```



```
                    <attribute_instance
express_attribute_name="english_names">
                        <set_literal>
                            <type_literal express_type_name="plant_name">
                                <string_literal>rose</string_literal>
                            </type_literal>
                        </set_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="survival_temperature_range">
                        <entity_instance_ref refid="tr3"/>
                    </attribute_instance>
                </partial_entity_instance>
                <partial_entity_instance
express_entity_name="outdoors_plant">
                    <attribute_instance
express_attribute_name="survival_ph_range">
                        <entity_instance_ref refid="phr1"/>
                    </attribute_instance>
                </partial_entity_instance>
            </entity_instance_as_group>
   <entity_instance_as_group id="op2">
                <partial_entity_instance express_entity_name="plant">
                    <attribute_instance express_attribute_name="colour">
                        <type_literal express_type_name="flower_colour">
                            <enumeration_ref>yellow</enumeration_ref>
                        </type_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="latin_name">
                        <type_literal express_type_name="plant_name">
                            <string_literal>rosa</string_literal>
                        </type_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="english_names">
                        <set_literal>
                            <type_literal express_type_name="plant_name">
                                <string_literal>rose</string_literal>
                            </type_literal>
                        </set_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="survival_temperature_range">
                        <entity_instance_ref refid="tr3"/>
                    </attribute_instance>
                </partial_entity_instance>
                <partial_entity_instance
express_entity_name="outdoors_plant">
                    <attribute_instance
express_attribute_name="survival_ph_range">
                        <entity_instance_ref refid="phr1"/>
                    </attribute_instance>
                </partial_entity_instance>
            </entity_instance_as_group>
```



```
   <entity_instance_as_group id="ap1">
                <partial_entity_instance express_entity_name="plant">
                    <attribute_instance express_attribute_name="colour">
                        <type_literal express_type_name="flower_colour">
                            <enumeration_ref>white</enumeration_ref>
                        </type_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="latin_name">
                        <type_literal express_type_name="plant_name">
                            <string_literal>Zephyranthis</string_literal>
                        </type_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="english_names">
                        <set_literal>
                            <type_literal express_type_name="plant_name">
                                <string_literal>water
lilly</string_literal>
                            </type_literal>
                            <type_literal express_type_name="plant_name">
                                <string_literal>Fairy
lilly</string_literal>
                            </type_literal>
                        </set_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="survival_temperature_range">
                        <entity_instance_ref refid="tr4"/>
                    </attribute_instance>
                </partial_entity_instance>
                <partial_entity_instance
express_entity_name="aquatic_plant">
                    <attribute_instance
express_attribute_name="aquatic_plant_type">
                        <type_literal
express_type_name="aquatic_plant_types">
                            <enumeration_ref>lilly</enumeration_ref>
                        </type_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="oxygen_volumetric_requirement">
                        <integer_literal>72</integer_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="aquatic_plant_size">
                        <integer_literal>20</integer_literal>
                    </attribute_instance>
                </partial_entity_instance>
            </entity_instance_as_group>
   <entity_instance_as_group id="ap2">
                <partial_entity_instance express_entity_name="plant">
                    <attribute_instance express_attribute_name="colour">
                        <type_literal express_type_name="flower_colour">
```



```
                            <enumeration_ref>yellow</enumeration_ref>
                        </type_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="latin_name">
                        <type_literal express_type_name="plant_name">
                            <string_literal>Zephyranthis</string_literal>
                        </type_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="english_names">
                        <set_literal>
                            <type_literal express_type_name="plant_name">
                                <string_literal>water
lilly</string_literal>
                            </type_literal>
                            <type_literal express_type_name="plant_name">
                                <string_literal>Calla
lillies</string_literal>
                            </type_literal>
                        </set_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="survival_temperature_range">
                        <entity_instance_ref refid="tr4"/>
                    </attribute_instance>
                </partial_entity_instance>
                <partial_entity_instance
express_entity_name="aquatic_plant">
                    <attribute_instance
express_attribute_name="aquatic_plant_type">
                        <type_literal
express_type_name="aquatic_plant_types">
                            <enumeration_ref>lilly</enumeration_ref>
                        </type_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="oxygen_volumetric_requirement">
                        <integer_literal>72</integer_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="aquatic_plant_size">
                        <integer_literal>20</integer_literal>
                    </attribute_instance>
                </partial_entity_instance>
            </entity_instance_as_group>

   <entity_instance_as_group id="gp1">
                <partial_entity_instance express_entity_name="plant">
                    <attribute_instance express_attribute_name="colour">
                        <type_literal express_type_name="flower_colour">
                            <enumeration_ref>yellow</enumeration_ref>
                        </type_literal>
                    </attribute_instance>
```



```
                    <attribute_instance
express_attribute_name="latin_name">
                        <type_literal express_type_name="plant_name">
                            <string_literal>Viola pedata</string_literal>
                        </type_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="english_names">
                        <set_literal>
                            <type_literal express_type_name="plant_name">
                                <string_literal>Violets</string_literal>
                            </type_literal>
                        </set_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="survival_temperature_range">
                        <entity_instance_ref refid="tr5"/>
                    </attribute_instance>
                </partial_entity_instance>
                <partial_entity_instance
express_entity_name="greenhouse_plant"/>
            </entity_instance_as_group>
   <entity_instance_as_group id="gp2">
                <partial_entity_instance express_entity_name="plant">
                    <attribute_instance express_attribute_name="colour">
                        <type_literal express_type_name="flower_colour">
                            <enumeration_ref>red</enumeration_ref>
                        </type_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="latin_name">
                        <type_literal express_type_name="plant_name">
                            <string_literal>Viola sp</string_literal>
                        </type_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="english_names">
                        <set_literal>
                            <type_literal express_type_name="plant_name">
                                <string_literal>Pansies</string_literal>
                            </type_literal>
                        </set_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="survival_temperature_range">
                        <entity_instance_ref refid="tr5"/>
                    </attribute_instance>
                </partial_entity_instance>
                <partial_entity_instance
express_entity_name="greenhouse_plant"/>
            </entity_instance_as_group>

   <entity_instance id="tr1" express_schema_name="support_items"
express_entity_name="real_value_range">
                <attribute_instance express_attribute_name="minimum_value">
```



```
                    <real_literal>60</real_literal>
                </attribute_instance>
                <attribute_instance express_attribute_name="maximum_value">
                    <real_literal>90</real_literal>
                </attribute_instance>
                <partial_entity_instance
express_schema_name="support_items"
express_entity_name="temperature_range"/>
            </entity_instance>
   <entity_instance id="tr2" express_schema_name="support_items"
express_entity_name="real_value_range">
                <attribute_instance express_attribute_name="minimum_value">
                    <real_literal>70</real_literal>
                </attribute_instance>
                <attribute_instance express_attribute_name="maximum_value">
                    <real_literal>92</real_literal>
                </attribute_instance>
                <partial_entity_instance
express_schema_name="support_items"
express_entity_name="temperature_range"/>
            </entity_instance>
   <entity_instance id="tr3" express_schema_name="support_items"
express_entity_name="real_value_range">
                <attribute_instance express_attribute_name="minimum_value">
                    <real_literal>65</real_literal>
                </attribute_instance>
                <attribute_instance express_attribute_name="maximum_value">
                    <real_literal>85</real_literal>
                </attribute_instance>
                <partial_entity_instance
express_schema_name="support_items"
express_entity_name="temperature_range"/>
            </entity_instance>
   <entity_instance id="tr4" express_schema_name="support_items"
express_entity_name="real_value_range">
                <attribute_instance express_attribute_name="minimum_value">
                    <real_literal>65</real_literal>
                </attribute_instance>
                <attribute_instance express_attribute_name="maximum_value">
                    <real_literal>90</real_literal>
                </attribute_instance>
                <partial_entity_instance
express_schema_name="support_items"
express_entity_name="temperature_range"/>
            </entity_instance>
   <entity_instance id="tr5" express_schema_name="support_items"
express_entity_name="real_value_range">
                <attribute_instance express_attribute_name="minimum_value">
                    <real_literal>75</real_literal>
                </attribute_instance>
                <attribute_instance express_attribute_name="maximum_value">
                    <real_literal>90</real_literal>
                </attribute_instance>
```



```
                <partial_entity_instance
express_schema_name="support_items"
express_entity_name="temperature_range"/>
            </entity_instance>
   <entity_instance id="efr1" express_entity_name="efficiency_range">
                <partial_entity_instance
express_entity_name="real_value_range">
                    <attribute_instance
express_attribute_name="minimum_value">
                        <real_literal>75</real_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="maximum_value">
                        <real_literal>85</real_literal>
                    </attribute_instance>
                </partial_entity_instance>
            </entity_instance>

   <entity_instance id="phr1" express_schema_name="support_items"
express_entity_name="real_value_range">
                <attribute_instance express_attribute_name="minimum_value">
                    <real_literal>6.0</real_literal>
                </attribute_instance>
                <attribute_instance express_attribute_name="maximum_value">
                    <real_literal>8.0</real_literal>
                </attribute_instance>
                <partial_entity_instance
express_schema_name="support_items" express_entity_name="ph_range"/>
            </entity_instance>

   <entity_instance_as_group id="fs1">
                <partial_entity_instance
express_entity_name="filtration_system">
                    <attribute_instance
express_attribute_name="filtration_efficiency">
                        <type_literal express_type_name="efficiency">
                            <entity_instance_ref refid="efr1"/>
                        </type_literal>
                    </attribute_instance>
                    <attribute_instance
express_attribute_name="number_of_filters">
                        <integer_literal>4</integer_literal>
                    </attribute_instance>
                </partial_entity_instance>
                <partial_entity_instance
express_entity_name="water_treatment_system">
                    <attribute_instance express_attribute_name="capacity">
                        <integer_literal>5000</integer_literal>
                    </attribute_instance>
                </partial_entity_instance>
            </entity_instance_as_group>
```



```
   <entity_instance id="sg1" express_schema_name="mr_smiths_garden"
express_entity_name="garden">
                <attribute_instance express_attribute_name="has_bed">
                    <entity_instance_ref refid="msbed1"/>
                </attribute_instance>
            </entity_instance>

   <entity_instance id="msbed1" express_schema_name="mr_smiths_garden"
express_entity_name="bed">
                <attribute_instance express_attribute_name="description">
                    <string_literal>Mr. Smith's flower bed</string_literal>
                </attribute_instance>
            </entity_instance>



   <entity_instance id="splant1" express_schema_name="mr_smiths_garden"
express_entity_name="plant">
                <attribute_instance express_attribute_name="name">
                    <string_literal>rose</string_literal>
                </attribute_instance>
            </entity_instance>

   </schema_instance>
 </express_data>
</iso_10303_28>
```


Annex I
(informative)

Mapping of Object Serialization Early Binding to Late Binding using XSLT

I.1 Introduction

This annex describes an algorithm that generates an XSLT stylesheet to transform an Object Serialization Early Binding (OSEB) XML document to its representation as a Late Bound XML document. The algorithm requires as input the EXPRESS schemas from which the Object Serialization Early Binding document was derived and knowledge of the Object Serialization Early Binding naming rule for EXPRESS entity data types and attributes specified in 9.2. The generated XSLT stylesheet is schema dependent and will transform Object Serialization Early Binding document instances that were derived from the EXPRESS schemas that were used to generate the script.

The generation of the XSLT stylesheet is described in this clause as a two step process:

— first, the XSLT stylesheet is initialized with the XSLT markup that is schema independent. This includes markup to process or ignore Object Serialization Early Binding-specific elements and markup necessary to map the exchange file header to the Late Binding;

— second, the XSLT markup that is schema dependent is added to the stylesheet. This is accomplished by recursively processing the EXPRESS entity datatypes and their corresponding attributes in the EXPRESS schema and outputting the resulting XSLT to the style sheet based on the rules specified in this Annex.

I.2 XSLT stylesheet initialization

The XSLT stylesheet is initialized with the following XSLT markup:

```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
   <xsl:apply-templates/>
  </xsl:template>
```

If the EXPRESS schemas being processed result in the use of namespaces in the Object Serialization Early Binding, the same namespace defintions are added to the `xsl:stylesheet` element.

EXAMPLE -The following example illustrates the addition of XML namespaces.

In OSEB XML:

```
<osb:uos c="g1 splant1" unset="unset" schema="mr_jones_garden"

 xmlns:osb="urn:10303-28:oseb"
 xmlns:mjg="urn:10303-28:oseb/Mr_jones_garden/"
 xmlns:si="urn:10303-28:oseb/Support_items/"
              xmlns:msg="urn:10303-28:oseb/Mr_smiths_garden/">
```

In XSLT stylesheet:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform"
 xmlns:osb="urn:10303-28:oseb"
 xmlns:mjg="urn:10303-28:oseb/Mr_jones_garden/"
 xmlns:si="urn:10303-28:oseb/Support_items/"
              xmlns:msg="urn:10303-28:oseb/Mr_smiths_garden/">
```

The following XLST markup is added with the XML namespace prefix for the OSEB uos element included if used.

```
<xsl:template match="uos">
  <schema_instance express_schema_name="{@schema}">
   <xsl:apply-templates/>
  </schema_instance>
</xsl:template>
```

The XSLT stylesheet is then initialized with the XSLT markup necessary to map the document header from the the Object Serialization Early Binding header to the Late Binding header followed by the XSLT markup necessary to support the processing of Object Serialization Early Binding collections and nested collections. .

I.2.1 Schema independent XSLT markup for document header transformation

Since both bindings use the header specified in clause 6 , the XSLT markup required to map from the early bound document to the late bound document is defined as follows:

```
<xsl:template
match="iso_10303_28_header|data_section_header|documentation|purpose|tim
e_stamp|document_name|author|originating_organization|authorization|orig
inating_system|preprocessor_version">
 <xsl:copy>
  <xsl:apply-templates/>
 </xsl:copy>
</xsl:template>

<xsl:template match="express_schema|external_refid|schema_decl">
 <xsl:copy>
```



```
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>

<xsl:template match="iso_10303_28">
  <iso_10303_28>
    <xsl:attribute name="version">PDTS</xsl:attribute>
    <xsl:attribute name="representation_category">LB</xsl:attribute>
    <xsl:apply-templates/>
  </iso_10303_28>
</xsl:template>

<xsl:template match="express_data">
  <express_data>
    <xsl:attribute name="id"><xsl:value-of select="@id"/></xsl:attribute>
    <xsl:attribute name="representation_category">LB</xsl:attribute>
    <xsl:apply-templates/>
  </express_data>
</xsl:template>

<xsl:template
match="external_refid/@id|external_refid/@xlink:type|external_refid/@xli
nk:href|external_refid/@xlink:arcrole|external_refid/@xlink:title|extern
al_refid/@xlink:role|external_refid/@xlink:show|external_refid/@xlink:ac
tuate">
  <xsl:copy/>
</xsl:template>

<xsl:template
match="express_schema/@id|express_schema/@express_schema_description|exp
ress_schema/@express_schema_identifier|express_schema/@express_schema_ve
rsion">
  <xsl:copy/>
</xsl:template>
```

I.2.2 Schema independent XSLT markup for Object Serialization Early Binding-specific element transformation

The XSLT stylesheet is initialized with the markup for a named template called `process_ctn` that is called when it is necessary to transform Object Serialization Early Binding collections into their Late Bound representation. The markup for `process_ctn` is defined as:

```
<xsl:template name="process_ctn">
  <xsl:param name="lbtype"/>
  <xsl:param name="deftype">none</xsl:param>
  <xsl:param name="stext"/>
  <xsl:variable name="ntext" select="concat(normalize-space($stext),'
')"/>
    <xsl:choose>
      <xsl:when test="$lbtype='entity_instance_ref'">
        <xsl:variable name="pid" select="substring-before($ntext,' ')"/>
```



```
     <xsl:choose>
      <xsl:when test="local-name(id($pid))='unset'">
  <unset></unset>
      </xsl:when>
      <xsl:otherwise>
       <entity_instance_ref>
   <xsl:attribute name="refid">
    <xsl:value-of select="$pid"/>
   </xsl:attribute>
  </entity_instance_ref>
      </xsl:otherwise>
     </xsl:choose>
   </xsl:when>
   <xsl:when test="$lbtype='string_literal' or
$lbtype='binary_literal'">
    <xsl:choose>
      <xsl:when test="$deftype='none'">
         <xsl:element name="{$lbtype}">
           <xsl:value-of select="id(substring-before($ntext,' '))"/>
         </xsl:element>
      </xsl:when>
      <xsl:otherwise>
       <xsl:element name="type_literal">
   <xsl:attribute name="express_type_name"><xsl:value-of
select="$deftype"/></xsl:attribute>
         <xsl:element name="{$lbtype}">
          <xsl:value-of select="id(substring-before($ntext,' '))"/>
         </xsl:element>
        </xsl:element>
       </xsl:otherwise>
      </xsl:choose>
   </xsl:when>
   <xsl:when test="$lbtype='select'">
        <xsl:apply-templates select="id(substring-before($ntext,' '))"
mode="xselect"/>
   </xsl:when>
   <xsl:otherwise>
     <xsl:choose>
      <xsl:when test="$deftype='none'">
         <xsl:element name="{$lbtype}">
    <xsl:value-of select="substring-before($ntext,' ')"/>
         </xsl:element>
      </xsl:when>
      <xsl:otherwise>
       <xsl:element name="type_literal">
  <xsl:attribute name="express_type_name"><xsl:value-of
select="$deftype"/></xsl:attribute>
         <xsl:element name="{$lbtype}">
    <xsl:value-of select="substring-before($ntext,' ')"/>
         </xsl:element>
       </xsl:element>
      </xsl:otherwise>
     </xsl:choose>
   </xsl:otherwise>
  </xsl:choose>
```



```
  <xsl:if test="substring-after($ntext,' ')">
    <xsl:call-template name="process_ctn">
      <xsl:with-param name="lbtype"><xsl:value-of
select="$lbtype"/></xsl:with-param>

 <xsl:with-param name="deftype"><xsl:value-of
select="$deftype"/></xsl:with-param>
 <xsl:with-param name="stext" select="substring-after($ntext,' ')"/>
 </xsl:call-template>
  </xsl:if>
</xsl:template>
```

The XSLT stylesheet is initialized with the markup for a named template, `process_nested_ctn`, that is called when it is necessary to transform Object Serialization Early Binding representations of nested collections into their Late Bound representation. The markup for `process_nested_ctn` is defined as:

```
<xsl:template name="process_nested_ctn">
 <xsl:param name="inner_aggs"/>
 <xsl:param name="lbtype"/>
 <xsl:param name="deftype">none</xsl:param>
 <xsl:param name="stext"/>
 <xsl:variable name="naggs" select="concat(normalize-
space($inner_aggs),' ')"/>
 <xsl:variable name="ntext" select="concat(normalize-space($stext),'
')"/>
 <xsl:choose>
    <xsl:when test="substring-after($naggs,' ')">
              <xsl:element name="{substring-before($naggs,' ')}">
        <xsl:call-template name="process_nested_ctn">
        <xsl:with-param name="lbtype"><xsl:value-of
select="$lbtype"/></xsl:with-param>
    <xsl:with-param name="inner_aggs"><xsl:value-of select="substring-
after($naggs,' ')"/></xsl:with-param>
        <xsl:with-param name="stext"><xsl:value-of
select="id(substring-before($ntext,' '))/@c"/></xsl:with-param>
     </xsl:call-template>
              </xsl:element>
   </xsl:when>
   <xsl:otherwise>
           <xsl:element name="{substring-before($naggs,' ')}">
            <xsl:variable name="ctntext" select="substring-
before($ntext,' ')"/>
     <xsl:call-template name="process_ctn">
               <xsl:with-param name="lbtype"><xsl:value-of
select="$lbtype"/></xsl:with-param>
          <xsl:with-param name="deftype"><xsl:value-of
select="$deftype"/></xsl:with-param>
          <xsl:with-param name="stext" select="id($ctntext)/@c"/>
     </xsl:call-template>
             </xsl:element>
    </xsl:otherwise>
      </xsl:choose>
```



```
      <xsl:if test="substring-after($ntext,' ')">
       <xsl:call-template name="process_nested_ctn">
         <xsl:with-param name="lbtype"><xsl:value-of
select="$lbtype"/></xsl:with-param>
    <xsl:with-param name="inner_aggs"><xsl:value-of
select="$inner_aggs"/></xsl:with-param>
         <xsl:with-param name="stext"><xsl:value-of select="substring-
after($ntext,' ')"/></xsl:with-param>
      </xsl:call-template>
      </xsl:if>
</xsl:template>
```

In addition, the XSLT style sheet is initialized with the following template to prevent OSEB XML elements with a content model, such as string or binary from inadvertently being output into the Late Bound representation by the default processing rules of XSLT.

```
<xsl:template match="*">

</xsl:template>
```

I.3 Schema dependent XSLT markup generation

The remaining XSLT markup to be added to the XSLT stylesheet is the result of processing the EXPRESS SELECT types and EXPRESS entity data types and their corresponding attributes in the EXPRESS schema.

I.3.1 XSLT markup for EXPRESS SELECT datatypes

An XSLT template is generated for each EXPRESS SELECT type defined in the EXPRESS schema. The template has the form:

```
<xsl:template match="oseb_select_name" mode="xselect">
  <type_literal express_type_name="express_select_name">
 <xsl:choose>
   <xsl:when test="contains(@utype,'oseb_select_choice1')">
       <!—-processing as defined in I.3.3 to I.3.9 goes here -->

   </xsl:when>

   <xsl:when test="contains(@utype,'oseb_select_choiceN')">
      <!—-processing as defined in I.3.3 to I.3.9 goes here -->
   </xsl:when>
   <xsl:otherwise>
   </xsl:otherwise>
 </xsl:choose>
  </type_literal>
</xsl:template>
```


where *exp_select_name* represents the name of the EXPRESS SELECT type, *oseb_select_name* represents the name of the element representing the EXPRESS SELECT type in the OSEB and *oseb_select_choice1*…*oseb_select_choiceN* represent the names of the choices in the element representing the EXPRESS SELECT type in the OSEB. The XSLT markup contained within each `xsl:when` element represents the processing of a single choice and is dependent on the underlying type the choice represents as specified in I.3.3 to I.3.9.

I.3.2 XSLT markup for EXPRESS entity datatypes

If an EXPRESS entity data type has no subtypes and it is not ABSTRACT, the XSLT required to map it to the Late Binding from the OSEB may be given as:

```
<xsl:template match="oseb_entity_name">
   <entity_instance  express_entity_name="exp_entity_name" id="{@x-id}">
…
 </entity_instance>
</xsl:template>
```

where *exp_entity_name* represents the name of the EXPRESS entity data type, *oseb_entity_name* represents the name of the element representing the EXPRESS entity data type in the OSEB DTD and the ellipses '…' represent the XSLT markup resulting from processing the EXPRESS attributes of the EXPRESS entity data type as specified in I.3.2.

If an EXPRESS entity data type is a subtype, the XSLT required to map it to the Late Binding from the OSEB may be given as:

```
<xsl:template match="oseb_entity_name">
  <entity_instance_as_group id="{@x-id}">
   <partial_entity_instance express_entity_name="exp_entity_name1">
         …
   </partial_entity_instance>

…
   <partial_entity_instance express_entity_name="exp_entity_nameN">
         …
   </partial_entity_instance>
 </entity_instance_as_group>
</xsl:template>
```

where *exp_entity_name1*…*express_entity_nameN* represent the names of the EXPRESS entity data types from which the subject EXPRESS entity data type inherits attributes, *oseb_entity_name* represents the name of the element representing the EXPRESS entity data type in the OSEB DTD and the small ellipses ('…') represent the XSLT markup resulting from processing the EXPRESS attributes belonging to each EXPRESS entity data type that makes up the subtype as specified in I.3.3 to I.3.9.

I.3.3 XSLT markup for EXPRESS attributes of primitive data types

If an EXPRESS attribute is of type INTEGER, REAL, NUMBER , BOOLEAN, or LOGICAL the XSLT markup required to map it to the Late Binding from the OSEB may be given as:

```
<attribute_instance express_attribute_name="exp_att_name">
   <primitive_type_label>
      <xsl:value-of select="@oseb_att_name"/>
   </primitive_type_label>
</attribute_instance>
```

where *exp_att_name* represents the name of the EXPRESS attribute, *oseb_att_name* represents the name XML attribute representing the EXPRESS attribute in the OSEB DTD, and <*primitive_type_label*> is dependent on the type of the EXPRESS attribute as follows:

<*primitive_type_label*> = <integer_literal> if type is INTEGER;
<*primitive_type_label*> = <real_literal> if type is REAL;
<*primitive_type_label*> = <real_literal> if type is NUMBER;
<*primitive_type_label*> = <boolean_literal> if type is BOOLEAN;
<*primitive_type_label*> = <logical_literal> if type is LOGICAL.

If an EXPRESS attribute is of type STRING or BINARY, the XSLT required to map it to the Late Binding from the OSEB may be given as:

```
<attribute_instance express_attribute_name="exp_att_name">
  <primitive_type_label>
    <xsl:value-of select="id(@oseb_att_name)"/>
  </primitive_type_label>
</attribute_instance>
```

where *exp_att_name* represents the name of the EXPRESS attribute, *oseb_att_name* represents the name of the XML attribute representing the EXPRESS attribute in the OSEB DTD, and <*primitive_type_label*> is dependent on the type of the EXPRESS attribute as follows:

<*primitive_type_label*> = <string_literal> if type is STRINGl
<*primitive_type_label*> = <binary_literal> if type is BINARY.

I.3.4 XSLT markup for EXPRESS attributes of simple defined types

If an EXPRESS attribute is a simple defined type with an underlying type of INTEGER, REAL, NUMBER , BOOLEAN, or LOGICAL the XSLT required to map it to the Late Binding from the OSEB may be given as:

```
<attribute_instance express_attribute_name="exp_att_name">
   <type_literal express_type_name="exp_simple_type_name">
      <simple_type_literal>
        <xsl:value-of select="id(@oseb_att_name)/@val"/>
      </simple_type_literal>
```



```
    </type_literal>
</attribute_instance>
```

where *exp_att_name* represents the name of the EXPRESS attribute, *expt_simple_type_name* represents the name of the simple defined type, *oseb_att_name* represents the name of the XML attribute representing the EXPRESS attribute in the OSEB DTD, and <*simple_type_literal*> is dependent on the type of the EXPRESS attribute as follows:

<*simple_type_literal*> = <integer_literal>, if the defined type is INTEGER,
<*simple_type_literal*> = <real_literal>, if the defined type is REAL,
<*simple_type_literal*> = <real_literal>, if the defined type is NUMBER,
<*simple_type_literal*> = <boolean_literal>, if the defined type is BOOLEAN,
<*simple_type_literal*> = <logical_literal>, if the defined type is LOGICAL.

If an EXPRESS attribute is a simple defined type with an underlying type of STRING or BINARY, the XSLT required to map it to the Late Binding from the OSEB may be given as:

```
<attribute_instance express_attribute_name="exp_att_name">
 <type_literal express_type_name="exp_simple_type_name">
    <simple_type_literal>
      <xsl:value-of select="id(@oseb_att_name)"/>
    </simple_type_literal>
 </type_literal>
</attribute_instance>
```

where *exp_att_name* represents the name of the EXPRESS attribute, *exp_simple_type_name* represents the defined type name of the EXPRESS attribute, *oseb_att_name* represents the name of the XML attribute representing the EXPRESS attribute in the OSEB DTD, and <*simple_type_literal*> is dependent on the type of the EXPRESS attribute as follows:

<*simple_type_literal*> = <string_literal> if the simple defined type is STRING,
<*simple_type_literal*> = <binary_literal> if the simple defined type is BINARY.

I.3.5 XSLT markup for EXPRESS attributes of EXPRESS ENUMERATION

If an EXPRESS attribute is of type ENUMERATION, the XSLT required to map it to the Late Binding from the OSEB is given by:

```
<attribute_instance express_attribute_name="exp_att_name">
   <type_literal express_type_name="exp_att_type_name">
      <enumeration_ref>
            <xsl:value-of select="id(@oseb_att_name)/@val"/>
       </enumeration_ref>
    </type_literal>
</attribute_instance>
```


where *exp_att_name* represents the name of the EXPRESS attribute, *exp_att_type_name* represents the type of the EXPRESS attribute and *oseb_att_name* represents the name of the XML attribute representing the EXPRESS attribute in the OSEB DTD.

I.3.6 XSLT markup for EXPRESS attributes of EXPRESS SELECT

If an EXPRESS attribute is of type SELECT type, it is mapped to the Late Binding from the OSEB by applying the template generated for its SELECT type as defined in I.3.1. The XSLT required to invoke the template may be given as:

```
<attribute_instance express_attribute_name="exp_select_att_name">
     <xsl:apply-templates select="id(@oseb_select_att_name)"
mode="xselect"/>

</attribute_instance>
```

where *exp_select_att_name* represents the name of the EXPRESS attribute and *oseb_select_att_name* represents the name of the XML representation of the SELECT type in the OSEB.

I.3.7 XSLT markup for EXPRESS attributes of entity data types

If an EXPRESS attribute is an EXPRESS entity data type, the XSLT required to map it to the Late Binding from the OSEB may be given as:

```
<attribute_instance express_attribute_name="exp_att_name">
  <entity_instance_ref  entity_instance_idref="{@oseb_att_name}" />
 </attribute_instance>
```

where *exp_att_name* represents the name of the EXPRESS attribute and *oseb_att_name* represents the name of the XML attribute representing the EXPRESS attribute in the OSEB DTD.

I.3.8 XSLT markup for EXPRESS attributes of aggregates (one-dimensional)

Single dimension collection types are mapped by invoking the named template `process_ctn`.

If an EXPRESS attribute is an EXPRESS aggregate type the XSLT required to map it to the Late Binding from the OSEB may be given as:

```
<attribute_instance express_attribute_name="exp_att_name">
   <aggregate_type_literal>
    <xsl:call-template name="process_ctn">
      <xsl:with-param name="lbtype">late_bound_type</xsl:with-param>

      <xsl:with-param name="deftype">exp_deftype_name</xsl:with-
param>
      <xsl:with-param name="stext" select="id(@oseb_att_name)/@c"/>
     </xsl:call-template>
  </aggregate_type_literal>
</attribute_instance>
```


where *exp_att_name* represents the name of the EXPRESS attribute, *oseb_att_name* represents the name of the XML attribute representing the EXPRESS attribute in the OSEB DTD, *aggregate_type_literal* is dependent on the aggregate type of the EXPRESS attribute as follows:

aggregate_type_literal = set_literal, if aggregate type is SET,
aggregate_type_literal = bag_literal, if attribute type is BAG,
aggregate_type_literal = array_literal, if attribute type is ARRAY,
aggregate_type_literal= list_literal, if attribute type is LIST.

The value of *late_bound_type* is dependent on the underlying type of the aggregate collection as follows:

For a collection of type EXPRESS entity data type, *late_bound_type*=entity_instance_ref
For a collection of type SELECT type, *late_bound_type*=select
For a collection of type ENUMERATION type, *late_bound_type*=enumeration_ref
For a collection of type STRING, *late_bound_type*=string_literal
For a collection of type REAL, *late_bound_type*=real_literal
For a collection of type INTEGER, *late_bound_type*=integer_literal
For a collection of type BOOLEAN, *late_bound_type*=boolean_literal
For a collection of type LOGICAL, *late_bound_type*=logical_literal
For a collection of type BINARY, *late_bound_type*=binary_literal
For a collection of type NUMBER , *late_bound_type*=number_literal

If the aggregate is of an EXPRESS named type then the value of *exp_deftype_name* shall be the name of the named type, otherwise the `<xsl:with-param name=”deftype”>` element shall not be present.

I.3.9 XSLT markup for EXPRESS attributes of Aggregates (multidimensional)

Multi-dimensional collection types are mapped by invoking the named template `process_nested_ctn`. If the type of an EXPRESS attribute is a multidimensional collection data type, the XSLT required to map it to the Late Binding from the OSEB may be given as:

```
<attribute_instance express_attribute_name="exp_att_name">
   <aggregate_type_literal>
     <xsl:call-template name="process_nested_ctn">
       <xsl:with-param name="lbtype">late_bound_type</xsl:with-param>
        <xsl:with-param name="deftype">exp_deftype_name</xsl:with-param>
          <xsl:with-param name="inner_aggs">agg_list</xsl:with-param>
          <xsl:with-param name="stext" select="id(@oseb_att_name)/@c"/>
        </xsl:call-template>
   </aggregate_type_literal>
</attribute_instance>
```

where *exp_att_name* represents the name of the EXPRESS attribute and *oseb_att_name* represents the name of the XML attribute representing the EXPRESS attribute in the OSEB DTD.

The value of *aggregate_type_literal* represents the type of the outermost aggregation and follows the same mapping defined for single dimension collections in I.3.8. The representation of *late_bound_type* and *exp_deftype_name* is the same as that defined for single dimension aggregations defined in I.3.8.

The value of the `inner_aggs` parameter is represented by *agg_list* and is represented as a list of whitespace seperated *aggregate_type_literals* starting with the outermost aggregate to the left followed by the next deepest aggregate in the nesting, etc. The value of each whitespace separated aggregate type follows the same mapping rules defined for *aggregate_type_literal* given in I.3.8.

EXAMPLE - For the EXPRESS attribute declaration:

```
transform_matrix : ARRAY [1:3] OF LIST [1:3] OF ARRAY [1:3];
```

The XSLT containing the value of the *inner_aggs* would be

```
<xsl:with-param name="inner_aggs"> list_literal
array_literal</xsl:with-param>
```

I.4 XSLT stylesheet termination

In order to ensure that the stylesheet is well-formed, the XSLT stylesheet must be terminated with closing stylesheet element tag upon completion of the schema dependent processing of the EXPRESS schema defined in I.3. The final element tag is specified as:

```
</xsl:stylesheet>
```


Annex J
(informative)

Mapping of EXPRESS-Typed Early Binding to Late Binding using XSLT

J.1 Introduction

This annex specifies two XLST stylesheets that process an input ETEB XML document and creates a LB XML document containing the same data. The stylesheets are schema-independent and operate on any ETEB XML document.

The first stylesheet is a simple conversion script that uses the #FIXED attributes in the document's DTD. The second stylesheet is based on architectural processing. Both stylesheets produce the same results.

J.2 Simple stylesheet

The following is the simple XSLT stylesheet for producing a LB XML document from an ETEB XML document.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes" encoding="iso-8859-1"/>

<xsl:template match="/">
<iso_10303_28>
  <xsl:apply-templates/>
</iso_10303_28>
</xsl:template>

<xsl:template match="iso_10303_28_header">
  <xsl:copy>
  <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>

<xsl:template match="iso_10303_28_header//*">
  <xsl:copy>
  <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>

<xsl:template match="external_refid">
  <xsl:copy>
  <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>

<xsl:template match="external_refid//*">
```



```
 <xsl:copy>
 <xsl:apply-templates/>
 </xsl:copy>
</xsl:template>

<xsl:template match="express_data">
 <xsl:copy>
 <xsl:apply-templates/>
 </xsl:copy>
</xsl:template>

<xsl:template match="data_section_header">
    <xsl:copy>
 <xsl:apply-templates/>
    </xsl:copy>
</xsl:template>

<xsl:template match="data_section_header//*">
 <xsl:copy>
 <xsl:apply-templates/>
 </xsl:copy>
</xsl:template>

<xsl:template match="*[@late-bound-element='schema_instance']">
 <schema_instance>
  <xsl:copy-of select='@express_schema_name'/>
  <xsl:copy-of select='@id'/>
 <xsl:apply-templates/>
 </schema_instance>
</xsl:template>

<xsl:template match="*[@late-bound-element='entity_instance']">
 <entity_instance>
  <xsl:copy-of select='@express_entity_name'/>
  <xsl:copy-of select='@id'/>
  <xsl:copy-of select='@express_schema_name'/>
 <xsl:apply-templates/>
 </entity_instance>
</xsl:template>

<xsl:template match="*[@late-bound-element='attribute_instance']">
 <attribute_instance>
  <xsl:copy-of select='@express_attribute_name'/>
  <xsl:apply-templates/>
 </attribute_instance>
</xsl:template>

<xsl:template match="*[@late-bound-element='entity_instance_as_group']">
 <entity_instance_as_group>
  <xsl:copy-of select='@id'/>
 <xsl:apply-templates/>
 </entity_instance_as_group>
</xsl:template>
<xsl:template match="*[@late-bound-element='partial_entity_instance']">
 <partial_entity_instance>
```



```
   <xsl:copy-of select='@express_entity_name'/>
   <xsl:copy-of select='@id'/>
  <xsl:apply-templates/>
  </partial_entity_instance>
</xsl:template>

<xsl:template match="*[contains(local-name(.),'-ref')]">
  <entity_instance_ref>
   <xsl:copy-of select='@refid'/>
  </entity_instance_ref>
</xsl:template>

<xsl:template match="*[@late-bound-element='set_literal']">
  <set_literal>
  <xsl:apply-templates/>
  </set_literal>
</xsl:template>

<xsl:template match="*[@late-bound-element='bag_literal']">
  <bag_literal>
  <xsl:apply-templates/>
  </bag_literal>
</xsl:template>

<xsl:template match="*[@late-bound-element='list_literal']">
  <list_literal>
  <xsl:apply-templates/>
  </list_literal>
</xsl:template>

<xsl:template match="*[@late-bound-element='array_literal']">
  <array_literal>
  <xsl:apply-templates/>
  </array_literal>
</xsl:template>


<xsl:template match="*[@late-bound-element='type_literal']">
  <type_literal>
   <xsl:copy-of select='@express_type_name'/>
  <xsl:apply-templates/>
  </type_literal>
</xsl:template>

<xsl:template match="string">
  <string_literal>
   <xsl:value-of select="."/>
  </string_literal>
</xsl:template>

<xsl:template match="real">
  <real_literal>
   <xsl:value-of select="."/>
  </real_literal>
</xsl:template>
```



```
<xsl:template match="integer">
 <integer_literal>
  <xsl:value-of select="."/>
 </integer_literal>
</xsl:template>

<xsl:template match="boolean">
 <boolean_literal>
  <xsl:value-of select="."/>
 </boolean_literal>
</xsl:template>

<xsl:template match="logical">
 <logical_literal>
  <xsl:value-of select="."/>
 </logical_literal>
</xsl:template>

<xsl:template match="true">
 <xsl:copy/>
</xsl:template>

<xsl:template match="false">
 <xsl:copy/>
</xsl:template>

<xsl:template match="unset">
 <xsl:copy/>
</xsl:template>

</xsl:stylesheet>
```

J.3 Architectural stylesheet

The following is the architectural XSLT stylesheet for producing a LB XML document from an ETEB XML document.

```
<?xml version="1.0"?>

<!-- APEX (Architectural Processor Employing XSLT), an XSLT stylesheet
for creating an architectural document from a client document.

     Author: Joshua Lubell
             lubell@nist.gov

     $Id: apex.xsl,v 1.9 2000/07/07 18:29:27 lubell Exp $

THIS STYLESHEET HAS NOT BEEN APPROVED BY NIST FOR PUBLIC DISTRIBUTION!

APEX implements a simple subset of the Architectural Form Definition
Requirements (AFDR) specified in Annex A.3 of ISO 10744:1997. APEX
```



```
behaves similarly to David Megginson's XAF package for Java
(http://www.megginson.com/XAF/) and differs from the AFDR in the same
ways as XAF. In addition, APEX differs from the AFDR as follows:

1. Bridge forms are not supported.

2. The architecture use declaration processing instruction is
ingored. Information about an architecture is instead conveyed to APEX
through stylesheet parameters.

The APEX stylesheet requires the following parameter:

name - the name of the architecture being processed

The following optional parameters correspond to pseudo-attributes from
the architecture use declaration:

form-att
auto
renamer-att
suppressor-att
ignore-data-att

The following parameter is also optional:

strip - a list of attributes to be stripped during architectural
processing, specified as a space-delimited string. This is useful for
suppressing architectural support attributes from architectures other
than the architecture being processed. For example, if your XML
document is a client of architectures "foo" and "bar", you want to
create an architectural document for "foo", and your document contains
"bar" form attributes and "bar-atts" renamer attributes for the bar
architecture, you would specify the value "bar bar-atts" for the strip
parameter.

To use APEX, simply process your client document using the apex.xsl
stylesheet and your favorite XSLT processor. Supply a value for the
required "name" parameter as well as any desired optional parameters.

-->

<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <!-- Global parameters -->

  <!-- name of the architecture being processed -->
  <xsl:param name="name"/>

  <!-- values from the architecture use declaration -->
  <xsl:param name="form-att" select="''"/>
  <xsl:param name="renamer-att" select="''"/>
  <xsl:param name="suppressor-att" select="''"/>
  <xsl:param name="ignore-data-att" select="''"/>
  <xsl:param name="auto" select="''"/>
```



```
  <!-- space-delimited list of support attributes from architectures
other than the architecture being processed -->
  <xsl:param name="strip" select="''"/>

  <!-- Global variables -->

  <!-- form-att-name
       set to value of form-att if specified in architecture use
declaration; otherwise set to architecture name -->
  <xsl:variable name="form-att-name">
    <xsl:choose>
      <xsl:when test="$form-att=''">
        <xsl:value-of select="$name"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$form-att"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <!-- auto-map
       set to value of auto if specified in architecture use
declaration; otherwise set to ArcAuto -->
  <xsl:variable name="auto-map">
    <xsl:choose>
      <xsl:when test="$auto=''">
        <xsl:value-of select="'ArcAuto'"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$auto"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <!-- Template rules -->

  <!-- Template rule parameters:

       suppress-state - specifies whether to suppress architectural
processing; value determined from suppressor architecture use
declaration attribute; default is 'sArcNone'

       ignore-state - specifies whether to ignore data; value
determined from ignore data architecture use declaration attribute;
default is 'cArcIgnD' -->

  <xsl:template match="/">
    <xsl:apply-templates>
      <xsl:with-param name="suppress-state" select="'sArcNone'"/>
      <xsl:with-param name="ignore-state" select="'cArcIgnD'"/>
    </xsl:apply-templates>
  </xsl:template>

  <xsl:template match="*">
```



```
    <!-- Unless architectural processing is irrevocably suppressed,
update suppress-state and ignore-state, process the element arcording
to the old suppress-state, and process the element's content using the
updated suppress-state and ignore-state. -->
    <xsl:param name="suppress-state"/>
    <xsl:param name="ignore-state"/>
    <xsl:choose>
      <xsl:when test="$suppress-state='sArcNone'">
        <!-- Update suppress-state if a suppressor attribute value is
specified and ignore-state if an ignore data attribute is
specified. Process the element. Process the element's content
according to the updated suppress-state and ignore-state. -->
        <xsl:call-template name="process-element">
          <xsl:with-param name="suppress-state">
            <xsl:call-template name="set-suppress-state">
              <xsl:with-param name="suppress-state"
           select="$suppress-state"/>
            </xsl:call-template>
          </xsl:with-param>
          <xsl:with-param name="ignore-state">
            <xsl:call-template name="set-ignore-state">
              <xsl:with-param name="ignore-state"
           select="$ignore-state"/>
            </xsl:call-template>
          </xsl:with-param>
        </xsl:call-template>
      </xsl:when>
      <xsl:when test="$suppress-state='sArcForm'">
        <!-- Update suppress-state if a suppressor attribute value is
specified and ignore-state if an ignore data attribute is
specified. Don't process the element. Process the element's content
using the updated suppress-state and ignore-state. -->
        <xsl:call-template name="apply-data-only">
          <xsl:with-param name="suppress-state">
            <xsl:call-template name="set-suppress-state">
              <xsl:with-param name="suppress-state"
           select="$suppress-state"/>
            </xsl:call-template>
          </xsl:with-param>
          <xsl:with-param name="ignore-state">
            <xsl:call-template name="set-ignore-state">
              <xsl:with-param name="ignore-state"
           select="$ignore-state"/>
            </xsl:call-template>
          </xsl:with-param>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
         <!-- suppress-state is sArcAll, so don't do any processing at
all. -->
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template match="text()">
```



```
    <!-- Copy the data if architectural processing is not being
suppressed and the ignore-state is nArcIgnD. If architectural
processing is not being suppressed and the ignore-state is cArcIgnD,
copy the data only the parent element is architectural. The parent
element is architectural if it has a form attribute with a value other
than '#IMPLIED', or if element-type names are being derived
automatically and there is no form attribute with the value
'#IMPLIED'. -->
    <xsl:param name="suppress-state"/>
    <xsl:param name="ignore-state"/>
    <xsl:if test="$suppress-state='sArcNone'">
      <xsl:choose>
        <xsl:when test="$ignore-state='nArcIgnD'">
          <xsl:value-of select="."/>
        </xsl:when>
        <xsl:when test="$ignore-state='cArcIgnD'">
          <xsl:choose>
            <xsl:when test="../@*[name()=$form-att-name] and
not(../@*[name()=$form-att-name]='#IMPLIED')">
              <xsl:value-of select="."/>
            </xsl:when>
            <xsl:when test="$auto-map='ArcAuto' and
             not(../@*[name()=$form-att-name])">
              <xsl:value-of select="."/>
            </xsl:when>
          </xsl:choose>
        </xsl:when>
      </xsl:choose>
    </xsl:if>
  </xsl:template>

  <xsl:template match="@*">
    <!-- Attributes are architectural unless they are mapped to
'#DEFAULT' by an attribute renamer. Therefore, if the attribute has no
renamer sibling, copy it. If a renamer sibling exists, inspect the
renamer attribute's value. If the attribute being processed does not
appear as a client attribute name in the list of pairs assigned to the
renamer attribute, copy the attribute unmodified. If the attribute
being processed appears maps to an architectural attribute, substitute
the architectural attribute name for the client attribute name. If the
attribute being processed maps to '#DEFAULT', don't copy the
attribute. -->
    <xsl:param name="suppress-state"/>
    <xsl:param name="ignore-state"/>
    <xsl:choose>
      <xsl:when test="../@*[name()=$renamer-att]">
        <xsl:variable name="newname">
          <xsl:call-template name="rename">
            <xsl:with-param name="attr" select="name()"/>
            <xsl:with-param name="list" select="../@*[name()=$renamer-
att]"/>
          </xsl:call-template>
        </xsl:variable>
        <xsl:choose>
          <xsl:when test="$newname=''">
```



```
            <xsl:call-template name="copy-att"/>
          </xsl:when>
          <xsl:when test="$newname='#DEFAULT'"/>
          <xsl:otherwise>
            <xsl:attribute name="{$newname}">
              <xsl:value-of select="string(.)"/>
            </xsl:attribute>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <xsl:otherwise>
        <xsl:call-template name="copy-att"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <!-- Named templates -->

  <xsl:template name="copy-att">
    <!-- Copy the attribute currently being processed unless it is an
architecture use declaration attribute or is specified in the 'strip'
stylesheet parameter. -->
    <xsl:variable name="is-arch-use-attr">
      <xsl:call-template name="member">
        <xsl:with-param name="item" select="name()"/>
        <xsl:with-param name="list" select="$strip"/>
      </xsl:call-template>
    </xsl:variable>
    <xsl:if test=
"not(name()= $is-arch-use-attr or name()=$form-att-name or name()=$renamer-
att or name()=$suppressor-att or name()=$ignore-data-att)">
      <xsl:copy/>
    </xsl:if>
  </xsl:template>

  <xsl:template name="apply-all">
    <!-- Process an element's attributes and content. -->
    <xsl:param name="suppress-state"/>
    <xsl:param name="ignore-state"/>
    <xsl:apply-templates select="*|@*|text()">
      <xsl:with-param name="suppress-state" select="$suppress-state"/>
      <xsl:with-param name="ignore-state" select="$ignore-state"/>
    </xsl:apply-templates>
  </xsl:template>

  <xsl:template name="apply-data-only">
    <!-- Process an element's content. -->
    <xsl:param name="suppress-state"/>
    <xsl:param name="ignore-state"/>
    <xsl:apply-templates select="*|text()">
      <xsl:with-param name="suppress-state" select="$suppress-state"/>
      <xsl:with-param name="ignore-state" select="$ignore-state"/>
    </xsl:apply-templates>
  </xsl:template>
```



```
  <xsl:template name="deep-copy">
    <!-- Copy an element, and process its attributes and content. -->
    <xsl:param name="suppress-state"/>
    <xsl:param name="ignore-state"/>
    <xsl:copy>
      <xsl:call-template name="apply-all">
        <xsl:with-param name="suppress-state" select="$suppress-state"/>
        <xsl:with-param name="ignore-state" select="$ignore-state"/>
      </xsl:call-template>
    </xsl:copy>
  </xsl:template>

  <xsl:template name="deep-copy-data-only">
    <!-- Copy an element, and process its content. -->
    <xsl:param name="suppress-state"/>
    <xsl:param name="ignore-state"/>
    <xsl:copy>
      <xsl:call-template name="apply-data-only">
        <xsl:with-param name="suppress-state" select="$suppress-state"/>
        <xsl:with-param name="ignore-state" select="$ignore-state"/>
      </xsl:call-template>
    </xsl:copy>
  </xsl:template>

  <xsl:template name="process-element">
    <!-- Perform architectural processing on the current
element. Process its content as per the suppress-state and
ignore-state. -->
    <xsl:param name="suppress-state"/>
    <xsl:param name="ignore-state"/>
    <xsl:choose>
      <xsl:when test="$auto-map='ArcAuto' and not(@*[name()=$form-att-
name])">
        <!-- Element names are derived automatically, and no form
attribute exists. Copy the element and process its attributes and
content. -->
        <xsl:call-template name="deep-copy">
          <xsl:with-param name="suppress-state"
      select="$suppress-state"/>
          <xsl:with-param name="ignore-state" select="$ignore-state"/>
        </xsl:call-template>
      </xsl:when>
      <xsl:when test=
"@*[name()=$form-att-name]='#IMPLIED' or ($auto-map='nArcAuto' and
not(@*[name()=$form-att-name]))">
        <!-- A form attribute maps this element to '#IMPLIED', or
automatic element-type name derivation is turned off and no mapping
is specified for this element. Don't include this element in the
architectural document, but process its content. -->
        <xsl:call-template name="apply-data-only">
          <xsl:with-param name="suppress-state"
      select="$suppress-state"/>
          <xsl:with-param name="ignore-state" select="$ignore-state"/>
        </xsl:call-template>
      </xsl:when>
```



```
      <xsl:when test="@*[name()=$form-att-name]">
        <!-- A form attribute maps this element to an architectural
element, so substitute the architectural name for the client
name. Process the element's attributes and content. -->
        <xsl:element name="{@*[name()=$form-att-name]}">
          <xsl:call-template name="apply-all">
            <xsl:with-param name="suppress-state"
        select="$suppress-state"/>
            <xsl:with-param name="ignore-state" select="$ignore-state"/>
          </xsl:call-template>
        </xsl:element>
      </xsl:when>
    </xsl:choose>
  </xsl:template>

  <xsl:template name="set-suppress-state">
    <!-- If the element currently being processed has a suppressor
attribute whose value is not '#IMPLIED', set suppress-state to this
value. -->
    <xsl:param name="suppress-state"/>
    <xsl:choose>
      <xsl:when test=
"@*[name()=$suppressor-att] and not(@*[name()=$suppressor-
att]='#IMPLIED')">
        <xsl:value-of select="@*[name()=$suppressor-att]"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$suppress-state"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template name="set-ignore-state">
    <!-- If the element currently being processed has an ignore data
attribute whose value is not '#IMPLIED', set ignore-state to this
value. -->
    <xsl:param name="ignore-state"/>
    <xsl:choose>
      <xsl:when test=
"@*[name()=$ignore-data-att] and not(@*[name()=$ignore-data-
att]='#IMPLIED')">
        <xsl:value-of select="@*[name()=$ignore-data-att]"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$ignore-state"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template name="rename">
    <!-- Given the name of an attribute from the client document and a
list of pairs associating client attribute names with architectural
attribute names, return the architectural attribute name associated
with the client attribute. If the client attribute name does not
appear in the list of pairs, return ''. -->
```



```
    <xsl:param name="attr"/>
    <xsl:param name="list"/>
    <xsl:variable name="normlist">
      <xsl:value-of select="normalize-space($list)"/>
    </xsl:variable>
    <xsl:choose>
      <xsl:when test="$normlist=''">
        <xsl:value-of select="''"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:variable name="normlist-1">
          <xsl:value-of select="substring-after($normlist,' ')"/>
        </xsl:variable>
        <xsl:variable name="clientattr">
          <xsl:value-of select="substring-before($normlist,' ')"/>
        </xsl:variable>
        <xsl:variable name="archattr">
          <xsl:variable name="temp"
    select="substring-before($normlist-1,' ')"/>
            <xsl:choose>
              <xsl:when test="$temp=''">
                <xsl:value-of select="$normlist-1"/>
              </xsl:when>
              <xsl:otherwise>
                <xsl:value-of select="$temp"/>
              </xsl:otherwise>
            </xsl:choose>
        </xsl:variable>
        <xsl:choose>
          <xsl:when test="$clientattr=$attr">
            <xsl:value-of select="$archattr"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:call-template name="rename">
              <xsl:with-param name="attr" select="$attr"/>
              <xsl:with-param name="list"
           select="substring-after($normlist-1,' ')"/>
            </xsl:call-template>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template name="member">
    <!-- Determine whether 'item', a string, is a member of 'list', a
string whose value is a space-delimited list of substrings. Return
item if it is a member; an empty string if it is not. -->
    <xsl:param name="item"/>
    <xsl:param name="list"/>
    <xsl:choose>
      <xsl:when test="$list=''">
        <xsl:value-of select="''"/>
      </xsl:when>
      <xsl:otherwise>
```



```
        <xsl:variable name="normlist">
          <xsl:value-of select="normalize-space($list)"/>
        </xsl:variable>
        <xsl:variable name="rest">
          <xsl:value-of select="substring-after($normlist,' ')"/>
        </xsl:variable>
        <xsl:variable name="first">
          <xsl:choose>
            <xsl:when test="contains($normlist,' ')">
              <xsl:value-of select="substring-before($normlist,' ')"/>
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="$normlist"/>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:variable>
        <xsl:choose>
          <xsl:when test="$first=$item">
            <xsl:value-of select="$item"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:call-template name="member">
              <xsl:with-param name="item" select="$item"/>
              <xsl:with-param name="list" select="$rest"/>
            </xsl:call-template>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```


Annex K
(informative)

Use of URNs in XML Namespace fully qualified names

This annex recommends an approach for standardizing URNs for use in XML Namespace fully qualified names.

For the OSEB, the XML Namespace URI, which represents the non-local part of the qualified name, could consist of the concatenation of the string “urn:iso10303-28:oseb/” and the name of the EXPRESS schema that contains the named element with the first character uppercase and all other characters lowercase.

EXAMPLE - The following example illustrates a mapping from an EXPRESS entity name defined in a schema to a fully qualified XML name.

In EXPRESS:

```
SCHEMA ABCD;
   ENTITY saMPLE_Name;
   END_ENTITY;
END_SCHEMA;
```

In XML instance data:

```
<Sample_name x-id = “i1”  xmlns="urn:iso10303-28:oseb/Abcd”>
```


Bibliography

[1] ISO 10303-21:1994, *Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure*.

[2] ISO 10303-22: 1999, *Industrial automation systems and integration – Product data representation and exchange – Part 22: Implementation method: Standard data access interface specification*.

[3] *XML Schema*. World Wide Web Consortium Working Draft 07 April 2000 [cited 2000-08-11]. Available from World Wide Web: < http://www.w3.org/TR/xmlschema-1/ >.

[4] *Structuring XML Documents*. by David Megginson, Chapters 10 and 11, Prentice Hall ISBN 0-13-642299-3, 1998.

Index

